

# Composing Heterogeneous Reactive Systems

ALBERT BENVENISTE and BENOÎT CAILLAUD

Irisa/Inria

LUCA P. CARLONI

Columbia University

PAUL CASPI

Verimag

and

ALBERTO L. SANGIOVANNI-VINCENTELLI

University of California, Berkeley

---

We present a compositional theory of heterogeneous reactive systems. The approach is based on the concept of tags marking the events of the signals of a system. Tags can be used for multiple purposes from indexing evolution in time (time stamping) to expressing relations among signals, like coordination (e.g., synchrony and asynchrony) and causal dependencies. The theory provides flexibility in system modeling because it can be used both as a unifying mathematical framework to relate heterogeneous models of computations and as a formal vehicle to implement complex systems by combining heterogeneous components. In particular, we introduce an algebra of tag structures to define heterogeneous parallel composition formally. Morphisms between tag structures are used to define relationships between heterogeneous models at different levels of abstraction. In particular, they can be used to represent design transformations from tightly synchronized specifications to loosely-synchronized implementations. The theory has an important application in the correct-by-construction deployment of synchronous design on distributed architectures.

Categories and Subject Descriptors: C.3.0 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Interactive and reactive computation*

General Terms: Design, Theory

Additional Key Words and Phrases: Compositionality, correct-by-construction design, GALS, models of computation, reactive systems

---

This research was supported in part by the European Commission under the projects COLUMBUS, IST-2002-38314, ARTIST, IST-2001-34820, and IST-004527 ARTIST2 Network of Excellence on Embedded Systems Design, by the NSF under the project ITR (CCR-0225610), and by the GSRC. Authors' addresses: benveniste@irisa.fr.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org). © 2008 ACM 1539-9087/2008/07-ART43 \$5.00 DOI 10.1145/1376804.1376811 <http://doi.acm.org/10.1145/1376804.1376811>

ACM Transactions on Embedded Computing Systems, Vol. 7, No. 4, Article 43, Publication date: July 2008.

**ACM Reference Format:**

Benveniste, A., Caillaud, B., Carloni, L. P., Caspi, P., Sangiovanni-Vincentelli, A. L. 2008. Composing heterogeneous reactive systems. *ACM Trans. Embedd. Comput. Syst.* 7, 4, Article 43 (July 2008), 36 pages. DOI = 10.1145/1376804.1376811 <http://doi.acm.org/10.1145/1376804.1376811>

---

## 1. INTRODUCTION

Heterogeneity is a typical characteristic of embedded systems. It manifests itself naturally at the component level where different models of computation may be used to represent the operations of the various components, for example, when a digital controller is applied to a continuous-time plant. In addition, heterogeneity may appear across different levels of abstraction in the design process because of different modeling goals. For instance, designers may decide to rely on the rich set of properties of synchronous languages in the specification phase of a system and then proceed with an implementation based on a globally asynchronous locally synchronous (GALS) architecture. Dealing with heterogeneity is quite often problematic. The composition of heterogeneous models is, in general, not well defined and it is often impossible to determine its properties from the known properties of the components. When heterogeneity appears during the design process across different layers of abstraction, it is difficult to assess whether the lower level of abstraction maintains certain properties of the higher level. The main cause of this difficulty is the lack of an all-encompassing mathematical framework for reasoning about heterogeneous composition.

In our view, there are actually three kinds of heterogeneity: one that deals with different components that have different models of computation at the same abstraction level; one that deals with different models that capture different aspects of the behavior of a *single component*; and, finally, one that applies to models at different levels of abstraction. Figure 1 offers a conceptual view of this concept of heterogeneity. A system is represented by Diagram (b) as a collection of interacting components. Each component can be represented with models of computation that have different properties and semantics. For example, one of the components may be expressed as a finite-state machine (the light block in the figure) and others (the dark blocks) as synchronous data flow. These models are intrinsically different so that making them communicate is, at best, problematic, unless the model of the interaction is explicitly stated perhaps in a third model of computation. In addition to this heterogeneity that has been analyzed in a number of papers and environments, we may have heterogeneity where, for each of component multiple coexisting models of computation and communication (MoCC) can be used to reason formally, or semiformally, on different aspects of their behavior. This is illustrated in diagram (a), which, for instance, from left to right, shows: a functional model, used for formal verification or functional simulation purposes, and a timed model, used for timing analysis.

While we would generally rather build and consider these three types of models separately in order to reason separately on these design issues (principle of orthogonalization of concerns), they do typically interact. For example, certain

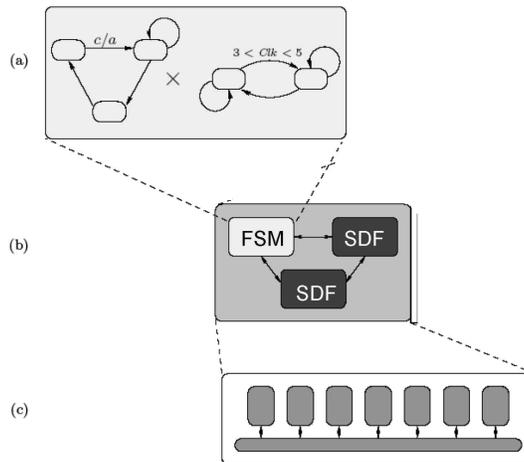


Fig. 1. Motivation: coping with heterogeneous modeling.

failure propagation scenarios are not possible in certain operating modes, for example, a sleeping device is not expected to fail, or the performance of a component may vary based on its functional behavior, e.g., the speed of a component processing a video data stream depends on the particular sequence of input frames. In these cases, there is a need for a heterogeneous behavioral modeling technique that makes it possible to reason formally on the combination of these models, that is, their “product” (represented by the “ $\times$ ” symbol in Figure 1).

Finally, diagram (c) shows an example of vertical heterogeneity. In this case, the system specification may be represented with a different model of computation from the one of an execution platform, for example, by deploying embedded software on programmable components communicating via a shared bus. The platform is captured by a MoCC, which is motivated by architectural concerns, for example, modularity and compartmentalization aiming at robustness against faults. Also, the platform performance model will consist of worst-case execution time (WCET) characteristics, different from the timing requirements pertaining to the application. Again, to have a sound progress from diagram (b) to diagram (c), we need a notion of behavioral heterogeneous product of systems that makes it possible to: Seamlessly integrate different models, guarantee properties by construction, reduce cost verification (by not expanding the MoCC into low level detailed models), and efficiently perform the deployment analysis.

In this article, we address these fundamental problems involving the composition of heterogeneous models and we propose a mathematical framework for modeling heterogeneous reactive systems that provides a solid foundation to handle formally communication and coordination among heterogeneous components. Interesting work along similar lines has been the Ptolemy project [Eker et al. 2003; Lee and Xiong 2001], the MoBIES project [Alur et al. 2003], the model-integrated computing (MIC) framework [Karsai et al. 2003], the Metropolis project [Balarin et al. 2003; Burch et al. 2001], *Interface Theories* [de Alfaro and Henzinger 2001], the concepts of *endochrony* and *isochrony* [Benveniste et al. 1999, 2000; Le Guernic et al. 2003; Potop-Butucaru et al.

2004] and the ForSyDe and SML-Sys modeling frameworks [Mathaikutty et al. 2004; Sander and Jantsch 2004].

The basis of our compositional theory of heterogeneous reactive systems is the notion of *Tag Systems*, a variation of the Lee and Sangiovanni-Vincentelli's (LSV) tagged-signal model [Lee and Sangiovanni-Vincentelli 1998]. The LSV model is a denotational approach where a system is modeled as a set of behaviors. Behaviors are sets of events. Each event is characterized by a data value and a tag. Complex systems are derived through the parallel composition of simpler subsystems, by taking the conjunction (intersection) of their corresponding sets of behaviors.

Our model departs from the LSV model in the following way. Most embedded electronics or control systems do not require the full generality of the original LSV model: signals are collected, exchanged between computers or network nodes, and transmitted to the actuators, either as periodically sampled sequences of data, or as sporadic sequences of messages. Our model captures these aspects by considering that behaviors are *finite sets of signals*, where each signal is a *sequence of events* labeled by a variable name. Being more structured, the nature of these behaviors allows us to develop more effective results and theorems.

For both the original LSV and our model, the role of tags varies according to the particular modeling intent. For instance, they can be used to index events belonging to the same reaction (when modeling synchronous systems) or to capture causality relations between events. In fact, tag systems can be seen as a common formalism to express different MoCCs computation and reason on their relationships. In this respect, tags play a fundamental role. When we combine components to build a system, tags are used to resolve ordering among events at the interface of the components. The mechanism of resolving tags and values of interface variables is called *unification*. By defining proper mappings between tag sets, we can formalize the process of constructing heterogeneous systems via the composition of subsystems that have different tag sets. Further, by introducing appropriate mappings between tag sets of systems with different coordination policies, we can state conditions under which the implementation of a synchronous design on distributed loosely-synchronized architectures maintains the same behavior, that is, the implementation is semantics preserving.

The main contributions of this article can be summarized as follows: In Section 2, we define the concept of *tag structure* and we use it to introduce *tag systems* as a mathematical model for both homogeneous and heterogeneous systems: homogeneous systems share the same tag structure while heterogeneous systems have different tag structures. While tags as such are sufficient to capture GALs architectures as extensively discussed in Benveniste et al. [2003], dealing with causality relations, scheduling constraints, and earliest execution times defined on each of the components requires a more sophisticated mechanism. The new concept of “stretching” is introduced for this purpose; it consists in implicitly allowing certain deformations of the tags for behaviors. When equipped with their associated stretching mechanisms, *tag structures entirely characterize MoCCs*. This is our first contribution.

In Section 3, we go beyond homogeneous systems by introducing an appropriate algebra of tag structures that allows us to define heterogeneous parallel composition, i.e., the composition of two systems having different tag structures. This makes it possible to define heterogeneous architectures as a network of heterogeneous components connected by heterogeneous communication media. The notion of *fibred product* of tag structures that we propose adequately formalizes the concept of “heterogenous MoCC.” This is our second and most important contribution.

Our results are particularly valuable when applied to the problem of deriving the correct deployment of a system specification on a distributed architecture. In Section 4, we present an application of our theoretical framework to the problem of “matching” a specification and an implementation that are heterogeneous. To our knowledge, this is the first attempt to provide a “deployment theory” ensuring correct-by-construction deployment in a general setting.

In Section 5, we use the proposed framework to examine the deployment of a synchronous specification on a loosely time-triggered architecture (LTTA) [Benveniste et al. 2002, 2003]. This architecture is an important concept developed within the aerospace industry where a precise notion of time and synchronous events cannot be guaranteed on the field because of the distributed nature of the target architecture. Our analysis shows that the deployment strategy proposed is, indeed, correct-by-construction.

Finally Section 6 is devoted to more detailed comparison with previous work. The comparison with LSV is discussed in more technical detail, as is the progress made with respect to the author’s previous results regarding GALS models. Finally, we compare our approach with the one in which details of communication semantics is expanded by using very low-level models.

## 2. TAG SYSTEMS

We begin this section by motivating our tag-based approach and stress the importance of the special mechanism of “stretching” tags. We then introduce our formalism.

### 2.1 Informal Discussion of a Few Different Models of Computation and Communication (MoCC)

**2.1.1 Synchrony.** This is the simplest MoCC for consideration. The simplest view of synchrony is that a synchronous machine progresses according to a global logical clock, or, equivalently, according to successive *reactions*. At each reaction, the machine reads its memory, reads its inputs, computes the outputs and refreshes its memories. We abstract the different operations performed by this machine as a finite number of different types of *actions*. For a more refined version of the model, not all actions are performed at each clock tick. For each type of action (we simply say “action” in the sequel), we may, therefore, label its successive occurrences by the index of the reaction where it occurs.

An important information is when two different actions must occur at identical subsequences of reactions (they are sometimes called “strictly synchronous”); or when if the first action is performed in a given reaction, then

the second one must also be performed; or when two actions are exclusive. On the other hand, silent reactions, that is, reactions where no action at all is performed, may be discarded. Conversely, one may insert silent reactions between two successive reactions, to allow for the environment to progress while the considered system is sleeping—this was called *stuttering invariance* by Lamport [2003].

Therefore, reactions may be indexed by natural numbers, but the indexing is not really uniquely defined. We are free to “stretch” our indexing. For example, one may index the reactions with 1, 2, 3, 4, . . . or equally well with 2, 4, 6, 8, . . . or 3, 5, 9, 11, . . . , by leaving room for silent reactions. These different indexing are all as good and we regard them as equivalent.

**2.1.2 Asynchrony.** The next MoCC for consideration after synchrony is asynchrony. There are several commonly accepted definitions of asynchrony. Here we take the point of view that, in an asynchronous MoCC, different actions can progress on their own and are not synchronized. Thus, each action has its own successive occurrences, and there is no need for any additional indexing information. The typical associated communication medium consists of a bundle of point-to-point channels of unbounded FIFO type.

**2.1.3 Partial Ordering or Causal Dependencies.** A convenient way to coordinate actions is by partially ordering them. Such partial orders may result from causality constraints, for example, because a given occurrence of some action uses the results of other actions as its inputs. Partial orders may abstract a token-based actor mode of execution, or they may simply be scheduling constraints. In such type of MoCC, an execution is a labeled partial order, where the labels can be actions, or variables.

Partial orders can be specified by means of a directed graph (e.g., by taking its transitive reduction). For ordering successive occurrences of a finite set of different actions, vector clocks or *multiclocks* have been proposed by Mattern [1989] as a convenient alternative. If the action alphabet has cardinal  $P$ , then the multiclock can be a  $P$ -tuple of nonnegative integers. Different multiclocks are partially ordered by the product order on the set of natural integers, e.g., for  $P = 2$ ,  $(n, n') \geq (m, m')$  iff  $n \geq m$  and  $n' \geq m'$ . Again, this coding of partial orders is not unique: dilating the components of the multiclocks, or, alternatively, compressing them, yields the same partial order.

**2.1.4 WCET.** How to assess the timing performance of an application running on a given architecture is an important task for real-time systems development. Worst-case execution times (WCET) provide an upper bound on the task execution time under all possible circumstances. Meanwhile they provide a lower bound on the initiation period of multiple consecutive execution of the same task. Stated differently, if only earliest execution times are specified, all legal executions are obtained by simply delaying the earliest ones. Thus, if real time  $\mathbf{R}_+$  is used for indexing, then dilating this index in a legal execution yields another legal execution.

**2.1.5 Summarizing and Generalizing.** For all examples we discussed, the indexing of events by appropriate “tags” was instrumental. Different MoCCs

required different types of tags. For some cases, our “tags” were indeed not uniquely defined, and we could in fact describe the “same” execution by using different actual tags. For other cases, tags were useful at defining extremal executions, and other executions would be deduced from the extremal ones by simple operations on their tags. Finally, for some cases, tags must be rigid. We formalize this approach in the next section.

## 2.2 Tag Systems and Their (Homogeneous) Parallel Composition

Throughout this article,  $\mathbf{N} = \{1, 2, \dots\}$  denotes the set of positive integers;  $\mathbf{N}$  is equipped with its usual total order  $\leq$ .  $X \mapsto Y$  denotes the set of all partial functions from  $X$  to  $Y$ . If  $(X, \leq_X)$  and  $(Y, \leq_Y)$  are partial orders,  $f \in X \mapsto Y$  is called *increasing* if  $f(\leq_X) \subseteq \leq_Y$ , i.e.,  $\forall x, x' \in X : x \leq_X x' \Rightarrow f(x) \leq_Y f(x')$ . Finally,  $f \circ g$  denotes the composition of the two functions  $f$  and  $g$ , i.e.,  $f \circ g(x) = f(g(x))$ .

*Definition 2.1 (Tag Structure).* A *tag structure* is a triple  $(\mathcal{T}, \leq, \Phi)$ , where:

- $\mathcal{T}$  is the set of *tags*, and  $(\mathcal{T}, \leq)$  is a partial order;
- $\Phi$  is a set of increasing total functions  $\varphi : \mathcal{T} \mapsto \mathcal{T}$  containing the identity “*id*,” closed under composition, and satisfying the following property: for any two  $\varphi, \psi \in \Phi$ , there exist two *complementary* functions,  $\bar{\varphi}, \bar{\psi} \in \Phi$ , such that  $\bar{\psi} \circ \psi = \bar{\varphi} \circ \varphi$ . Elements of  $\Phi$  are called *stretching functions*.

When no confusion can occur, we shall simply denote a tag structure by  $\mathcal{T}$ , for short, instead of  $(\mathcal{T}, \leq, \Phi)$ .

**LEMMA 2.2.** *For  $(\mathcal{T}, \leq)$  any partial order, taking one of the following choices for  $\Phi$  yields a tag structure  $(\mathcal{T}, \leq, \Phi)$ :*

1.  $\Phi = \{id\}$ ;
2. Assume that there exists a least upper bound  $\max(\tau, \tau')$  for each pair  $(\tau, \tau')$  of tags and take for  $\Phi$  the set of all dilating increasing functions  $\varphi$ , i.e., such that  $\varphi(\tau) \geq \tau$  for every  $\tau$ ;
3. Assume that there exists a greatest lower bound  $\min(\tau, \tau')$  for each pair  $(\tau, \tau')$  of tags and take for  $\Phi$  the set of all contracting increasing functions  $\varphi$ , i.e., such that  $\varphi(\tau) \leq \tau$  for every  $\tau$ .

**PROOF.** See Appendix A.1. □

*Example 2.3 (Tag Structures).*

1. Take  $\mathcal{T} = \{.\}$ , the singleton set, with  $\leq$  the trivial order, and  $\Phi = \{id\}$ . As we shall see, this trivial tag structure will be used in modeling asynchrony; we denote it by  $\mathcal{T}_{\text{triv}}$ .
2. Take  $\mathcal{T} = \mathbf{N}$ , with  $\leq$  being the usual (total) order. This tag structure will be used in modeling synchrony (tags are reaction indices); we denote it by  $\mathcal{T}_{\text{synch}}$ . Regarding stretching functions, we can either take  $\Phi = \{id\}$ , or  $\Phi$  equals the set of all dilating increasing functions, if we want to consider stuttering invariant systems.

3. Take  $\mathcal{T} = \mathbf{R}_+$ , with  $\leq$  being the usual (total) order, and  $\Phi = \{id\}$ . This tag structure will be used in modeling time-triggered systems (tags are dates from realtime); we denote it by  $\mathcal{T}_{tta}$ .
4. Take  $\mathcal{T} = \mathbf{R}_+$ , with  $\leq$  the usual (total) order, and  $\Phi$  equals the set of all dilating increasing functions  $\varphi$ . This tag structure will be used in capturing execution times, where earliest possible dates for execution of events constitute the relevant specification and all later dates are permitted. We denote it by  $\mathcal{T}_{wct}$ .
5. Take  $\mathcal{T} = \mathbf{R}_+ \cup \{+\infty\}$ , with  $\leq$  the usual (total) order, and  $\Phi$  equals the set of all contracting increasing functions  $\varphi$ . This tag structure will be used in capturing deadlines, where latest possible dates for execution of events constitute the relevant specification and all earlier dates are permitted. We denote it by  $\mathcal{T}_{dead}$ .
6. Let  $\mathcal{V}$  be some underlying set of variables and set  $\mathbf{N}_o =_{\text{def}} \mathbf{N} \cup \{0\}$ . Define a *multiclock* to be a map:  $\kappa : \mathcal{V} \mapsto \mathbf{N}_o$ . Denote by  $\mathcal{T}_{mclk}$  the set of all multiclocks and equip it with the partial order  $\leq$  such that  $\kappa \leq \kappa'$  iff  $\forall v \in \mathcal{V}$ :  $\kappa(v) \leq \kappa'(v)$ . Then, take for  $\Phi$  the set of all dilating increasing total functions  $\mathcal{T}_{mclk} \mapsto \mathcal{T}_{mclk}$ , i.e., such that  $\forall v \in \mathcal{V}$ ,  $\varphi(\kappa(v)) \geq \kappa(v)$ . (In fact, we use all multiclocks that are related via stretching as “coded representations” of the same partial order of events.) This tag structure will be used in modeling causal dependencies or scheduling relations in the form of partial orders. We denote this tag structure by  $\mathcal{T}_{mclk}$ .

As pointed out, we restrict ourselves to a framework where each individual variable takes a totally ordered sequence of values, i.e., where each signal of the system is a total order in our abstractions. Let  $\mathcal{V}$  be an underlying set of variables with domain  $D$ . For  $V \subset \mathcal{V}$  finite, a *V-behavior*, or simply behavior, is an element:

$$\sigma \in V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D), \quad (1)$$

meaning that, for each  $v \in V$ , the  $n$ th occurrence of  $v$  in behavior  $\sigma$  has tag  $\tau \in \mathcal{T}$  and value  $x \in D$ . For  $v$  a variable, the map  $\sigma(v) \in \mathbf{N} \mapsto (\mathcal{T} \times D)$  is called a *signal*. For  $\sigma$  a behavior, an *event* of  $\sigma$  is a tuple  $e = (v, n, \tau, x) \in V \times \mathbf{N} \times \mathcal{T} \times D$  such that  $\sigma(v)(n) = (\tau, x)$ . Thus we can regard behaviors as sets of events.

Call a *clock* any increasing function  $(\mathbf{N}, \leq) \mapsto (\mathcal{T}, \leq)$ . We require that, for each  $v \in V$ , the first projection of the map  $\sigma(v)$  (it is a map  $\mathbf{N} \mapsto \mathcal{T}$ ) is a clock and we call it the *clock of  $v$  in  $\sigma$* . Thus, the clock of  $v$  yields the tags of the successive events of signal  $\sigma(v)$ .

For  $\sigma$  and  $\sigma'$  two  $V$  behaviors, say that  $\sigma'$  is a *stretching of  $\sigma$*  iff there exists a stretching function  $\varphi \in \Phi$  such that

$$\sigma' = \{e' = (v, n, \varphi(\tau), x) \mid e = (v, n, \tau, x) \in \sigma\}, \text{ written } \sigma' = \varphi.\sigma \quad (2)$$

Note that  $\varphi'.(\varphi.\sigma) = (\varphi' \circ \varphi).\sigma$ . A set of behaviors  $\Sigma$  is called *stretching-invariant* iff every stretching of every behavior of  $\Sigma$  belongs to  $\Sigma$ .

*Definition 2.4 (Tag Systems).* A tag system is a triple  $P = (V, \mathcal{T}, \Sigma)$ , where  $V$  is a finite set of variables,  $\mathcal{T}$  is a tag structure, and  $\Sigma$  a stretching-invariant set of  $V$ -behaviors.

Throughout this section, we consider only tag systems having identical tag structure; this is why we use the term “homogeneous” for our parallel composition. Consider two tag systems  $P_1 = (V_1, \mathcal{T}, \Sigma_1)$  and  $P_2 = (V_2, \mathcal{T}, \Sigma_2)$  with identical tag structure  $\mathcal{T}$ . For  $\sigma$  a  $V$ -behavior and  $\sigma'$  a  $V'$ -behavior, define, by abuse of notation:

$$\sigma \bowtie \sigma' \text{ iff } \sigma|_{V \cap V'} = \sigma'|_{V \cap V'},$$

where  $\sigma|_W$  denotes the restriction of behavior  $\sigma$  to the variables of  $W$ , and, if  $\sigma \bowtie \sigma'$  holds,

$$\sigma \sqcup \sigma' =_{\text{def}} (\sigma|_{V \cap V'}) \cup \sigma|_{V \setminus V'} \cup \sigma'|_{V' \setminus V},$$

Finally, for  $\Sigma$  and  $\Sigma'$  two sets of behaviors, define their *conjunction*

$$\Sigma \wedge \Sigma' =_{\text{def}} \{\sigma \sqcup \sigma' \mid \sigma \in \Sigma, \sigma' \in \Sigma' \text{ and } \sigma \bowtie \sigma'\}. \quad (3)$$

The *homogeneous parallel composition* of  $P_1$  and  $P_2$  is then defined as follows:

$$P_1 \parallel P_2 =_{\text{def}} (V_1 \cup V_2, \mathcal{T}, \Sigma_1 \wedge \Sigma_2) \quad (4)$$

Thus parallel composition is by intersection. The homogeneous parallel composition is associative and commutative.

*Example 2.5 (Tag Systems and Homogeneous Parallel Composition).*

Here, we show how to model, in our framework, some of the most useful models of computation.

1. We first consider tag systems with  $\mathcal{T}_{\text{triv}}$  as tag structure, see Example 2.3(1).  $\mathcal{T}_{\text{triv}}$  corresponds to a situation where there is no relation specified for events belonging to different signals of the system (remember that events of the same signal are always totally ordered). If we consider the representation of a system with this type of tag as a specification, then the formalism indicates that we are completely free to decide how to “synchronize” events belonging to different signals when we move towards implementation. In this sense, we claim that these systems are *asynchronous*.<sup>1</sup> Their behaviors have the form  $\sigma : V \mapsto \mathbf{N} \mapsto \{.\} \times D$ .

Let  $P = (V_P, \mathcal{T}_{\text{triv}}, \Sigma_P)$  and  $Q = (V_Q, \mathcal{T}_{\text{triv}}, \Sigma_Q)$  be two asynchronous tag systems, each having a single behavior with  $V_P = \{b, x, y\}$  and  $V_Q = \{b, x, z\}$ . Variable  $b$  is of Boolean type with values in  $\{\mathbf{F}, \mathbf{T}\}$ ; variables  $x, y$ , and  $z$  are of integer type. Notice that variables  $b$  and  $x$  are *shared* across the two systems. The system behaviors are as follows (where we omit the tag part since  $\mathcal{T}_{\text{triv}}$  is the trivial tag structure).

<sup>1</sup>The term asynchronous has been used in many different ways. We may think of asynchronous any system that is not synchronous. In Lee and Sangiovanni-Vincentelli [1998], asynchronous relates to systems where no two events have the same tag and, hence, it is a more restrictive definition than the one we use here.

$P :$	$b$	:	T	F	T	F	T	F	...
	$x$	:	1	1	1	.	.	.	
	$y$	:	1	3	5	.	.	.	
$Q :$	$b$	:	T	F	T	F	T	F	...
	$x$	:	1	1	1	.	.	.	
	$z$	:	2	4	6	.	.	.	

Using our notations, these behaviors are formally given by:

$$P : \begin{cases} \sigma(b)(2n-1) = \text{T}, \sigma(b)(2n) = \text{F} \\ \sigma(x)(n) = 1, \\ \sigma(y)(n) = 1 + 2(n-1) \end{cases}$$

$$Q : \begin{cases} \sigma(b)(2n-1) = \text{T}, \sigma(b)(2n) = \text{F} \\ \sigma(x)(n) = 1, \\ \sigma(z)(n) = 2 + 2(n-1) \end{cases}$$

Since parallel composition is by intersection, we have  $(P \parallel Q)_{|V_P} = P$ , and  $(P \parallel Q)_{|V_Q} = Q$ .

*Nota:* In the sequel, to simplify the discussion of this example, we shall discard the private variables  $y$  and  $z$  from  $P$  and  $Q$ , respectively. With this simplification, we get  $P = Q = P \parallel Q$ .

2. Consider the tag structure  $\mathcal{T}_{\text{synch}}$  introduced in Example 2.3(2). We claim that  $\mathcal{T}_{\text{synch}}$  defines synchronous systems. Indeed, provided that all clocks are *strictly increasing*, the tag index set  $\mathcal{T}_{\text{synch}}$  organizes behaviors into successive reactions, where a *reaction* is a maximal set of events of  $\sigma$  with identical tag. Since clocks are strictly increasing, no two events of the same reaction can have the same variable. Regard a behavior as a sequence of reactions:  $\sigma = \sigma_1, \sigma_2, \dots$ , with tags  $\tau_1, \tau_2, \dots \in \mathcal{T}_{\text{synch}}$ . Thus  $\mathcal{T}_{\text{synch}}$  provides a global, logical time basis: this feature characterizes synchrony. For this example, we equip  $\mathcal{T}_{\text{synch}}$  with  $\Phi = \{id\}$ . Let  $P_s = (V_{P_s}, \mathcal{T}_{\text{synch}}, \Sigma_{P_s})$  and  $Q_s = (V_{Q_s}, \mathcal{T}_{\text{synch}}, \Sigma_{Q_s})$  be two synchronous tag systems, each having a single behavior with  $V_{P_s} = \{b, x, y\}$  and  $V_{Q_s} = \{b, x, z\}$ . Variable  $b$  is of Boolean type with values in  $\{\text{F}, \text{T}\}$ ; variables  $x, y, z$  are of integer type. The behaviors are:

$P_s :$	$b$	:	T	F	T	F	T	F	...
	$x$	:	1		1		1		...
	$y$	:	1		3		5		...
$Q_s :$	$b$	:	T	F	T	F	T	F	...
	$x$	:		1		1		1	...
	$z$	:		2		4		6	...

In the above description, each behavior consists of a sequence of successive reactions, separated by vertical bars. Each reaction consists of an assignment of values to a subset of the variables; a blank indicates the absence of the considered variable in the considered reaction. These behaviors can be

expressed formally in our framework as follows:

$$P_s : \begin{cases} \sigma(b)(2n-1) = (2n-1, T), & \sigma(b)(2n) = (2n, F) \\ \sigma(x)(n) = (2n-1, 1) \\ \sigma(y)(n) = (2n-1, 1+2(n-1)) \end{cases}$$

$$Q_s : \begin{cases} \sigma(b)(2n-1) = (2n-1, T), & \sigma(b)(2n) = (2n, F) \\ \sigma(x)(n) = (2n, 1) \\ \sigma(z)(n) = (2n, 2+2(n-1)) \end{cases}$$

Now, the synchronous parallel composition of  $P_s$  and  $Q_s$ , defined by intersection:  $P_s \parallel Q_s =_{\text{def}} P_s \cap Q_s$ , is empty. The reason is that  $P_s$  and  $Q_s$  disagree on where to put absences for the shared variable  $x$ . Formally, they disagree on their respective tags.

*Nota:* In the sequel, to simplify the discussion of this example, we shall discard the private variables  $y$  and  $z$  from  $P_s$  and  $Q_s$ , respectively.

3. Consider the tag structure  $\mathcal{T}_{\text{mclk}}$  associated with dependencies in Example 2.3(6). Consider two tag systems  $P_\kappa$  and  $Q_\kappa$  having two variables,  $b$  and  $x$ , as before. Assume that each system possesses only a single behavior, equal to:

$$P_\kappa : \begin{array}{c} \begin{array}{cccccccc} b & : & T & F & T & F & T & F & \dots \\ & & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & \nearrow & \dots \\ x & : & 1 & & 1 & & 1 & & \dots \end{array} \\ \hline \begin{array}{cccccccc} b & : & T & F & T & F & T & F & \dots \\ & & & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & \dots \\ x & : & & 1 & & 1 & & 1 & \dots \end{array} \end{array}$$

The directed graphs shown are just the transitive reductions of the two partial orders. A possible multilock coding of these two partial orders is, for example:

$$P_\kappa : \begin{array}{c} \begin{array}{cccccccc} b & : & (0,0);T & (1,1);F & (2,1);T & (3,2);F & (4,2);T & (5,3);F & \dots \\ x & : & (1,0);1 & & (3,1);1 & & (5,2);1 & & \dots \end{array} \\ \hline \begin{array}{cccccccc} b & : & (0,0);T & (1,0);F & (2,1);T & (3,1);F & (4,2);T & (5,2);F & \dots \\ x & : & & (2,0);1 & & (4,1);1 & & (6,2);1 & \dots \end{array} \end{array}$$

where, for example, the pair  $(2,1)$  in  $(2,1); T$  means that, the multilock of this event is  $\kappa(b) = 2$  and  $\kappa(x) = 1$ . Other multilocks can encode the same partial order. For example, applying the dilating function  $\varphi$  defined by  $\forall(n, p) : \varphi(n, p) = (2n, p+1)$ , we would get the equivalent coding (we provide it only for  $P_\kappa$ ):

$$P_\kappa : \begin{array}{c} \begin{array}{cccccccc} b & : & (0,1);T & (2,2);F & (4,2);T & (6,3);F & (8,3);T & (10,4);F & \dots \\ x & : & (2,1);1 & & (6,2);1 & & (10,3);1 & & \dots \end{array} \end{array}$$

As the reader can check, there is no pair  $(\varphi_{P_\kappa}, \varphi_{Q_\kappa})$  of dilating functions that can make the tags of the two behaviors of  $P_\kappa$  and  $Q_\kappa$  identical. Since

parallel composition is by intersection, this means that the parallel composition  $P_\kappa \parallel Q_\kappa$  is empty, as was expected from confronting the above graphical definitions for these two systems.

4. Consider  $\mathcal{T}_{\text{wcet}}$ , the tag structure for performance evaluation introduced in Example 2.3(4), and consider the following behavior:

$$P_w : \begin{array}{cccccccc} \hline b : & 0;\text{T} & 1;\text{F} & 2;\text{T} & 3;\text{F} & 4;\text{T} & 5;\text{F} & \dots \\ x : & 0;1 & & 2;1 & & 4;1 & & \dots \\ \hline \end{array}$$

Since we are dealing with  $\mathcal{T}_{\text{wcet}}$ , the family  $\Phi_{\text{wcet}}$  of stretching functions consists of all dilating functions, and  $P_w$  must be closed by all corresponding stretching. Accordingly, we interpret this behavior as specifying the *earliest* possible dates of occurrences for the listed events.

Now, if we were instead dealing with  $\mathcal{T}_{\text{dead}}$ , then the same behavior should be interpreted as specifying the *deadlines* for the listed events.

The role stretching is enlighten by the previous examples. In Examples 2 (stuttering invariant synchronous systems) and 3 (multiclock systems), stretching is intended to capture sets of behaviors considered equivalent. In Example 4 (performance evaluation), stretching really defines a set of legal behaviors from a given subset of extremal ones (here, the earliest behaviors). For other examples, we have  $\Phi = \{id\}$ , meaning that tags are “rigid.”

### 2.3 Modeling with Tags

The concept of tags allows expressing various models of computation as illustrated by the following examples.

- Asynchronous systems.* were introduced in Examples 2.3(1) and 2.5(1). They are systems in which different signals are not synchronized at all. Therefore, no nontrivial tag is needed to coordinate them. This is why  $\mathcal{T}_{\text{triv}}$  was considered to capture asynchrony.
- Synchronous systems.* were introduced in Examples 2.3(2) and 2.5(2). We discuss here their two main variants and give some pointers.

The *activation clock* of behavior  $\sigma$  is the clock  $h : \mathbf{N} \mapsto \mathcal{T}_{\text{synch}}$  such that set  $h(\mathbf{N})$  collects the tags of all events belonging to  $\sigma$ . If instant  $n \notin h(\mathbf{N})$ , then we say that  $\sigma$  is *silent* at instant  $n$ . In most models of synchronous languages (e.g., Lustre [Halbwachs et al. 1991], Esterel [Berry 2000]), programs are such that all their behaviors possess  $h = id$  as activation clock: programs are never silent. This is a good model for *closed* systems. We cover it by taking as a tag structure  $\mathcal{T}_{\text{synch}}$  with  $\Phi = \{id\}$ .

Modeling open systems requires a different approach. In open systems, for any given system, other systems can exist that are working while the one being considered is “sleeping.” Thus we need to allow stretching activation clocks, which is known as *stuttering invariance* [Lampert 2003]; we can achieved it simply by equipping tag structure  $\mathcal{T}_{\text{synch}}$  with  $\Phi$  the set of all dilating functions. The *Signal* synchronous language [Le Guernic et al. 1991] models stuttering invariant systems.

Synchronous models are pervasive in the area of embedded systems. Automata and their synchronous product are the basic example. Synchronous languages [Benveniste et al. 2003] are another one. Lamport's TLA [2003] obeys the synchronous model in its stuttering invariant form. Reactive modules [Alur and Henzinger 1999] are also a model of synchronous systems: modules progress according to a sequence of "rounds"; rounds can implement *micro-* or *macrostep* transitions; variables are divided into state and communication variables; state variables are sustained, whereas communication variables are clocked; inter-module communication is performed by inserting a delay in each wire.

Synchronous models of computation are widely used in hardware designs. These models are more restrictive than the synchronous models used in software design. In synchronous hardware, all signals are triggered by a *global, periodic clock*. Thus, synchronous hardware does obey our synchronous tag systems model, but it is different from the synchronous language paradigm, since, in hardware, we impose a time-based, periodic clock to identify the synchronization barriers. These systems can be modeled by tag systems in which tags are pairs consisting of a date and a reaction index and belonging to  $\mathcal{T}_{\text{tta}} \times \mathcal{T}_{\text{synch}}$ .

Unfortunately, integrated circuit technology advances while making it possible to clock the circuits increasingly faster, introduce variabilities that force a more conservative design style if the synchronous paradigm is to be maintained. Skew as a result of the time needed to propagate the clock signal across the chip and delay uncertainties because of process variations are making the job of hardware designers more complicated in order to guarantee that the synchronous paradigm still holds. It is possible however, to use architectures and circuit structures that make it relatively easy to guarantee that the synchronous paradigm is satisfied at the upper level of abstraction. One such example is latency-insensitive design [Carloni et al. 2001] where assuming that each of the components of the design is a stallable process (this process is stuttering invariant according to the definition above), a circuit architecture and a related protocol can be designed to mask long signal delays in the communication among processes.

- Timed systems.* are captured by means of their timed traces. Timed traces are composed of events tagged with their physical date of occurrence, captured by the  $\mathcal{T}_{\text{tta}}$  tag structure.
- Time-triggered systems.* were discussed in Example 2.3(3). Time-triggering is a concept popularized by Hermann Kopetz [1997]. It formalizes the natural observation that, in real-time systems, architecture components share physical real-time: events are characterized by a "date" attribute and physical time triggers the entire system. Time-triggered systems are typically based on a periodic clock and hence, can be modeled with the same mechanism we used for synchronous hardware.
- Causalities and scheduling specifications.* These were discussed in Examples 2.3(6) and 2.5(3). This model is adequate for the trace "true concurrency" semantics of safe Petri nets. In fact, this model is just the "trace" part of event structures [Winskel 1983], that is, is the model suited to describe configurations in event structures.

- Earliest execution times.* These were discussed in Example 2.3(4). Composing two systems has the effect that the two components wait for the latest date of occurrence for each shared variable. For example, assume that variable  $v$  is an output of  $P$  and an input of  $Q$  in  $P \parallel Q$ . Then, the earliest possible date of every event of variable  $v$  in  $Q$  is by convention 0, whereas each event associated to  $v$  has a certain date of production in  $P$ . In the parallel composition  $P \parallel Q$ , the dates of production by  $P$  prevail.
- Composite tags.* As discussed in the next sections, different tags can be combined by using the product of tag structures

$$(\mathcal{T}, \leq, \Phi) \times (\mathcal{T}', \leq', \Phi') =_{\text{def}} (\mathcal{T} \times \mathcal{T}', \leq \times \leq', \Phi \times \Phi') \quad (5)$$

This combination yields a compound, heterogeneous tag. For instance, our model of TTA or synchronous hardware uses the composite tag structure  $\mathcal{T}_{\text{tta}} \times \mathcal{T}_{\text{synchron}}$ . One can also consider synchronous systems that are timed and enhanced with causalities. Such systems can be “desynchronized,” meaning that their reaction tag is erased, but their causality and time tags are kept.

### 3. HETEROGENEOUS SYSTEMS

Heterogeneous systems are systems made of components having different models of communication and computation (MoCC). In this section, we formalize this concept. Since MoCCs are captured by tag structures, we first need to define how to combine tags from different tag structures. Then, heterogeneous parallel composition will be defined.

#### 3.1 The Algebra of Tag Structures

In the sequel, tag structures will be denoted either explicitly as a triple  $(\mathcal{T}, \leq, \Phi)$ , or simply by the symbol  $\mathcal{T}$ , if no confusion can occur.

*Definition 3.1 (morphisms).* Given two tag structures  $(\mathcal{T}, \leq, \Phi)$  and  $(\mathcal{T}', \leq', \Phi')$ , call *morphism* an increasing total map  $\rho : (\mathcal{T}, \leq) \mapsto (\mathcal{T}', \leq')$ , such that

$$\forall \varphi' \in \Phi', \exists \varphi \in \Phi \text{ such that } \rho \circ \varphi = \varphi' \circ \rho \quad (6)$$

$$\forall \varphi \in \Phi, \exists \varphi' \in \Phi' \text{ such that } \varphi' \circ \rho = \rho \circ \varphi \quad (7)$$

Morphisms compose. For  $\sigma$  a behavior defined over tag structure  $\mathcal{T}$ , let  $\rho.\sigma$  be the behavior over  $\mathcal{T}'$  obtained by replacing, in every event  $e = (v, n, \tau, x) \in \sigma$ , tag  $\tau$  by its image  $\rho(\tau)$ . For  $\Sigma$  a set of behaviors over  $\mathcal{T}$ , set  $\rho.\Sigma =_{\text{def}} \{\rho.\sigma \mid \sigma \in \Sigma\}$ . Morphisms satisfy the following key property:

**LEMMA 3.2.** *For  $P = (V, \mathcal{T}, \Sigma)$  a tag system and  $\rho : \mathcal{T} \mapsto \mathcal{T}'$  a tag morphism,  $P' = (V, \mathcal{T}', \rho.\Sigma)$  is a tag system associated with  $\mathcal{T}'$ .*

**PROOF.** See Appendix A.2; the proof uses property (6). □

The canonical projection

$$(\mathcal{T}, \leq, \Phi) \times (\mathcal{T}', \leq', \Phi') \mapsto (\mathcal{T}, \leq, \Phi)$$

is a morphism; we shall use projections to represent desynchronization, that is, the removal of some tag components in composite tags. As usual,  $\mathcal{T}$  and  $\mathcal{T}'$  are called *isomorphic* when there exist two morphisms  $\rho : \mathcal{T} \mapsto \mathcal{T}'$  and  $\rho' : \mathcal{T}' \mapsto \mathcal{T}$ , such that  $\rho' \circ \rho = id_{\mathcal{T}}$  and  $\rho \circ \rho' = id_{\mathcal{T}'}$ . We do not distinguish isomorphic tag structures. Morphisms induce a preorder on tag structures:

$$\mathcal{T}' \ll \mathcal{T} \quad \text{iff there exists a morphism } \rho : \mathcal{T} \mapsto \mathcal{T}'. \quad (8)$$

Note that there is no lower bound associated to this preorder, as we further discuss in Example 3.5(4) that follows. For convenience, we shall sometimes write  $\mathcal{T} \xrightarrow{\rho} \mathcal{T}'$  to mean  $\rho : \mathcal{T} \mapsto \mathcal{T}'$ .

*Example 3.3 (Morphisms and projections).*

1. Take  $\mathcal{T} = \mathcal{T}_{\text{synch}}$ ,  $\mathcal{T}' = \mathcal{T}_{\text{triv}}$ , and  $\rho$  is the morphism that maps all  $\tau \in \mathcal{T}$  to the unique tag constituting  $\mathcal{T}'$ ; morphism  $\rho$  models desynchronization in the usual sense. An illustration of this is given in Examples 2.5(1) and 2.5(2)  $P$  and  $Q$  are obtained by desynchronizing  $P_s$  and  $Q_s$ , respectively. This example illustrates that desynchronization is not injective.
2. Take  $\mathcal{T} = \mathbf{R}_+$ ,  $\mathcal{T}' = \mathbf{N}$  with  $\leq$  the usual order and  $\Phi = \{id\}$ . The map  $\rho : \mathbf{R}_+ \ni t \mapsto \lfloor t \rfloor \in \mathbf{N}$ , where  $\lfloor t \rfloor$  denotes the largest integer smaller than  $t$ , is a morphism. This morphism models the mechanism of periodically polling an unbounded buffer to form successive reactions: data are collected from the buffer between dates  $t = n - 1$  and  $t' = n$  to form the  $n$ th reaction. Thus,  $\rho$  models the input interface between a synchronous system and its asynchronous real-time environment.
3. One may consider enhancing synchronous systems with causalities. This is achieved by taking the product  $\mathcal{T}_{\text{synch}} \times \mathcal{T}_{\text{mclk}}$  (cf. (5)) as tag structure. Referring to the Examples 2.5(2) and 2.5(3), the informal description of the first system is:

$$P_{sk} : \begin{array}{c|c|c|c|c|c|c|c} \hline b & : & T & F & T & F & T & F & \dots \\ \hline & & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & \nearrow & \dots \\ \hline x & : & 1 & & 1 & & 1 & & \dots \\ \hline \end{array}$$

and its formal counterpart by using multiclocks is:

$$\begin{aligned} \sigma(b)(2n+1) &= ((2n+1, [2n, n]), T) \\ \sigma(b)(2n) &= ((2n, [2n-1, n]), F) \\ \sigma(x)(n) &= ((2n-1, [2n-1, n-1]), 1) \end{aligned}$$

where the composite tag  $(2n+1, [2n, n])$  combines the reaction index  $2n+1$  with the multiclock  $[2n, n]$ . Projecting away multiclocks from composite tags in  $P_{sk}$  yields  $P_s$ , whereas projecting away synchronization barriers yields  $P_{\kappa}$ .

An essential step in defining heterogeneous parallel composition consists in defining how to unify tags from different tag structures. The idea is that we can do this provided that these two different tag structures share something in common regarding synchronization. This heterogeneous unification mechanism is captured by the concept of fibered product, which we introduce next.

*Fibered product.* For  $\mathcal{T}_1 \xrightarrow{\rho_1} \mathcal{T} \xleftarrow{\rho_2} \mathcal{T}_2$  two morphisms, define their *fibered product*:

$$\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2 =_{\text{def}} \{(\tau_1, \tau_2) \in \mathcal{T}_1 \times \mathcal{T}_2 \mid \rho_1(\tau_1) = \rho_2(\tau_2)\}, \quad (9)$$

also written  $\mathcal{T}_1 \times_{\mathcal{T}} \mathcal{T}_2$ . The fibered product is equipped with the restriction of the product order  $\leq_1 \times \leq_2$ . Write

$$\tau_1 \times_{\rho_1 \times \rho_2} \tau_2 \text{ to mean that } (\tau_1, \tau_2) \in \mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2, \quad (10)$$

(we say that the pair  $(\tau_1, \tau_2)$  is  $(\rho_1, \rho_2)$ -*unifiable*) and denote by

$$\tau_1 \sqcup_{\rho_1 \times \rho_2} \tau_2$$

the pair  $(\tau_1, \tau_2)$  seen as an element of  $\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2$ . Then, the corresponding set of stretching functions, we denote it by  $\Phi_1 \times_{\rho_1 \times \rho_2} \Phi_2$ , consists of all pairs  $(\varphi_1, \varphi_2)$  such that  $\varphi_i \in \Phi_i$  for  $i \in \{1, 2\}$  and

$$\tau_1 \times_{\rho_1 \times \rho_2} \tau_2 \Rightarrow \varphi_1(\tau_1) \times_{\rho_1 \times \rho_2} \varphi_2(\tau_2) \quad (11)$$

The map  $\rho$  defined by:

$$(\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2) \ni (\tau_1 \sqcup_{\rho_1 \times \rho_2} \tau_2) \longmapsto \rho_1(\tau_1) = \rho_2(\tau_2) \in \mathcal{T} \quad (12)$$

defines the canonical morphism associated with this fibered product.

**LEMMA 3.4.** *The above definition for  $\Phi_1 \times_{\rho_1 \times \rho_2} \Phi_2$  satisfies the requirements for a set of stretching functions.*

**PROOF.** See Appendix A.3; the proof uses properties (6) and (7).  $\square$

*Example 3.5 (Fibered product).*

1. Note first that  $\mathcal{T} \times_{id \times id} \mathcal{T} = \mathcal{T}$ .
2. The property  $\mathcal{T}_{\text{triv}} \ll \mathcal{T}$ , cf. (8), holds for any tag structure  $\mathcal{T}$ . Therefore, any pair  $(\mathcal{T}, \mathcal{T}')$  of tag structures can be synchronized by means of the trivial fibered product  $\mathcal{T} \times_{\mathcal{T}_{\text{triv}}} \mathcal{T}'$ . This is the weakest synchronization that can be considered between  $\mathcal{T}$  and  $\mathcal{T}'$ . This always works but may be of moderate interest in general.
3. A more interesting case is when  $\mathcal{T}_i = \mathcal{T}'_i \times \mathcal{T}$ ,  $i = 1, 2$ , and the two morphisms are the projections  $\pi_i : \mathcal{T}_i \mapsto \mathcal{T}$ . Then,  $\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2$  reduces to the product  $\mathcal{T}'_1 \times \mathcal{T} \times \mathcal{T}'_2$ . This is the typical situation when composing systems with different tag structures: the two tag structures are composite and share one of their components. (See for example our LTTA example and Figure 6).
4. Figure 2 illustrates the interaction of a synchronous system with input and output Time-Triggered environments. The input interface consists of the polling mechanism  $\rho$  described in Example 3.3(2).  $P_s$  is the synchronous system. The output interface consists in performing the following at each reaction: (1) read the output event (it consists of a finite set of data), and (2) serialize the data and emit them through the output wire before the next reaction starts. The output interface is therefore nondeterministic since the precise dates for emission are not determined. In fact, the output interface implements a (nondeterministic) inverse of morphism  $\rho$ . This is illustrated

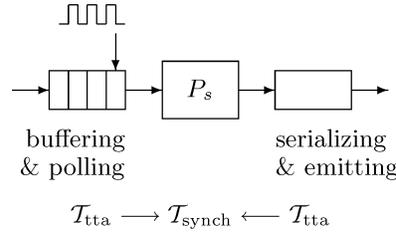


Fig. 2. Modeling the interaction of a synchronous system with input and output time-triggered environments.

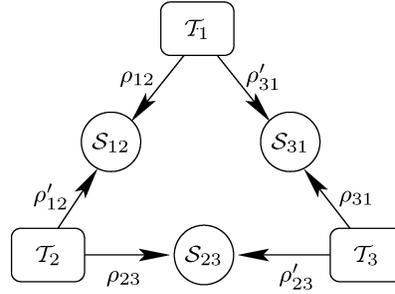


Fig. 3. Multiple fibered product.

by the symmetric double arrow  $\mathcal{T}_{\text{tta}} \rightarrow \mathcal{T}_{\text{synch}} \leftarrow \mathcal{T}_{\text{tta}}$ . In fact, the combination of  $P_s$  and its interfaces possesses the fibered product  $\mathcal{T}_{\text{tta}} \rho \times_{\rho} \mathcal{T}_{\text{tta}}$  as natural tag structure.

Formula (9) defines the fibered product for two tag structures. Because of its heterogeneous nature, this fibered product cannot be associative. Hence, how can we generalize it for more than two tag structures? Consider the particular case of the “triangle,” shown in Figure 3. The fibered product specified by Figure 3 is:

$$\{(\tau_1, \tau_2, \tau_3) \in \mathcal{T}_1 \times \mathcal{T}_2 \times \mathcal{T}_3 \mid \rho_{ij}(\tau_i) = \rho'_{ij}(\tau_j)\}, \quad (13)$$

where the pair  $(i, j)$  ranges over the set  $\{(1, 2), (2, 3), (3, 1)\}$ . In words, *the fibered product is the set of all tuples of tags satisfying all specified equality constraints*. This is formalized in the following definition:

**Definition 3.6 (*n*-ary fibered product).** Let  $(\mathcal{T}_i, \leq_i, \Phi_i)_{i \in I}$  be a finite set of tag structures, and let  $\mathbf{R}$  be a set of pairs of morphisms  $\mathcal{T}_i \xrightarrow{\rho_{ij}} \mathcal{S}_{ij} \xleftarrow{\rho'_{ij}} \mathcal{T}_j$ , where  $(i, j)$  ranges over  $I \times I$ . The *n*-ary fibered product of the  $\mathcal{T}_i$  via  $\mathbf{R}$  is:

$$\prod_{\mathbf{R}, i \in I} \mathcal{T}_i =_{\text{def}} \left\{ (\tau_i)_{i \in I} \in \prod_{i \in I} \mathcal{T}_i \mid \forall (\rho_{ij}, \rho'_{ij}) \in \mathbf{R} \Rightarrow \rho_{ij}(\tau_i) = \rho'_{ij}(\tau_j) \right\}, \quad (14)$$

equipped with the restrictions of the product order and the subset of  $\prod_{i \in I} \Phi_i$  consisting of the tuples  $(\varphi)_{i \in I}$  that leave  $\prod_{\mathbf{R}, i \in I} \mathcal{T}_i$  invariant. The corresponding tag unification map is denoted by

$$\sqcup_{\mathbf{R}, i \in I} \tau_i. \quad (15)$$

### 3.2 Heterogeneous Parallel Composition

In this section, we define the parallel composition of tag systems having different tag structures. Again, we first provide the detailed definition for two components, and then we generalize.

Given a morphism  $\rho : \mathcal{T} \mapsto \mathcal{S}$  and a behavior  $\sigma \in V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D)$ , replacing  $\tau$  by  $\rho(\tau)$  in  $\sigma$  defines a new behavior having  $\mathcal{S}$  as tag structure. This behavior is denoted as  $\sigma_\rho$ , or (with some abuse of notation) as  $\sigma \circ \rho$ . Performing this for every behavior of a tag system  $P$  with tag structure  $\mathcal{T}$  yields the tag system

$$P_\rho, \text{ also denoted by } P_S. \quad (16)$$

Assume two morphisms  $\mathcal{T}_1 \xrightarrow{\rho_1} \mathcal{T} \xleftarrow{\rho_2} \mathcal{T}_2$ . Write:  $\sigma_1 \rho_1 \bowtie_{\rho_2} \sigma_2$  iff  $\sigma_1 \circ \rho_1 \bowtie \sigma_2 \circ \rho_2$ . For  $(\sigma_1, \sigma_2)$  a pair satisfying  $\sigma_1 \rho_1 \bowtie_{\rho_2} \sigma_2$ , define

$$\sigma_1 \rho_1 \sqcup_{\rho_2} \sigma_2 \quad (17)$$

as being the set of events  $(v, n, (\tau_1, \tau_2), x)$  such that  $\rho_1(\tau_1) = \rho_2(\tau_2) =_{\text{def}} \tau$  and:

1. if  $v \in V_i$  for  $i \in \{1, 2\}$ , then  $(v, n, \tau_i, x)$  is an event of  $\sigma_i$ , and
2.  $(v, n, \tau, x)$  is an event of  $(\sigma_1 \circ \rho_1) \sqcup (\sigma_2 \circ \rho_2)$ .

We are now ready to define the *heterogeneous conjunction* of  $\Sigma_1$  and  $\Sigma_2$  by:

$$\Sigma_1 \rho_1 \wedge_{\rho_2} \Sigma_2 =_{\text{def}} \{ \sigma_1 \rho_1 \sqcup_{\rho_2} \sigma_2 \mid \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2, \sigma_1 \rho_1 \bowtie_{\rho_2} \sigma_2 \} \quad (18)$$

Finally, the *heterogeneous parallel composition* of  $P_1$  and  $P_2$  is defined by:

$$P_1 \rho_1 \parallel_{\rho_2} P_2 = (V_1 \cup V_2, \mathcal{T}_1 \times_{\rho_2} \mathcal{T}_2, \Sigma_1 \rho_1 \wedge_{\rho_2} \Sigma_2). \quad (19)$$

So far we only defined heterogeneous parallel composition of two components. Since this parallel composition cannot be associative, we need in fact to define it directly for more than two components.

*Definition 3.7 (Heterogeneous parallel composition).* Let  $P_i = (V_i, \mathcal{T}_i, \Sigma_i)$  be a finite set of tag systems, where  $i \in I$ , and let  $\mathbf{R}$  be a set of pairs of morphisms

$$\mathcal{T}_i \xrightarrow{\rho_{ij}} \mathcal{S}_{ij} \xleftarrow{\rho'_{ij}} \mathcal{T}_j$$

where  $(i, j) \in I \times I$ . The *heterogeneous parallel composition of the  $\mathcal{T}_i$ 's via  $\mathbf{R}$*  is the tag system

$$\mathbf{P} =_{\text{def}} \parallel_{\mathbf{R}, i \in I} P_i = (V, \mathcal{T}, \Sigma),$$

such that:

$$-V = \bigcup_{i \in I} V_i;$$

$$-\mathcal{T} = \prod_{\mathbf{R}, i \in I} \mathcal{T}_i;$$

— $\Sigma$  is the set of behaviors  $\sigma$  such that  $(v, n, \tau, x)$  is an event of  $\sigma$  iff:

1.  $\tau = \sqcup_{\mathbf{R}, i \in I} \tau_i$ , and
2.  $v \in V_i$  for some  $i \in I$  implies that  $(v, n, \tau_i, x)$  is an event of some behavior  $\sigma_i \in \Sigma_i$ .

Write  $\bowtie_{\mathbf{R}, i \in I} \sigma_i$  to indicate that the behaviors  $\sigma_i$  are unifiable, and  $\sigma = \sqcup_{\mathbf{R}, i \in I} \sigma_i$  to express that they unify into  $\sigma$ .

For  $\mathbf{P}$  as above, a tag structure  $S$  is called **P-consistent** iff  $S \ll S_{ij}$  for every pair  $(i, j)$ , cf. (8). Given a **P-consistent** tag structure  $S$ , denote by

$$\mathbf{P}_S \quad (20)$$

the homogeneous tag system having the same set of variables as  $\mathbf{P}$ , tag structure  $S$ , and a set of behaviors obtained by mapping all tags to  $S$ . This is well defined since, for every  $j$ ,  $S \ll S_{ij} \ll \mathcal{T}_i$ .

*Notation.* For convenience, when the morphisms  $\mathcal{T}_1 \xrightarrow{\rho_1} \mathcal{T} \xleftarrow{\rho_2} \mathcal{T}_2$  are understood, we shall sometimes use the following notation instead of (19):

$$P_1 \parallel_{\mathcal{T}} P_2, \text{ or, simply, } P_1 \parallel P_2, \quad (21)$$

when the two morphisms play no role in the considered context.

*Example 3.8 (Heterogenous parallel composition).* For the following examples, we use the tag structures introduced in Example 2.3.

1. Globally asynchronous locally synchronous (GALS) systems are made of synchronous components communicating via asynchronous FIFOs. This is captured by our heterogeneous parallel composition as follows. Take  $\mathcal{T}_1 = \mathcal{T}_2 = \mathcal{T}_{\text{synch}}$ , and consider  $P_1 \parallel_{\mathcal{T}_{\text{triv}}} P_2$ , where  $\mathcal{T} = \mathcal{T}_{\text{triv}}$  is the asynchronous tag structure. Referring to Example 2.5,  $P_s \parallel_{\mathcal{T}_{\text{triv}}} Q_s$  has a single nontrivial behavior obtained simply by pairing the two (different) behaviors of  $P_s$  and  $Q_s$ . This is in contrast with the homogeneous parallel composition  $P_s \parallel Q_s$ , which yields the empty set  $\emptyset$  as a result.
2. To model the interaction of a synchronous system  $P = (V, \mathcal{T}_{\text{synch}}, \Sigma)$  with its asynchronous environment  $A = (W, \mathcal{T}_{\text{triv}}, \Sigma')$  take the heterogeneous composition  $P \parallel_{\mathcal{T}_{\text{triv}}} A$ . Referring again to Example 2.5,  $P_s \parallel_{\mathcal{T}_{\text{triv}}} Q = Q$ .
3. To model the composition of timed systems with an untimed system, consider the heterogeneous system  $P \parallel_{\mathcal{T}_{\text{synch}}} Q$ , where  $P = (V, \mathcal{T}_{\text{synch}} \times \mathcal{T}_{\text{wcet}}, \Sigma)$  is a synchronous timed system, while  $Q = (W, \mathcal{T}_{\text{synch}}, \Sigma')$  is a synchronous but untimed system.

### 3.3 Dealing with Heterogeneous Systems

Ideally, heterogeneous parallel composition as defined above should be able to transparently producing a consistent  $P \parallel Q$  from two given systems  $P$  and  $Q$  obeying different models of composition and computation. Unfortunately, this is not possible without help from the user of the methodology. We conjecture that composing heterogeneous systems in general is not possible unless a particular rule is given as how to interpret the interaction occurring along the communication links. In fact, our heterogeneous parallel composition operator  $\parallel_{\rho_1, \rho_2}$  requires the user to provide the pair of desynchronization morphisms  $\rho_1$  and  $\rho_2$ . Now, Examples 3.3(2) and 3.5(4) reveal that these morphisms cannot be uniquely defined. Indeed, tag structures  $\mathcal{T}_{\text{tta}}$  and  $\mathcal{T}_{\text{synch}}$  can be synchronized by morphisms in different ways; in particular, our polling mechanism can be parameterized by its polling frequency.

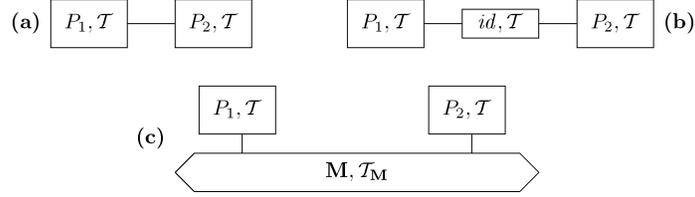
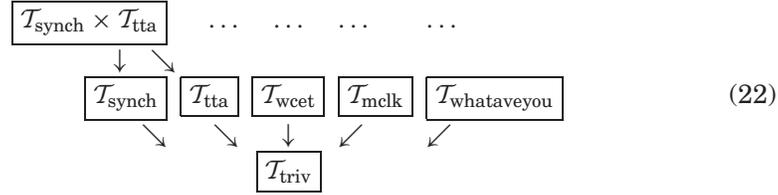


Fig. 4. A specification and its actual deployment: (a) specification; (b) same, but with an explicit “identity” channel; (c) deployment over a (possibly heterogeneous) communication medium. For each homogeneous system, we indicate its associated tag structure.  $\mathcal{T}_{\mathbf{M}}$  denotes a  $\mathbf{M}$ -consistent tag structure (see 20).

To ease the pain of the user of the methodology, a predefined dictionary of tag structures, with predefined heterogeneous parallel composition can be provided.<sup>2</sup> This is shown in the following diagram:



This diagram shows a DAG in which arrows indicate “standard” predefined morphisms. Each tag structure can be canonically mapped to the trivial tag structure  $\mathcal{T}_{\text{triv}}$ . Other standard morphisms shown in (22) are projections associated with composite tag structures. No other predefined morphism is offered in this dictionary, reflecting the fact that other morphisms can be user defined.

This approach fits a design methodology where tag structures are successively added and other removed, when tagging the events.

#### 4. APPLICATION TO CORRECT DEPLOYMENT

In this section, we apply our framework to the formalization of the requirement of “correct deployment.” This is an important concept with many practical application that, however, is often treated quite informally. We use the notational conventions (21).

##### 4.1 Preserving Semantics: Formalization

Diagram (a) in Figure 4 depicts a homogeneous specification  $P_1 \parallel P_2$ , where  $P_1 = (V_1, \mathcal{T}, \Sigma_1)$  and  $P_2 = (V_2, \mathcal{T}, \Sigma_2)$  possess identical tag structure  $(\mathcal{T}, \leq, \Phi)$ . Let  $W =_{\text{def}} V_1 \cap V_2$  be the set of shared variables of  $P_1$  and  $P_2$ . To prepare for distributed deployment, we wish to distinguish the shared variables seen from  $P_1$  or  $P_2$ , respectively. Formally, let  $W_1$  and  $W_2$  be two distinct copies of  $W$ . Rename each shared variable  $w$  of  $P_1$  by  $w_1 \in W_1$  and similarly for  $P_2$ . This renaming is modeled by the “renaming” channel having tag set  $\mathcal{T}$  and implementing  $w_1 = w_2$  for each  $w \in W$ :  $id = (W_1 \uplus W_2, \mathcal{T}, \Sigma)$ , where  $\Sigma$  is the

<sup>2</sup>This approach is consistent with the approach followed in Metropolis and Ptolemy, where heterogeneous composition can be provided as a library of methods.

set of behaviors such that, for each  $w \in W$ , the signals associated with  $w_1$  and  $w_2$  are equal. The homogeneous system  $P_1 \parallel id \parallel P_2$  is depicted in **(b)**.

When deploying the specification, the identity channel is replaced by a communication medium  $\mathbf{M}$ , which is a (possibly heterogeneous) tag system. Two semantics can be considered:

$$\begin{aligned} \text{the (homogeneous) specification semantics} & : \mathbf{S} = P_1 \parallel id \parallel P_2 \\ \text{the (heterogeneous) deployment semantics} & : \mathbf{D} = P_1 \parallel \mathbf{M} \parallel P_2 \end{aligned} \quad (23)$$

*Definition 4.1 (semantics preserving).* Say that  $P_1 \parallel \mathbf{M} \parallel P_2$  *simulates*  $P_1 \parallel id \parallel P_2$ , written  $P_1 \parallel \mathbf{M} \parallel P_2 \sqsubseteq P_1 \parallel id \parallel P_2$ , iff each pair of behaviors that is unifiable in the deployment is also unifiable in the specification, up to stretching. Formally,  $P_1 \parallel \mathbf{M} \parallel P_2 \sqsubseteq P_1 \parallel id \parallel P_2$  iff  $\forall(\sigma_1, \sigma_2) \in \Sigma_1 \times \Sigma_2, (i) \Rightarrow (ii)$  holds, where:

$$\begin{aligned} (i) : & \left. \begin{array}{l} \exists \sigma \in \Sigma_{\mathbf{D}} \\ \exists \varphi_1, \varphi_2 \in \Phi \end{array} \right\} \text{ s.t. } \mathbf{proj}_{V_1}(\sigma) = \varphi_1.\sigma_1 \text{ and } \mathbf{proj}_{V_2}(\sigma) = \varphi_2.\sigma_2 \\ (ii) : & \left. \begin{array}{l} \exists \sigma' \in \Sigma_{\mathbf{S}} \\ \exists \varphi'_1, \varphi'_2 \in \Phi \end{array} \right\} \text{ s.t. } \mathbf{proj}_{V_1}(\sigma') = \varphi'_1.\sigma_1 \text{ and } \mathbf{proj}_{V_2}(\sigma') = \varphi'_2.\sigma_2 \end{aligned}$$

We say that  $P_1 \parallel \mathbf{M} \parallel P_2$  is *semantics preserving* w.r.t.  $P_1 \parallel id \parallel P_2$ , written

$$P_1 \parallel \mathbf{M} \parallel P_2 \equiv P_1 \parallel id \parallel P_2 \quad (24)$$

iff both  $P_1 \parallel \mathbf{M} \parallel P_2 \sqsubseteq P_1 \parallel id \parallel P_2$  and  $P_1 \parallel \mathbf{M} \parallel P_2 \supseteq P_1 \parallel id \parallel P_2$  hold.

Perfect preserving, i.e., without the need for stretching in conditions (i) and (ii), cannot be achieved, in general, except of course when  $\Phi_1 = \Phi_2 = \{id\}$ .

For  $S \prec T$  and  $\rho : T \mapsto S$ , the heterogeneous parallel composition  $P_1 \parallel_S P_2$  can be seen as a particular case of deployment: we simply write

$$P_1 \parallel_S P_2 \equiv P_1 \parallel P_2 \quad (25)$$

iff  $P_1 \parallel_S (id, S) \parallel_S P_2 \equiv P_1 \parallel id \parallel P_2$  holds—note the heterogeneous parallel composition on the left-hand side, since  $(id, S)$  is the renaming channel with tag structure  $S$ .

*Example 4.2 (Semantics preserving).* Regarding semantics-preserving deployment, the following comments can be stated on Examples 2.5(2) and 2.5(3). The synchronous parallel composition of  $P_s$  and  $Q_s$  is equal to:  $P_s \parallel Q_s =_{\text{def}} P \cap Q = \emptyset$ . On the other hand,  $P_s \parallel_{\tau_{\text{triv}}} Q_s$  is just the pair  $(P_s, Q_s)$ . Thus, GALS deployment is not semantics preserving in this case.

An elegant solution to the problem of ensuring that semantics be preserved when replacing the ideal synchronous broadcast by the actual asynchronous communication, was proposed by Le Guernic and Talpin for the former GALS case [Le Guernic et al. 2003]. We cast their approach in the framework of tag systems and generalize it.

## 4.2 General Results on Correct Deployment

We first analyze requirement (25), and then we consider the more general case of (24).

*Analyzing Requirement (25).* Let  $\rho : \mathcal{T} \mapsto \mathcal{S}$  be a morphism and  $P = (V, \mathcal{T}, \Phi)$  be a tag system. Say that  $P$  is  $\rho$ -endochronous if the following holds: for every  $\sigma_S \in \Sigma_S$  (see (16) for notation  $P_S$ ), there exists  $\sigma \in \Sigma$  such that

$$\{\sigma' \in \Sigma \mid \sigma' \circ \rho = \sigma_S\} \subseteq \{\sigma'' \in \Sigma \mid \exists \varphi \in \Phi : \sigma'' = \varphi.\sigma\} \quad (26)$$

Condition (26) means that  $P$  can be “deterministically” reconstructed from  $P_S$ , since all inverse images of every  $\sigma_S \in \Sigma_S$  can be obtained by stretching a same behavior  $\sigma \in \Sigma$ .

**THEOREM 4.3.** *The pair  $(P_1, P_2)$  satisfies condition (25) if it satisfies the following two conditions:*

$$\forall i \in \{1, 2\}, P_i \text{ is } \rho\text{-endochronous} \quad (27)$$

$$(P_1 \parallel P_2)_S = (P_1)_S \parallel (P_2)_S \quad (28)$$

*Comments.* The primary application of this general theorem is when  $P$  and  $Q$  are synchronous systems, and  $\mathcal{S} = \mathcal{T}_{\text{triv}}$  is the tag set for asynchrony. This formalizes GALS deployment. Thus, Theorem 4.3 provides sufficient conditions to ensure correct GALS deployment. (See [Benveniste et al. 1999, 2000; Potop-Butucaru et al. 2004] for the concepts of *endochrony* and *isochrony*, which are related to conditions (27) and (28), respectively.

**PROOF.** See Appendix A.4; the proof uses the existence of complementary functions introduced in Definition 2.1.  $\square$

*Analyzing Requirement (24).* Here we investigate conditions ensuring (24). Using Definition 3.7, decompose the communication medium as  $\mathbf{M} = \parallel_{\mathbf{R}, i \in I} M_i$ . Let  $\mathcal{S}$  be a  $(P_1 \parallel \mathbf{M} \parallel P_2)$ -consistent tag set (see (20)), and let  $\mathcal{T}$  be the common tag set of  $P_i$ , for  $i = 1, 2$ . Note that  $\mathcal{S} \ll \mathcal{T}$ . The following theorem complements Theorem 4.3:

**THEOREM 4.4.** *The triple  $(P_1, \mathbf{M}, P_2)$  satisfies condition (24) if it satisfies the following two conditions:*

$$P_1 \parallel_S P_2 \equiv P_1 \parallel P_2 \quad (29)$$

$$\mathbf{M} \text{ is in bijection with } \mathbf{M}_S, \text{ and } \mathbf{M}_S = (id, \mathcal{S}) \quad (30)$$

where  $\mathbf{M}_S$  is defined in (20) and equality in (30) means that the two systems possess identical sets of behaviors when restricted to  $V_1 \cup V_2$  and local variables of  $\mathbf{M}$  being hidden.

Note that condition (29) is handled by Theorem 4.3.

**PROOF.** First, note that, by using the renaming convention of (23), condition (29) rewrites  $P_1 \parallel_S (id, \mathcal{S}) \parallel_S P_2 \equiv P_1 \parallel id \parallel P_2$ . Thus

$$P_1 \parallel_S \mathbf{M}_S \parallel_S P_2 \equiv P_1 \parallel id \parallel P_2 \quad (31)$$

follows, by the second statement of (30). Next, it is always true that  $P_1 \parallel \mathbf{M} \parallel P_2$  simulates  $P_1 \parallel_S \mathbf{M}_S \parallel_S P_2$ , i.e.,

$$P_1 \parallel \mathbf{M} \parallel P_2 \sqsubseteq P_1 \parallel_S \mathbf{M}_S \parallel_S P_2, \quad (32)$$

cf. Definition 4.1. Then, (31) and (32) and the first statement of (30) together complete the proof.  $\square$

Theorem 4.4 is a *separation theorem*: condition (29) does not directly involve the communication medium, since only tag set  $S$  and associated morphism  $\rho : \mathcal{T} \mapsto S$  play a role. On the other hand, condition (30) involves only the medium, not the application.

*Example 4.5 (Illustrating Theorem 4.3).* Here we propose some examples to illustrate Theorem 4.3. Theorem 4.4 will be illustrated in the next section.

1. Referring to Examples 2.5(2) and 2.5(3), since  $P_s$  and  $Q_s$  possess a single behavior, they clearly satisfy condition (27). However, the alternative condition (28) is violated: the left hand side is empty, while the right hand side is not. This explains why semantics is not preserved by desynchronization, for this example. In fact, it can be shown that the pair  $(P_s, Q_s)$  is not isochronous in the sense of Benveniste et al. [1999, 2000].
2. Our above example was a counterexample where condition (28) is violated. For the following counterexample, condition (27) is violated:  $P'_s$  possesses a local signal named  $x$ , and emits to  $Q'_s$  two signals with names  $y$  and  $z$ . Signal  $z$  is emitted each time  $x$  occurs, and signal  $y$  is emitted by  $P'_s$  if and only if  $x > 0$  (assuming, say,  $x$  integer). Signals  $y$  and  $z$  are awaited by  $Q'_s$ . Formally:

$$P'_s : \begin{cases} \sigma(x)(n) = (n, \bullet) \\ \sigma(y)(n) = (m(n), \bullet) \\ \text{where } m(n) = \min\{i \mid i > m(n-1) \wedge \sigma(x)(i) > 0\} \\ \sigma(z)(n) = (n, \bullet) \end{cases} \quad (33)$$

$$Q'_s : \begin{cases} \sigma(y)(n) = (l(n), \bullet) \\ \sigma(z)(n) = (k(n), \bullet) \end{cases}$$

In (33), symbol  $\bullet$  denotes an arbitrary value in the domain  $D$ , and  $k(\cdot), l(\cdot)$  are arbitrary strictly increasing maps, from  $\mathbf{N}$  to  $\mathbf{N}$ . As the reader can check,  $P'_s$  satisfies (27), but  $Q'_s$  does not. The desynchronization  $\alpha$  is not semantics preserving for this pair  $(P'_s, Q'_s)$ .

Now, consider the following modification of  $(P'_s, Q'_s)$ :  $P''_s$  possesses a local signal named  $x$ , and emits to  $Q''_s$  two signals with names  $y$  and  $z$ . Signal  $z$  is emitted each time  $x$  occurs, and signal  $y$  is emitted by  $P''_s$  if and only if  $x > 0$  (assuming, say,  $x$  integer). In addition,  $P''_s$  emits a Boolean guard  $b$  simultaneously with  $z$ , and  $b$  takes the value *true* iff  $x > 0$ . Signals  $y$  and  $z$  are awaited by  $Q''_s$ . In addition,  $Q''_s$  awaits the Boolean guard  $b$  simultaneously with  $z$ , and  $Q''_s$  knows that he should receive  $y$  simultaneously with

the *true* occurrences of  $b$ . Formally:

$$P_s'' : \begin{cases} \sigma(x)(n) = (n, \bullet) \\ \sigma(b)(n) = \text{if } \sigma(x)(n)(n) > 0 \text{ then } (n, t) \text{ else } (n, f) \\ \sigma(y)(n) = (m(n), \bullet) \\ \quad \text{where } m(n) = \min\{i \mid i > m(n-1) \wedge \sigma(x)(i) > 0\} \\ \sigma(z)(n) = (n, \bullet) \end{cases}$$

$$Q_s'' : \begin{cases} \sigma(b)(n) = (k(n), \bullet) \\ \sigma(y)(n) = (l(n), \bullet) \\ \quad \text{where } l(n) = \min\{k(i) \mid k(i) > l(n-1) \wedge \sigma(b)(i) = t\} \\ \sigma(z)(n) = (k(n), \bullet) \end{cases}$$

The guard  $b$  explicitly says when  $y$  must be awaited by  $Q_s''$ , this guarantees that  $Q_s''$  satisfies (27) (and so does  $P_s''$ ). On the other hand, the pair  $(P_s'', Q_s'')$  satisfies (28). Thus, the modified pair  $(P_s'', Q_s'')$  is semantics preserving for desynchronization. The modification, from  $(P_s, Q_s)$  to  $(P_s'', Q_s'')$ , was obtained by adding the explicit guard  $b$ . This can be made systematic, as outlined in Benveniste et al. [2000].  $\square$

## 5. DEPLOYING TIMED SYNCHRONOUS SPECIFICATIONS OVER LTTA

Loosely Time-Triggered Architectures (LTTA) were introduced in Benveniste et al. [2002] as a less constrained version of H. Kopetz' TTA [1997]. The reader is referred to references [Benveniste et al. 2002] for the motivations behind LTTA, and Buttazzo [2002], Section 4, for a related architecture based on asynchronous buffers. In this section, we provide the complete foundations for correct-by-construction deployment over LTTA. This complements the partial analysis provided in Benveniste et al. [2002]. The reader may find that the analysis provided to support LTTA looks straightforward. However, one should remember that, when this research goal was proposed in Caspi [2001], it was considered very hard to formally understand the engineering practice. Hence, an intrinsic value of our approach consists also of casting the problem into a framework where the fundamental issues emerge with clarity.

### 5.1 The LTTA Architecture

The LTTA protocol is illustrated in Figure 5. We consider three devices, the *writer*, the *bus*, and the *reader*, indicated by the superscripts  $(\cdot)^w$ ,  $(\cdot)^b$ , and  $(\cdot)^r$ , respectively. Each device is activated by its own, approximately periodic, clock. The different clocks are *not* synchronized. In the following specification, the different sequences written, fetched, or read, are indexed by the set  $\mathbf{N} = \{1, 2, 3, \dots, n, \dots\}$  of natural integers, and we reserve the index 0 for the initial conditions, whenever needed.  $\mathbf{N}$  will serve to index the successive events of each individual signal, exactly as in our model of Section 2.2.

**5.1.1 The Writer.** At the time  $t^w(n)$  of the  $n$ th tick of his clock, the writer generates a new value  $x^w(n)$  it wants to communicate and stores it in its private output buffer. Thus, at any time  $t$ , the writer's output buffer content

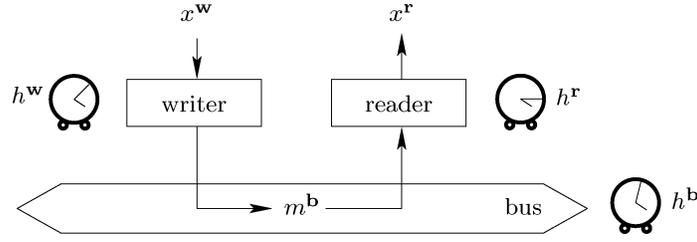


Fig. 5. The LTTA protocol.

$m^w$  is the last value that was written into it, that is the one with the largest index whose tick occurred before  $t$ :

$$m^w(t) = x^w(n),$$

$$\text{where } n = \sup\{n' \mid t^w(n') < t\}$$
(34)

**5.1.2 The Bus.** At the time  $t^b(n)$  of its  $n$ th clock tick, it fetches the value in the writer's output buffer and stores it, immediately after, in the reader's input buffer. Thus, at any time  $t$ , the reader's input buffer content offered by the bus, denote it by  $m^b$ , is the last value that was written into it, i.e., the one written at the latest bus clock tick preceding  $t$ :

$$m^b(t) = m^w(t^b(n)), \text{ where } n = \sup\{n' \mid t^b(n') < t\}$$
(35)

**5.1.3 The Reader.** At the time  $t^r(n)$  of its  $n$ -th clock tick, it copies the value of its input buffer into its output variable  $x^r(n)$ :

$$x^r(n) = m^b(t^r(n))$$
(36)

Call *LTTA-protocol* the protocol defined by formulas (34–36).

## 5.2 A Formal Tag System Model of LTTA

In this section, we study the preservation of semantics for multiple-channels, multiple-clocked synchronous systems.

**5.2.1 The Problem.** The situation is illustrated on Figure 6. In this figure, we show two multiple-clocked synchronous systems  $P$  and  $Q$ . The original model of communication is that of synchronous broadcast, in which tag is reaction index; this is shown in (a) and in (b) by making the identity channel explicit. The actual deployment is by means of the LTTA bus with its associated buffers, in which tag is physical time, this is shown in (c). In diagram (c), we show also the different tag structures involved in the LTTA medium. In the sequel, we shall denote by  $\mathcal{L}$  the tag system model of LTTA.

With reference to the predefined dictionary of tag structures (22),  $\mathcal{T}_{\text{triv}}$  is a natural candidate for a  $\mathcal{L}$ -consistent tag structure. Informally, switching from logical to physical time, and then back to logical time, destroys synchronization information. It is therefore nontrivial that LTTA deployment can preserve semantics.

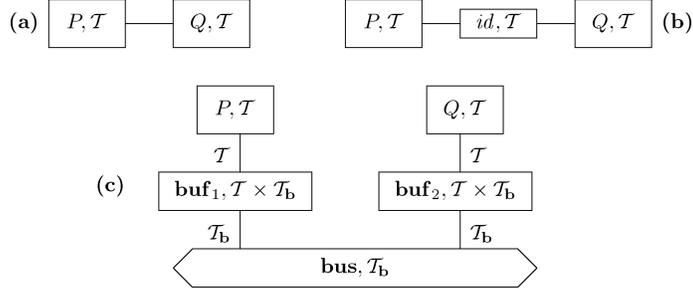


Fig. 6.  $LTTA$  deployment depicted as in Figure 4.  $\mathcal{T} = \mathcal{T}_{\text{synch}}$  captures logical time and  $\mathcal{T}_b = \mathcal{T}_{\text{tta}}$  models physical time from time-triggered systems (see Example 2.3). The symbols  $\mathcal{T}$  and  $\mathcal{T}_b$  sitting outside the boxes indicate the use of  $\parallel_{\mathcal{T}}$  and  $\parallel_{\mathcal{T}_b}$  heterogeneous parallel compositions, respectively.

**5.2.2 The Tag System  $\mathcal{L}^{w \rightarrow r}$ .** More precisely, our specification is  $P \parallel Q$ , with  $\parallel$  the synchronous parallel composition. The set of shared variables is  $V_P \cap V_Q$ . For convenience, we shall distinguish the instance of  $v \in V_P \cap V_Q$  that is attached to  $P$  by calling it  $v_P$ , and similarly for  $v_Q$ . Since the physical communication through  $LTTA$  medium takes time, it makes sense assuming that, for each individual shared variable, communication is directed, that is, for every pair  $(v_P, v_Q)$ , one of the two is an output and the other is an input. Suppose that  $v_P$  is an output of  $P$ , then the parallel composition can be reinterpreted as the directed assignment  $v_Q := v_P$ . This assignment can thus be seen as an instance of the generic single-channel communication  $r := w$ , where symbols  $r$  and  $w$  refer to *reader* and *writer*, respectively. We denote this ideal single-channel communication model by  $id^{w \rightarrow r}$ . Its actual deployment over  $LTTA$  is denoted by  $\mathcal{L}^{w \rightarrow r}$ . The latter decomposes as the following heterogeneous parallel composition:

$$\mathcal{L}^{w \rightarrow r} = P^w \parallel P^b \parallel P^r, \quad (37)$$

where:

— **$P^w$  is the writer buffer:** The writer buffer is a hybrid synchronous/timed tag system, described as follows.

- *tag structure:*  $\mathcal{T}^w = \mathcal{T}_{\text{synch}} \times \mathcal{T}_{\text{tta}} = \text{logical time} \times \text{physical time of TTA}$ . The former carries the synchronous semantics with its successive reactions, whereas the latter carries the timed semantics.
- *Variables:* the writer has a single variable  $w$ . The logical clock of  $w$  is a subclock of the activation clock of  $P^w$ .
- *Behaviors:*

$$\begin{aligned} \sigma(w)(k) &= ((m_k, t_{m_k}), x_k), \\ \text{where } t_m &= \lambda^w m + \varphi^w \end{aligned} \quad (38)$$

and where  $(\lambda^w, \varphi^w) = (\text{period, phase})$  of the writer buffer periodic clock. In (38),  $m_k$  is the index of the reaction  $\sigma(w)(k)$  belongs to, where  $m = 1, 2, \dots$  counts the reactions of the buffer (the map  $k \mapsto m_k$  is strictly increasing). Then,  $t_m$  is the physical date of the  $m$ th reaction of the buffer.

— **$P^r$  is the reader buffer:** Same comments as for the writer buffer.

- *tag structure:*  $\mathcal{T}^r = \mathcal{T}_{\text{synch}} \times \mathcal{T}_{\text{tta}} = \mathcal{T}^w$ .

- *Variables*: the reader has a single variable  $r$
- *Behaviors*:

$$\sigma(r)(\ell) = ((p_\ell, t_{p_\ell}), x_\ell), \quad (39)$$

$$\text{where } t_p = \lambda^r p + \varphi^r$$

and where  $(\lambda^r, \varphi^r) = (\text{period}, \text{phase})$  of the reader buffer periodic clock.

— **$P^b$  is the bus**: The bus is a purely timed system, described as follows. For  $e = (\tau, x)$  a pair (tag, value), we denote by  $x[e]$  the value carried by  $e$ .

- *tag structure*:  $T^b = \mathcal{T}_{\text{tta}} = \text{physical time of TTA}$ .
- *Variables*: the bus has three variables  $w, \xi, r$ , where  $\xi$  is local.
- *Behaviors*:

$$\sigma(w)(k) = (t_k^w, x_k)$$

$$\sigma(\xi)(n) = (t_n^\xi, x[\sigma(w)(k_n)]),$$

$$\text{where } k_n = \max\{k \mid t_k^w < t_n^\xi\} \quad (40)$$

$$\sigma(r)(n) = (t_n^r, x[\sigma(\xi)(n_\ell)]),$$

$$\text{where } n_\ell = \max\{n \mid t_n^\xi < t_\ell^r\}$$

and  $t_n^\xi = \lambda^b n + \varphi^b, (\lambda^b, \varphi^b) = (\text{period}, \text{phase})$  of the bus periodic clock.

### 5.3 Conditions for Correct-by-Construction Deployment over $\text{LTta}$

**5.3.1 Formal Modeling of Deployment.** We consider the two synchronous systems  $P$  and  $Q$ . For  $v \in V_P \cap V_Q$  a shared variable, we write  $v_P$  (resp.  $v_Q$ ) when referring to its instance in  $P$  (resp.  $Q$ ). Then,  $\text{out}_P$  denotes the set of outputs of  $P$ .

*The specification semantics  $\mathbf{S}$ .* It is given by

$$P \parallel \underbrace{(\|_{v \in \text{out}_P \cap V_Q} id^{v_P \rightarrow v_Q})}_{id: \text{ a bundle of directed synchronous identity channels}} \parallel \underbrace{(\|_{v \in V_P \cap \text{out}_Q} id^{v_Q \rightarrow v_P})}_{id: \text{ a bundle of directed synchronous identity channels}} \parallel Q. \quad (41)$$

Note that  $S$  defined in (41) is a (purely) synchronous system.

*The deployment semantics  $\mathbf{D}$ .* It is given by

$$P \parallel \underbrace{(\|_{v \in \text{out}_P \cap V_Q} \mathcal{L}^{v_P \rightarrow v_Q})}_{\mathcal{L}: \text{ a bundle of directed Ltta channels}} \parallel \underbrace{(\|_{v \in V_P \cap \text{out}_Q} \mathcal{L}^{v_Q \rightarrow v_P})}_{\mathcal{L}: \text{ a bundle of directed Ltta channels}} \parallel Q, \quad (42)$$

Here,  $\mathbf{D}$  defined in (42) is a hybrid system, consisting of two synchronous systems  $P$  and  $Q$  communicating via synchronous/timed system  $\mathcal{L}$ . The model given in (42) is somewhat inaccurate. It implicitly assumes that a bundle of  $\text{LTta}$  channels is indeed available, meaning that a new bus should be assigned to each peer communication. In practice, only one bus is available and the different peer communications are multiplexed. However, this time-division multiplexing is easily handled by a proper choice of the pair (period, phase).

**5.3.2 Preserving Semantics.** We shall use the general results of Section 4. Our communication medium is  $\mathcal{L}$ . It is a heterogeneous tag system that is a mix of timed/synchronous and purely timed system. On the other hand, the

application for deployment is purely synchronous. Therefore, the trivial tag set  $\mathcal{T}_{\text{triv}}$  is the natural candidate for a **D**-consistent tag structure. Using Theorem 4.4, we derive the following sufficient conditions for the LTTA deployment to preserve semantics:

$$P_1 \parallel_{\mathcal{T}_{\text{triv}}} P_2 \equiv P_1 \parallel P_2 \quad (43)$$

$$\mathcal{L} \text{ is in bijection with } \mathcal{L}_{\mathcal{T}_{\text{triv}}}, \text{ and } \mathcal{L}_{\mathcal{T}_{\text{triv}}} = (id, \mathcal{T}_{\text{triv}}) \quad (44)$$

Condition (43) involves only the robustness of deploying the pair  $(P_1, P_2)$  over a GALS architecture, it does not depend on LTTA. Condition (44) involves only LTTA, not the considered application. Since condition (43) has already been addressed elsewhere [Benveniste et al. 1999, 2000; Potop-Butucaru et al. 2004], we focus on (44).

To this end, recall the following result from Benveniste et al. [2002], we rephrase it slightly differently for convenience:

**THEOREM 5.1 (BENVENISTE ET AL. 2002).** *Assume the following condition for the respective periods of the writing / bus / reading systems:*

$$\lambda^{\mathbf{w}} \geq \lambda^{\mathbf{b}}, \quad \text{and} \quad \left\lfloor \frac{\lambda^{\mathbf{w}}}{\lambda^{\mathbf{b}}} \right\rfloor \geq \frac{\lambda^{\mathbf{r}}}{\lambda^{\mathbf{b}}}, \quad (45)$$

where, for  $x$  a real,  $\lfloor x \rfloor$  denotes the largest integer  $\leq x$ . Then, the reader misses no data sent by the writer. Formally, there exists a strictly increasing sequence  $k_n, n = 1, 2, \dots$  of integers such that, for each  $n$ :  $x_{k_n}^{\mathbf{r}} = x_{k_n}^{\mathbf{w}}$  and  $\forall k : k_n \leq k < k_{n+1} \Rightarrow x_k^{\mathbf{r}} = x_{k_n}^{\mathbf{r}}$ .

(In fact, a stronger result is proved in Benveniste et al. [2002], allowing for slight drifts and jitter with respect to strict periodicity.) Now, the problem of “excessive sampling” at the reader can be compensated for by associating a Boolean alternating flag to the data sent, so that switching of this flag marks, to the receiver, the successive instants  $k_n$  where correct sampling should occur. Thus, the map  $x_n^{\mathbf{w}} \mapsto x_{k_n}^{\mathbf{r}}$  is an asynchronous identity channel, i.e., it satisfies condition (44). Note that the  $k_n$  sequence is not periodic in general. The original presentation of the protocol in Benveniste et al. [2002] involved this flag from the beginning. We show here that its very reason is to enforce condition (44).

## 6. RELATED WORK ON FRAMEWORKS FOR HETEROGENEITY

In this section, we discuss related work. We devote a special section to the LSV model [Lee and Sangiovanni-Vincentelli 1998], as this model will allow us to position our approach in a larger context. Then, we review other work, including results from some of the authors of this article.

### 6.1 Comparison with the Original LSV Model [Lee and Sangiovanni-Vincentelli 1998]

According to the original LSV model, we assume an underlying partial order  $(\mathcal{T}, \leq)$  of *tags*. We assume also an underlying set  $\mathcal{V}$  of variables, with domain  $D$ . Elements of  $\mathcal{T} \times D$  are called *events*, and subsets of events are called *signals*. Thus, the set of all signals is  $S = \mathcal{P}(\mathcal{T} \times D)$ . Isomorphically, we also

have  $S = T \mapsto \mathcal{P}(D)$ . A *tag system* is simply a subset of the set  $\mathcal{V} \mapsto S$  of all *behaviors*, mapping variables to signals. Stretching is not considered. *Parallel composition* is by intersection and is restricted to tag systems having identical tag sets. This means that two signals can be unified if and only if they are identical (identical tags and identical data at each event). Similarly, two behaviors can be unified if and only if their shared signals are identical. Then, unification is by superimposition. Any coordination between heterogeneous communicating processes is captured by an additional process that functions as an “impedance-matching” element between two heterogeneous components. This element has as inputs the outputs of the sender and as outputs the inputs of the receiver. Then, unification by superpositions makes sense.

To compare our model with the original LSV model, consider the following specialization of the latter.

- Restricting to deterministic signals.* In the original LSV setting, a signal can take nondeterministically several values at a given tag, whereas this is forbidden in our model. Restricting the LSV model to deterministic signals can be enforced by requiring that the possible outcome at a given tag is either a single value or no value at all. With this restriction, the set of all LSV signals becomes  $S = T \mapsto D$ .
- Using composite tags.* When tags are composite, e.g.,  $(\mathcal{T}, \leq) = (\mathcal{T}_1, \leq_1) \times (\mathcal{T}_2, \leq_2)$ , we get  $S = \mathcal{T}_1 \times \mathcal{T}_2 \mapsto D$ . This is known to be isomorphic to  $S = \mathcal{T}_1 \mapsto (\mathcal{T}_2 \mapsto D)$ . Now, taking  $\mathcal{T}_1 = \mathbf{N}$  seems to yield exactly our model of signals, namely,  $S = \mathbf{N} \mapsto (\mathcal{T} \mapsto D)$ .

From the above specialization, the following conclusions can be drawn.

- When restricted to tag systems with  $\Phi = \{id\}$ , our model is a restricted class of LSV.* For example, asynchrony is modeled in LSV by  $(\mathcal{T}_1, \leq) = (\mathbf{N}, \leq)$  and  $(\mathcal{T}_2, \leq)$  is the trivial order. Tag  $\tau$  counts events in each individual signal; unification of behaviors in this model requires that signals with identical variable have identical length and values. Then, synchrony is modeled in LSV by  $(\mathcal{T}, \leq) = (\mathcal{T}_1, \leq_1) \times (\mathcal{T}_2, \leq_2)$ , where  $(\mathcal{T}_i, \leq_i), i = 1, 2$  are copies of  $\mathbf{N}$  with its natural order; tag  $\tau_1$  counts events in each individual signal, whereas the second tag  $\tau_2$  indexes reactions; unification of behaviors in this model requires that signals with identical variable have identical length and values, and reaction tag must be identical too—to have the second tag  $\tau_2$  strengthens the synchronization constraints and makes less behaviors unifiable.
- The comparison is less clear regarding the ordering of events via tags. We take the following interpretation of LSV: the partial ordering on events induced by tags is generally local to signals, not global. Doing this is essential if we want that asynchrony be captured as indicated above (otherwise tag  $\tau_1$  would relate events from different signals, a contradiction with our principle of asynchrony). In turn, referring to the model of synchrony, to have the second tag  $\tau_2$  local also causes the reaction tag not to be global—something less intuitive. In contrast, in our approach, *the counting index  $\mathbf{N}$  is not seen as a tag*: it is local to signals but tags can then be global and thus can order events from different signals (this one would expect from reaction tags).

- The way we handle heterogeneity also differs from LSV. In LSV, there is some underlying set  $\mathcal{T}_*$  of tags that “subsumes” all possible tag sets. Thus, one event can be tagged by an integer  $n$ , whereas another one can be tagged by a multiclock constraint with respect to other events. Our approach to handling heterogeneity is more constrained and more algebraic: we insist that tag structures should capture MoCCs. Blending MoCCs is achieved by using morphisms. This allowed deriving theorems and a theory for correct-by-construction deployment.
- Finally, we have introduced stretching as an essential mechanism. This proved instrumental in having the needed algebraic apparatus: fibered product of tag structures to capture heterogeneous MoCCs and the proper notion of heterogeneous parallel composition. Also, this allowed capturing nonfunctional characteristics as well as scheduling constraints.

## 6.2 Other Related Approaches

**6.2.1 Endochrony and Isochrony** [Benveniste et al. 1999, 2000; Potop-Butucaru et al. 2004]. The work on GALS deployment and its characterization via the properties of *endochrony* and *isochrony* has been the second main source of inspiration for the present work. Seen from this article, endochrony and isochrony are effective conditions that, respectively, imply conditions (27) and (28) of Theorem 4.3, for the particular case of  $P_1$  and  $P_2$  synchronous and  $S = \mathcal{T}_{\text{triv}}$ . Thus, endo/isochrony guarantees correct-by-construction GALS deployment. By “effective,” we mean that these conditions involve transitions systems and not sets of behaviors, thus they can be model checked and even synthesized. In Potop-Butucaru et al. [2004], these notions are improved toward handling concurrency and compositionality in a proper way.

**6.2.2 Interface Automata** [de Alfaro and Henzinger 2001a, 2001b; Chakrabarti et al. 2002; Lee and Xiong 2001]. *Interface automata* have been proposed in Alur and Henzinger [1999] and used in Halbwachs et al. [1991] for heterogeneous modeling within Ptolemy [Eker et al. 2003]. Interface automata are low-level finite state machines equipped with their usual synchronous product. They model in a generic way (an abstraction of) the micro-step interface behavior of different Ptolemy II domains. Interface automata are used as a typing mechanism between domains. Informally, if  $P$  and  $Q$  are models of respective domains  $D_P \neq D_Q$  and respective interface automata  $\mathcal{A}_P$  and  $\mathcal{A}_Q$ , then these two domains can be composed if the automaton  $\mathcal{A}_P \times \mathcal{A}_Q$  is deadlock free.

**6.2.3 Microstep Automata** [Potop-Butucaru and Caillaud 2005]. This recent work considers the GALS case. It improves the work on endo-/isochrony and has similarities with interface automata. Microstep automata provide a low-level operational description of the microstep execution of synchronous systems. The effective property of *weak endochrony* implies condition (27) of Theorem 4.3, for microstep automata. Weak endochrony handles concurrency in a proper way and is compositional. The simpler condition of deadlock freedom of the parallel composition replaces isochrony.

6.2.4 *The ForSyDe and SML-Sys Modeling Frameworks* [Mathaikutty et al. 2004; Sander and Jantsch 2004]. The formal system design (ForSyDe) provides a set of formal design-transformation methods for a transparent refinement process of a system model into an implementation model that is optimized for synthesis. Similarly to other modeling approaches, in ForSyDe the design process starts with a functional specification based on the synchronous paradigm. This system specification, which must be expressed in the Haskell language, is then refined through the stepwise application of formally defined design transformations. Designers can either select among a predefined set of basic transformation rules from a given library or define new transformation rules to capture the characteristics of target execution platform. The SML-Sys is based on the same approach as ForSyDe, but it uses instead the functional language Standard-ML and more importantly, it aims at enabling the use of heterogeneous models of computation at the specification phase [Mathaikutty et al. 2004]. Noticing that functional languages are not widely used in the industry mainly because of issues of efficiency and reusability, the authors of ForSyDe and SML-Sys have recently proposed a type-safe framework for the *untimed model of computation* (UMoC) that uses C++ while supporting a higher-order functional programming style [Mathaikutty et al. 2005].

These approaches are all based on semantics that are similar to the one used by Ptolemy and Metropolis in the sense that they are geared toward the *representation and simulation* of heterogeneous systems. Our approach is oriented toward a formal framework that can ensure properties that are analytically verified or correct by construction. In fact, our main results are in the area of theorems proving properties rather than execution models that can yield efficient simulation.

### 6.3 Tags Versus Detailed Low-level Expansions of MoCC Models

It is now a common approach to address heterogeneity in the following way. Systems obeying different MoCCs communicate through *interfaces*. How communication occurs via these interfaces is made explicit by giving, for each type of interface, a detailed automaton that schedules the different actions from and to the environment. Such interface models can be used in combination with low level models of communication media. The whole-system model is then obtained by composing together: (1) the subsystems with their different MoCCs, (2) the instantiated interfaces, and (3) the instantiated media. This approach is quite common and reasonably flexible, as long as one sticks to capturing variations in the behavioral style of functional or architectural MoCCs (synchronous, buffered, message-passing, etc.). However, introducing all these low-level models causes exponential blowup and prevents them from performing effective analyses at system scale. In contrast, our approach is of higher level and addresses the specific issue of correct deployment by providing more abstract and much more compact tools.

## 7. CONCLUSION

We developed a compositional theory of heterogeneous reactive systems based on tag systems that are inspired by the LSV-tagged signal model. Logical time,

physical time of various kinds, causalities, scheduling constraints, the simple local ordering of events of each individual signal, as well as their combination, can be captured by this formalism. We also developed a behavioral theory of heterogeneous architectures. We use it to formally study how to deploy a specification into an implementation, often a distributed architecture, so that their behaviors are equivalent. This framework makes it relatively straightforward to study formally the correctness of the design principles in use in aeronautics industries, based on LTTA type of architecture.

Our models are denotational; they deal only with “traces,” not with “agents” or “machines.” Therefore, their value is mostly in providing a mathematical machinery to prove theorems about the correctness of particular methods and to develop solid foundations to design. Introducing “tag machines” is the next step, allowing that algorithms and tools could be developed effectively to generate correct deployments.

## A. APPENDIX: PROOFS

### A.1 Proof of Lemma 2.2

Case 1 is trivial. For case 2, let  $(\varphi, \psi)$  be the considered pair of functions, and let  $\varphi(\mathcal{T})$  and  $\psi(\mathcal{T})$  be their ranges. Let  $\bar{\varphi}$  and  $\bar{\psi}$  be the two unique functions with respective domains  $\varphi(\mathcal{T})$  and  $\psi(\mathcal{T})$ , such that  $\bar{\varphi}(\varphi(\tau)) = \bar{\psi}(\psi(\tau)) = \max(\varphi(\tau), \psi(\tau))$ . Clearly,  $\bar{\varphi} \circ \varphi = \bar{\psi} \circ \psi$  and it remains to show that we can extend  $\bar{\varphi}$  and  $\bar{\psi}$  to total increasing functions on  $\mathcal{T}$ . Assume this is not the case. Then there exist, for example,  $\tau_1$  and  $\tau_2$  such that  $\varphi(\tau_1) \leq \varphi(\tau_2)$  but  $\bar{\varphi}(\varphi(\tau_1)) > \bar{\varphi}(\varphi(\tau_2))$ .

Suppose that  $\bar{\varphi}(\varphi(\tau_1)) = \varphi(\tau_1)$ ; this implies that  $\varphi(\tau_1) > \bar{\varphi}(\varphi(\tau_2))$ . Two cases can occur: Either  $\bar{\varphi}(\varphi(\tau_2)) = \varphi(\tau_2)$ , hence,  $\varphi(\tau_1) > \varphi(\tau_2)$ , a contradiction. Otherwise  $\bar{\varphi}(\varphi(\tau_2)) = \psi(\tau_2)$ , whence  $\varphi(\tau_2) > \psi(\tau_2)$ —a contradiction to the former. Therefore  $\bar{\varphi}(\varphi(\tau_1)) = \varphi(\tau_1)$  cannot occur.

This requires  $\bar{\varphi}(\varphi(\tau_1)) = \psi(\tau_1) > \bar{\varphi}(\varphi(\tau_2))$ . Again, two cases can occur: Either  $\bar{\varphi}(\varphi(\tau_2)) = \psi(\tau_2)$ , whence  $\psi(\tau_1) > \psi(\tau_2)$ —a contradiction with  $\psi(\tau_1) \leq \psi(\tau_2)$  since both  $\varphi$  and  $\psi$  are increasing. Or  $\bar{\varphi}(\varphi(\tau_2)) = \varphi(\tau_2)$ , whence  $\psi(\tau_1) > \varphi(\tau_2)$ , and thus  $\psi(\tau_2) > \varphi(\tau_2)$  since  $\psi$  is increasing, whence  $\bar{\varphi}(\varphi(\tau_2)) = \psi(\tau_2)$ —a contradiction. Thus, both cases are impossible and the considered extensions must exist.

Case 3 is handled similarly.  $\square$

### A.2 Proof of Lemma 3.2

The key point to be shown is that, if  $P$  is  $\Phi$ -stretching invariant, then  $P'$  is  $\Phi'$ -stretching invariant. Let  $\varphi' \in \Phi'$ . We need to show that  $\varphi'.\Sigma' \subseteq \Sigma'$ . However, we have  $\varphi'.\Sigma' = \varphi'.\rho.\Sigma = (\varphi' \circ \rho).\Sigma = (\rho \circ \varphi).\Sigma = \rho.\varphi.\Sigma \subseteq \rho.\Sigma = \Sigma'$ , where the third equality uses (6) and the inclusion follows from the stretching invariance of  $\Sigma$ .  $\square$

### A.3 Proof of Lemma 3.4

First note that the so-defined set of functions is closed under composition. Next, let  $\Phi$  be the set of stretching functions of  $\mathcal{T}$ . Let  $\varphi = (\varphi_1, \varphi_2)$  and  $\psi = (\psi_1, \psi_2)$

belong to  $\Phi_{1 \ \rho_1 \times \rho_2} \ \Phi_2$ ; we need to show the existence of two complementary elements  $\bar{\varphi}, \bar{\psi} \in \Phi_{1 \ \rho_1 \times \rho_2} \ \Phi_2$ .

For  $i = 1, 2$ , let  $\bar{\varphi}_i, \bar{\psi}_i$  be the complementary of  $\varphi_i, \psi_i$  in  $\Phi_i$ . We have:

$$\bar{\varphi}_1 \circ \varphi_1 = \bar{\psi}_1 \circ \psi_1, \quad \bar{\varphi}_2 \circ \varphi_2 = \bar{\psi}_2 \circ \psi_2 \quad (\text{A1})$$

However,  $\rho_1 \circ \bar{\varphi}_1 \circ \varphi_1 \neq \rho_2 \circ \bar{\varphi}_2 \circ \varphi_2$  in general, and similarly for the  $\psi_i$ 's. So, simply pairing the functions  $(\bar{\varphi}_1, \bar{\varphi}_2)$  and  $(\bar{\psi}_1, \bar{\psi}_2)$  will not yield complements for  $\varphi$  and  $\psi$  and we need further work.

By (7), since, for  $i = 1, 2$ ,  $\rho_i$  is a morphism,

$$\text{for } i \in \{1, 2\}, \exists \gamma_i \in \Phi : \gamma_i \circ \rho_i = \rho_i \circ \bar{\varphi}_i \circ \varphi_i = \rho_i \circ \bar{\psi}_i \circ \psi_i \quad (\text{A2})$$

where  $\Phi$  is the set of stretching functions of  $\mathcal{T}$ . Let  $(\bar{\gamma}_1, \bar{\gamma}_2) \in \Phi$  be the complementary pair of  $(\gamma_1, \gamma_2)$ . We have:

$$\bar{\gamma}_1 \circ \gamma_1 = \bar{\gamma}_2 \circ \gamma_2 \quad (\text{A3})$$

Next, using (6), there exist stretching functions  $\varepsilon_i \in \Phi_i, i = 1, 2$ , such that, for any pair  $(\tau_1, \tau_2) \in \mathcal{T}_1 \ \rho_1 \times \rho_2 \ \mathcal{T}_2$ , i.e., such that  $\rho_1(\tau_1) = \rho_2(\tau_2)$ :

$$\rho_1 \circ \varepsilon_1(\tau_1) = \bar{\gamma}_1 \circ \gamma_1 \circ \rho_1(\tau_1) = \bar{\gamma}_2 \circ \gamma_2 \circ \rho_2(\tau_2) = \rho_2 \circ \varepsilon_2(\tau_2) \quad (\text{A4})$$

where the midequality in (A4) uses (A3) and the fact that  $\rho_1(\tau_1) = \rho_2(\tau_2)$ . By definition of  $\Phi_{1 \ \rho_1 \times \rho_2} \ \Phi_2$ , (A4) implies

$$\varepsilon =_{\text{def}} (\varepsilon_1, \varepsilon_2) \in \Phi_{1 \ \rho_1 \times \rho_2} \ \Phi_2.$$

The last step of the proof consists in showing that there exist  $\bar{\varphi} \in \Phi_{1 \ \rho_1 \times \rho_2} \ \Phi_2$  such that

$$\bar{\varphi} \circ \varphi = \varepsilon \quad (\text{A5})$$

and similarly for  $\bar{\psi}$ . Note first that, for any  $(\tau_1, \tau_2) \in \mathcal{T}_1 \ \rho_1 \times \rho_2 \ \mathcal{T}_2$ , we have

$$\bar{\gamma}_1 \circ \rho_1 \circ \bar{\varphi}_1 \circ \varphi_1(\tau_1) = \bar{\gamma}_2 \circ \rho_2 \circ \bar{\varphi}_2 \circ \varphi_2(\tau_2) \quad (\text{A6})$$

By (6), for  $i = 1, 2$ ,

$$\exists \bar{\delta}_i \in \Phi_i : \rho_i \circ \bar{\delta}_i = \bar{\gamma}_i \circ \rho_i \quad (\text{A7})$$

Thus

$$\rho_1 \circ \bar{\delta}_1 \circ \bar{\varphi}_1 \circ \varphi_1(\tau_1) = \rho_2 \circ \bar{\delta}_2 \circ \bar{\varphi}_2 \circ \varphi_2(\tau_2) \quad (\text{A8})$$

Finally,

$$\bar{\varphi} =_{\text{def}} (\bar{\delta}_1 \circ \bar{\varphi}_1, \bar{\delta}_2 \circ \bar{\varphi}_2) \in \Phi_{1 \ \rho_1 \times \rho_2} \ \Phi_2$$

satisfies (A5). This proves the lemma.  $\square$

#### A.4 Proof of Theorem 4.3

For the proof, we use the notations involving morphisms instead of tag sets (see (16) and (19)). Thus, (28) rewrites

$$(P_1 \parallel P_2)_\rho = (P_1)_\rho \parallel (P_2)_\rho. \quad (\text{A9})$$

Simulation  $\sqsupseteq$  in (25) always hold, meaning that every pair of behaviors unifiable in the right-hand side of (25) is also unifiable in the left-hand side, up to stretching. Thus, it remains to show that, if the two conditions of Theorem 4.3 hold, then  $\sqsubseteq$  in (25) also holds. Set  $\mathbf{S} =_{\text{def}} P_1 \parallel P_2$  and  $\mathbf{D} =_{\text{def}} P_1 \parallel_S P_2$ . Pick a pair  $(\sigma_1, \sigma_2)$  of behaviors satisfying condition (1) of Definition 4.1. We want to show that  $(\sigma_1, \sigma_2)$  also satisfies condition (2) of Definition 4.1.

Now, assume (27) and (A9). Let  $\varphi_i, i \in \{1, 2\}$  be as in condition (1) of Definition 4.1. By definition of  $\rho \parallel \rho$ , the pair  $((\varphi_1.\sigma_1)_\rho, (\varphi_2.\sigma_2)_\rho)$  is unifiable in  $(P_1)_\rho \parallel (P_2)_\rho$ . Next, (A9) guarantees that  $(\varphi_1.\sigma_1)_\rho \sqcup (\varphi_2.\sigma_2)_\rho$  is a behavior of  $(P_1 \parallel P_2)_\rho$ . Hence, there must exist some pair  $(\sigma'_1, \sigma'_2)$  unifiable in  $P_1 \parallel P_2$ , such that  $(\sigma'_1 \sqcup \sigma'_2)_\rho = (\varphi_1.\sigma_1)_\rho \sqcup (\varphi_2.\sigma_2)_\rho$ . Consequently,  $((\sigma'_1)_\rho, (\sigma'_2)_\rho)$  is also unifiable in  $(P_1)_\rho \parallel (P_2)_\rho$ , and  $(\sigma'_1)_\rho \sqcup (\sigma'_2)_\rho = (\varphi_1.\sigma_1)_\rho \sqcup (\varphi_2.\sigma_2)_\rho$ . But  $(\sigma'_1)_\rho$  is the restriction of  $(\sigma'_1)_\rho \sqcup (\sigma'_2)_\rho$  to its events labeled by variables belonging to  $V_1$ , and similarly for  $(\sigma'_2)_\rho$ . Thus  $(\sigma'_i)_\rho = (\varphi_i.\sigma_i)_\rho$  for  $i = 1, 2$  follows. To summarize this step, we have shown the existence of a pair

$$(\sigma'_1, \sigma'_2) \text{ unifiable in } P_1 \parallel P_2, \text{ such that } (\sigma'_i)_\rho = (\varphi_i.\sigma_i)_\rho \text{ for } i = 1, 2. \quad (\text{A10})$$

Next, we use (27). There exist, for  $i = 1, 2$ , a behavior  $\sigma_i^o$  and two stretching functions  $\psi'_i$  and  $\psi_i$  belonging to  $\Phi$ , such that

$$\sigma'_i = \psi'_i.\sigma_i^o \text{ and } \varphi_i.\sigma_i = \psi_i.\sigma_i^o \quad (\text{A11})$$

Since  $\Phi$  is a set of stretching functions, there exist a pair  $(\bar{\psi}'_1, \bar{\psi}_1)$  that is complementary to  $(\psi'_1, \psi_1)$ :  $\bar{\psi}'_1 \circ \psi'_1 = \bar{\psi}_1 \circ \psi_1$ . Set  $\psi''_2 =_{\text{def}} \bar{\psi}'_1 \circ \psi'_2$ , and let  $(\bar{\psi}''_2, \bar{\psi}_2)$  be complementary to the pair  $(\psi''_2, \psi_2)$ . We have

$$\bar{\psi}_2 \circ \psi_2 = \bar{\psi}''_2 \circ \psi''_2 = \bar{\psi}''_2 \circ \bar{\psi}'_1 \circ \psi'_2. \quad (\text{A12})$$

In the sequel, the superscript over symbol “=” indicates the formula used to justify having the considered equality. We have

$$(\bar{\psi}_2 \circ \varphi_2).\sigma_2 \stackrel{(\text{A11})}{=} (\bar{\psi}_2 \circ \psi_2).\sigma_2^o \stackrel{(\text{A12})}{=} (\bar{\psi}''_2 \circ \bar{\psi}'_1 \circ \psi'_2).\sigma_2^o \stackrel{(\text{A11})}{=} (\bar{\psi}''_2 \circ \bar{\psi}'_1).\sigma'_2 \quad (\text{A13})$$

and

$$(\bar{\psi}''_2 \circ \bar{\psi}'_1 \circ \varphi_1).\sigma_1 \stackrel{(\text{A11})}{=} (\bar{\psi}''_2 \circ \bar{\psi}'_1 \circ \psi_1).\sigma_1^o = (\bar{\psi}''_2 \circ \bar{\psi}'_1 \circ \psi'_1).\sigma_1^o \stackrel{(\text{A11})}{=} (\bar{\psi}''_2 \circ \bar{\psi}'_1).\sigma'_1. \quad (\text{A14})$$

Since  $(\sigma'_1, \sigma'_2)$  is a unifiable pair, then so is the stretched pair  $(\sigma''_1, \sigma''_2) =_{\text{def}} ((\bar{\psi}''_2 \circ \bar{\psi}'_1).\sigma'_1, (\bar{\psi}''_2 \circ \bar{\psi}'_1).\sigma'_2)$ . This fact, together with (A13) and (A14), shows that the pair  $(\sigma_1, \sigma_2)$  satisfies condition (ii) of Definition 4.1, which proves the theorem.  $\square$

## ACKNOWLEDGMENT

The authors gratefully thank the reviewers for their constructive comments on a previous version of this article.

## REFERENCES

- ALUR, R. AND HENZINGER, T. A. 1999. Reactive modules. *Formal Meth. Syst. Design* 15, 7–48.
- ALUR, R., DANG, T., ESPOSITO, J., HUR, Y., IVANCIC, F., KUMAR, V., LEE, I., MISHRA, P., PAPPAS, G. J., AND SOKOLSKY, O. 2003. Hierarchical modeling and analysis of embedded systems. *Proc. IEEE* 91, 1, 11–28.
- BALARIN, F., WATANABE, Y., HSIEH, H., LAVAGNO, L., PASSERONE, C., AND SANGIOVANNI-VINCENTELLI, A. 2003. Metropolis: An integrated electronic system design environment. *IEEE Comput.* 36, 4, 45–52.
- BENVENISTE, A., CAILLAUD, B., AND LE GUERNIC, P. 1999. From synchrony to asynchrony. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*. Lecture Notes in Computer Science, vol. 1664, 162–177. Springer, Berlin.
- BENVENISTE, A., CAILLAUD, B., AND LE GUERNIC, P. 2000. Compositionality in dataflow synchronous languages: Specification & distributed code generation. *Inform. Comput.* 163, 125–171.
- BENVENISTE, A., CASPI, P., LE GUERNIC, P., MARCHAND, H., TALPIN, J.-P., AND TRIPAKIS, S. 2002. A protocol for loosely time-triggered architectures. In *Proceedings of the 2nd International Workshop on Embedded Software (EMSOFT'02)*. A. Sangiovanni-Vincentelli and J. Sifakis Eds., Lecture Notes in Computer Science, vol. 2491, 252–265, Springer, Berlin.
- BENVENISTE, A., CARLONI, L. P., CASPI, P., AND SANGIOVANNI-VINCENTELLI, A. L. 2003. Heterogeneous reactive systems modeling and correct-by-construction deployment. In *Proceedings of the 3rd International Conference on Embedded Software (EMSOFT'03)*. Lecture Notes in Computer Science, vol. 2855, Springer, Berlin.
- BENVENISTE, A., CASPI, P., EDWARDS, S., HALBWACHS, N., LE GUERNIC, P., AND DE SIMONE, R. 2003. The synchronous language twelve years later. *Proc. IEEE* 91, 1, 64–83.
- BERRY, G. 2000. *The Foundations of Esterel*. MIT Press, Cambridge, MA.
- BURCH, J., PASSERONE, R., AND SANGIOVANNI-VINCENTELLI, A. L. 2001. Overcoming heterophobia: Modeling concurrency in heterogeneous systems. *Proceedings of the 2nd International Conference on Application of Concurrency to System Design*.
- BUTTAZZO, G. 2002. Scalable applications for energy-aware processors. In *Proceedings of the 2nd International Workshop on Embedded Software (EMSOFT'02)*. A. Sangiovanni-Vincentelli and J. Sifakis Eds., Lecture Notes in Computer Science, vol. 2491, 153–165. Springer, Berlin.
- CARLONI, L. P., McMILLAN, K. L., AND SANGIOVANNI-VINCENTELLI, A. L. 2001. Theory of latency-insensitive design. *IEEE Trans. Comput.-Aid. Design Integr. Circuits Syst.* 20, 9, 1059–1076.
- CASPI, P. 2001. Embedded control: From asynchrony to synchrony and back. In *Proceedings of 1st International Workshop on Embedded Software (EMSOFT'01)*. T.A. Henzinger and C.M. Kirsch Eds., Lecture Notes in Computer Science, vol. 2211. Springer, Berlin, 80–96.
- CHAKRABARTI, A., DE ALFARO, L., HENZINGER, T. A., AND MANG, F. Y. C. 2002. Synchronous and bidirectional component interfaces. In *Proceedings of the 14th International Conference on Computer-Aided Verification (CAV'02)*, Lecture Notes in Computer Science, Springer, Berlin, 414–427.
- CORTADELLA, J., KONDRATYEV, A., LAVAGNO, L., AND SOTIRIOU, C. 2003. A concurrent model for de-synchronization. In *Proceedings of the International Workshop on Logic Synthesis*.
- DE ALFARO, L. AND HENZINGER, T. A. 2001a. Interface automata. In *Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE01)*.
- DE ALFARO, L., AND HENZINGER, T. A. 2001. Interface theories for component-based design. In *Proceedings of 1st International Workshop on Embedded Software (EMSOFT'01)*. T. A. Henzinger and C. M. Kirsch Eds., Lecture Notes in Computer Science, vol. 2211, Springer, Berlin, 32–49.
- EKER, J., JANNECK, J. W., LEE, E. A., LIU, J., LUDWIG, J., NEUENDORFFER, S., SACHS, S., AND XIONG, Y. 2003. Taming heterogeneity—The Ptolemy approach. *Proc. IEEE*, 91, 1, 127–144. <http://ptolemy.eecs.berkeley.edu/ptolemyII/>

- HALBWACHS, N., CASPI, P., RAYMOND, P., AND PILAUD, D. 1991. The synchronous data flow programming language LUSTRE. *Proc. IEEE* 79, 9, 1305–1320.
- KARSAI, G., SZTIPANOVITS, J., LEDECZI, A., AND BAPTY, T. 2003. Model-integrated development of embedded software. *Proc. IEEE* 91, 1, 127–144.
- KOPETZ, H. 1997. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishing, Novell, MA.
- LAMPORT, L. 1983. Specifying concurrent program modules. *ACM Trans. Prog. Lang. Sys.*, 5, 2, 190–222.
- LAMPORT, L. 2003. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison Wesley Professional, Reading, MA.
- LE GUERNIC, P., GAUTIER, T., LE BORGNE, M., AND LE MAIRE, C. 1991. Programming real-time applications with SIGNAL. *Proc. IEEE* 79, 9, 1326–1333.
- LE GUERNIC, P., TALPIN, J.-P., AND LE LANN, J.-C. 2003. Polychrony for system design. *J. Circuits Syst. Comput.*
- LEE, E. A. AND SANGIOVANNI-VINCENTELLI, A. 1998. A framework for comparing models of computation. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 17, 12, 1217–1229.
- LEE, E. A., AND XIONG, Y. 2001. System-level types for component-based design. In *Proceedings of 1st International Workshop on Embedded Software (EMSOFT'01)*. T. A. Henzinger and C. M. Kirsch Eds., Lecture Notes in Computer Science, vol. 2211. Springer, Berlin, 32–49.
- MATTERN, F. 1989. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel & Distributed Algorithms*. M. Cosnard et al. Eds., Elsevier Science, Berlin, 215–226.
- MATHAIKUTTY, D., PATEL, H., AND SHUKLA, S. 2004. A functional programming framework of heterogeneous model of computations for system design. In *Proceedings of the Forum on Specification and Design Languages (FDL)*.
- MATHAIKUTTY, D., PATEL, H., SHUKLA, S., AND JANTSCH, A. 2005. UMoC++: A C++-based multi-MoC modeling environment. In *Advances in Design and Specification Languages for SoCs—Selected Contributions from FDL05*, A. Vachoux, Ed. Chapter 7. Springer, Berlin.
- POTOP-BUTUCARU, D., CAILLAUD, B., AND BENVENISTE, A. 2004. Concurrency in synchronous systems. In *Proceedings of the 4th International Conference on Applications of Concurrency in System Design (ACSD)*. Hamilton, Canada.
- POTOP-BUTUCARU, D. AND CAILLAUD, B. 2005. Correct-by-construction asynchronous implementation of modular synchronous specifications. In *Proceedings of the 5th International Conference on Application of Concurrency to System Design (ACSD'05)*. 48–57.
- SANDER, I. AND JANTSCH, A. 2004. System modeling and transformational design refinement in ForSyDe. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* 23, 1, 17–32.
- WINSKEL, G. 1982. Event structure semantics for CCS and related languages. In *Proceedings of ICALP*. M. Nielsen and M. Schmidt Eds., Lecture Notes in Computer Science, vol. 140. Springer-Verlag, Berlin, 561–576.

Received March 2005; revised November 2005, October 2006; accepted March 2007