

Composing Heterogeneous Reactive Systems ^{*}

Albert Benveniste[†] Benoît Caillaud Luca P. Carloni[‡]
Paul Caspi[§] Alberto L. Sangiovanni-Vincentelli

12th May 2004

Abstract

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.

^{*}This research was supported in part by the European Commission under the projects COLUMBUS, IST-2002-38314, and ARTIST, IST-2001-34820, by the NSF under the project ITR (CCR-0225610), and by the GSRC.

[†]Corresponding author: A. Benveniste; A.B. and B.C. are with Irisa/Inria, Campus de Beaulieu, 35042 Rennes cedex, France; Albert.Benveniste@irisa.fr, <http://www.irisa.fr/sigma2/benveniste/>.

[‡]L.C. and A.S.V. are with U.C. Berkeley, CA 94720, USA; {lcarloni,alberto}@eecs.berkeley.edu, <http://www-cad.eecs.berkeley.edu/HomePages/{lcarloni,alberto}>

[§]P.C. is with Verimag, Centre Equation, 2, rue de Vignate, F-38610 Gieres, Paul.Caspi@imag.fr, <http://www.imag.fr/VERIMAG/PEOPLE/Paul.Caspi>

Contents

1	Introduction	3
2	Tagged Systems	4
2.1	Tagged systems and their (homogeneous) parallel composition . .	5
2.2	Modeling with Tags	6
3	Heterogeneous Systems	9
3.1	The Algebra of Tag Structures	9
3.2	Heterogeneous Parallel Composition	12
3.3	Heterogeneous Systems and Architectures	13
3.4	A Simple Example	14
4	Application to Correct Deployment	17
4.1	Preserving Semantics: Formalization	17
4.2	General Results on Correct Deployment	19
4.3	Examples	21
5	Deploying Timed Synchronous Specifications over LTTA	22
5.1	The LTTA Architecture	22
5.2	A Formal Tagged System Model of LTTA	23
5.3	Conditions for Correct-by-Construction Deployment over LTTA .	26
6	Conclusion	27

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 component operation, for example, when a digital controller is applied to a continuous-time plant. In addition, heterogeneity may appear across different levels of abstraction, when, for example, a synchronous-language specification of the design may be implemented as 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 this paper, we address this fundamental problem 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 [18, 21], the MoBIES project [1], the Model-Integrated Computing (MIC) framework [22], the Metropolis project [4, 12], *Interface Theories* [19], and the concepts of *endochrony* and *isochrony* [6, 7, 27, 29].

The basis of our compositional theory of heterogeneous reactive systems is the notion of *Tagged Systems*, a variation of Lee and Sangiovanni-Vincentelli's (LSV) Tagged-Signal Model [28]. The LSV model is a denotational approach where a system is modeled as a set of behaviours. Behaviours are sets of events. Each event is characterized by a data value and a tag. 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, tagged systems can be seen as a common formalism to express different models of 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 sub-systems that have different tag sets. Further, by introducing appropriate mappings between tag sets of systems with different coordination policy, we can state conditions under which the implementation of a synchronous design on distributed loosely-synchronized architectures maintains the same behaviour, i.e., the implementation is semantics preserving.

The main contributions presented here are:

- In Section 2, we introduce *Tagged Systems* as a mathematical model for heterogeneous systems. We first restrict our attention to tagged systems in which parallel composition is by intersection, i.e., two behaviours can be composed if they agree on variable, data value, and tag. This restriction is sufficient to handle semantics-preserving deployment on GALS architectures as extensively discussed in [8]. By making the unification rule for tags parameterized, we show how to deal with causality relations, scheduling constraints, and earliest execution times defined on each of the components.
- In Section 3, we go beyond homogeneous system by introducing an appropriate algebra of tag structures that allows us to define heterogeneous parallel composition. Also, we use concepts from category theory to handle properly the case of systems derived by composing multiple (more than two) heterogeneous components. This allows us define heterogeneous architectures as network of heterogeneous components connected by heterogeneous communication media.
- 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.
- in Section 5, we use the proposed framework to examine the deployment of a synchronous specification on a Loosely Time Triggered Architecture (LTTA) [10, 8]. This architecture is an important concept developed in collaboration with the aerospace industry where a precise notion of time and synchronous events cannot be guaranteed on the field due to the distributed nature of the target architecture. Our analysis shows that the deployment strategy proposed is indeed correct-by-construction.

2 Tagged Systems

In this section, we define tagged systems and their homogeneous parallel composition. This is based on the common notion of parallel composition *by intersection*, i.e.: two behaviours can be composed if they agree on their shared variables, call them *unifiable*. In this usual setting, unifiability is simply equality and unification is by superimposition. However, we relax here the request that unifiability corresponds to the equality of tags and we replace it with a notion of parameterizable unifiability that is modeled through a partial order relation. This becomes the key for capturing diverse models of computation.

2.1 Tagged systems and their (homogeneous) parallel composition

Throughout this paper, $\mathbf{N} = \{1, 2, \dots\}$ 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., $\forall x, x' \in X : x \leq_X x' \Rightarrow f(x) \leq_Y f(x')$.

Tag structures. A *tag structure* is a triple $(\mathcal{T}, \leq, \sqsubseteq)$, where \mathcal{T} is a set of *tags*, and \leq and \sqsubseteq are two partial orders.

Partial order \leq relates tags seen as time stamps. Call a *clock* any increasing function $(\mathbf{N}, \leq) \mapsto (\mathcal{T}, \leq)$.

Partial order \sqsubseteq , called the *unification order*, defines how to unify tags and is essential to express coordination among events. Write $\tau_1 \bowtie \tau_2$ to mean that there exists $\tau \in \mathcal{T}$ such that $\tau_i \sqsubseteq \tau$. We assume that any pair (τ_1, τ_2) of tags, such that $\tau_1 \bowtie \tau_2$ holds, possesses an upper bound. We denote it by $\tau_1 \sqcup \tau_2$. In other words, $(\mathcal{T}, \sqsubseteq)$ is a sup-semi-lattice. We call \bowtie and \sqcup the *unification relation* and *unification map*, respectively.

We assume that unification is causal with respect to partial order of time stamps: the result of the unification cannot occur prior in time than its constituents. Formally, if $\tau_1 \bowtie \tau_2$ is a unifiable pair then $\tau_i \leq (\tau_1 \sqcup \tau_2)$, for $i = 1, 2$. Equivalently:

$$\forall \tau, \tau' : \tau \sqsubseteq \tau' \Rightarrow \tau \leq \tau'. \quad (1)$$

Condition (1) has the following consequence: if $\tau_1 \leq \tau'_1$, $\tau_2 \leq \tau'_2$, $\tau_1 \bowtie \tau_2$, and $\tau'_1 \bowtie \tau'_2$ together hold, then $(\tau_1 \sqcup \tau_2) \leq (\tau'_1 \sqcup \tau'_2)$ must also hold.

This ensures that the system obtained via parallel composition preserves the agreed order of its components.

Tagged systems. Let \mathcal{V} be an underlying set of variables with domain D . For $V \subset \mathcal{V}$ finite, a *V-behaviour*, or simply behaviour, is an element:

$$\sigma \in V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D), \quad (2)$$

meaning that, for each $v \in V$, the n -th occurrence of v in behaviour σ 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 σ a behaviour, an *event* of σ is a tuple $(v, n, \tau, x) \in V \times \mathbf{N} \times \mathcal{T} \times D$ such that $\sigma(v)(n) = (\tau, x)$. Thus we can regard behaviours as sets of events. 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 increasing. Thus it is a clock and we call it the *clock of v in σ* . A *tagged system* is a triple $P = (V, \mathcal{T}, \Sigma)$, where V is a finite set of variables, \mathcal{T} is a tag structure, and Σ a set of V -behaviours.

Homogeneous parallel composition. Consider two tagged systems $P_1 = (V_1, \mathcal{T}_1, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}_2, \Sigma_2)$ with identical tag structures $\mathcal{T}_1 = \mathcal{T}_2 = \mathcal{T}$.

Let \sqcup be the unification function of \mathcal{T} . For two events $e = (v, n, \tau, x)$ and $e' = (v', n', \tau', x')$, define

$$\begin{aligned} e \bowtie e' &\text{ iff } v = v', n = n', \tau \bowtie \tau', x = x', \text{ and} \\ e \bowtie e' &\Rightarrow e \sqcup e' =_{\text{def}} (v, n, \tau \sqcup \tau', x). \end{aligned}$$

The unification map \sqcup and relation \bowtie extend point-wise to behaviours. Then, for σ a V -behaviour and σ' a V' -behaviour, define, by abuse of notation: $\sigma \bowtie \sigma'$ iff $\sigma|_{V \cap V'} \bowtie \sigma'|_{V \cap V'}$, and then

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

where $\sigma|_W$ denotes the restriction of behaviour σ to the variables of W . Finally, for Σ and Σ' two sets of behaviours, 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

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

2.2 Modeling with Tags

The concepts of tags allows us to express various models of computation as illustrated by the following examples. Further, tags facilitate the combination of diverse models of computation to model heterogeneous systems as discussed in Section 3.

We first discuss the case of parallel composition by intersection, with synchronous, asynchronous, and time-triggered systems as particular instances. Then, by making the unification rule for tags parameterized, we show how to deal with causality relations, scheduling constraints, and earliest execution times defined on each of the components.

Parallel composition by intersection of tagged systems corresponds to the following situation:

- the tag set \mathcal{T} is arbitrary;
- the unification function is such that $\tau \bowtie \tau'$ iff $\tau = \tau'$, and $\tau \sqcup \tau' =_{\text{def}} \tau$.

Modeling synchronous systems, asynchronous systems, and timed systems, with this framework is extensively discussed in [8]. We summarize here the main points.

Synchrony. To represent *synchronous systems* with our model, take $\mathcal{T}_{\text{synch}} = \mathbf{N}$ as tag structure, and require that all clocks are strictly increasing. The tag index set $\mathcal{T}_{\text{synch}}$ organizes behaviours into successive reactions, as explained next. Call *reaction* a maximal set of events of σ with identical τ . Since clocks are strictly increasing, no two events of the same reaction can have the same

variable. Regard a behaviour as a sequence of global 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.

The *activation clock* of behaviour σ 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 σ . If instant $n \notin h(\mathbf{N})$, then we say that σ is *silent* at instant n . In most models of synchronous languages (e.g., Lustre, Esterel), programs are such that all their behaviours possess $h = Id$ as activation clock: programs are never silent. Said differently, by convention, non-silent instants are “erased” from all clocks, since they are considered not useful (recall that for synchronous systems clocks capture logical, not physical time). This is a good model for closed systems.

Modeling open systems requires a different approach. In open systems, for any given sub-system, there exists another sub-system that is working while the former is “sleeping”. Thus, activation clocks must be local, not global, to be able to track the absence/presence of events in the corresponding sub-systems. Adequate models for open systems are stuttering-invariant systems we define next¹. Call *time change* any strictly increasing function $\rho : \mathcal{T}_{\text{synch}} \mapsto \mathcal{T}_{\text{synch}}$, and denote by $\mathbf{R}_{\text{synch}}$ the set of all time changes over $\mathcal{T}_{\text{synch}}$. Then a synchronous system $P = (V, \mathcal{T}_{\text{synch}}, \Sigma)$ is called *stuttering invariant* when it is invariant under time change, i.e.:

$$\forall \sigma \in \Sigma, \forall \rho \in \mathbf{R}_{\text{synch}} \Rightarrow \sigma^\rho \in \Sigma, \quad (5)$$

where $(v, n, \rho(\tau), x) \in \sigma^\rho \Leftrightarrow_{\text{def}} (v, n, \tau, x) \in \sigma$. Examples of stuttering invariant systems are the stallable processes of latency-insensitive design [14].

Time-triggered systems. Timed systems such as those used to model Time-Triggered Architectures (TTA) [24] are similar to synchronous systems. The main difference is that the reaction index is replaced by physical real-time. Formally, the associated tag structure is $\mathcal{T}_{\text{tta}} =_{\text{def}} \mathbf{R}_+$ with its usual order \leq for time stamping and the unification order is flat ($\tau \bowtie \tau'$ iff $\tau = \tau'$).

Asynchrony. The notion of asynchrony is vague. Any system that is not synchronous could be called asynchronous. However, we often want to restrict somewhat this notion to capture particular characteristics of the system in our modeling. In this paper, we take a very liberal interpretation for *asynchronous systems*. If we interpret a tag set as a constraint on the coordination of different signals of a system and the integer $n \in \mathbf{N}$ as the basic constraint of the sequence of events of the behaviour of a variable, then the most “coordination-unconstrained” system, i.e. the one with the highest degree of freedom in terms of choice of coordination mechanisms, could be considered an ideal asynchronous system. This translate into a model where the tag set does not give any information on the absolute or relative ordering of events. In a more formal way, to model asynchrony we choose $\mathcal{T} = \mathcal{T}_{\text{triv}} =_{\text{def}} \{\emptyset\}$, i.e. the trivial set consisting of a single element. Then, behaviours identify with elements $\sigma \in V \mapsto \mathbf{N} \mapsto D$.

¹Stuttering invariance is an important property advocated by Lamport [25].

Hence, the behaviour of an asynchronous system is a set of sequences of values with each sequence associated to a distinct variable.

So far we discussed parallel composition by intersection. The following examples require a more sophisticated way to unify tags.

Causalities and scheduling specifications. Causalities or scheduling specifications were integral part of the original LSV model. Hence, it should be fairly simple to cast them in the tagged systems as well. We consider causality as a relation between tags of different signals that is in between the null relation among events of different signals in our notion of asynchronous systems and the existence of reactions that impose strong equality constraints among tags of different signals. The intent is, for example, to state that “the 2nd occurrence of x depends on the 3rd occurrence of b ”. Define $\mathbf{N}_0 =_{\text{def}} \mathbf{N} \cup \{0\}$. Define a *dependency* to be a map: $\delta = \mathcal{V} \mapsto \mathbf{N}_0$. We denote by \mathcal{T}_{dep} the set of all dependencies, and we take \mathcal{T}_{dep} as our tag structure. Thus an event has the form $e = (v, n, \delta, x)$, with the following interpretation: event e has v as associated variable, it is ranked n among the events with variable v , and it depends on the event of variable w that is ranked $\delta(w)$. The special case $\delta(w) = 0$ is interpreted as the absence of dependency. We take the convention that, for $e = (v, n, \delta, x)$ an event, $\delta(v) = n - 1$. Thus, on $\sigma(v)$, the set of dependencies reproduces the ranking. \mathcal{T}_{dep} is equipped with the partial order defined by $\delta \leq \delta'$ iff $\forall v : \delta(v) \leq \delta'(v)$. Then we define the unification map \sqcup for this case (note that $\sqsubseteq = \leq$):

$$\delta \sqcup \delta' =_{\text{def}} \max(\delta, \delta'). \quad (6)$$

With this definition, behaviours become labelled preorders as explained next. For σ a behaviour, and e, e' two events of σ , write:

$$e' \rightarrow_{\sigma} e \quad \text{iff} \quad \begin{cases} e = (v, n, \delta, x) \\ e' = (v', n', \delta', x') \\ \delta(v') = n' \end{cases} \quad (7)$$

Note that when $n' > 0$ the condition $\delta(v') = n'$ makes this dependency effective. Definition (7) makes σ a labeled directed graph. Denote by \preceq_{σ} the transitive and reflexive closure of \rightarrow_{σ} . It is a preorder².

Earliest execution times. To capture the earliest execution times of concurrent systems take $\mathcal{T}_{\text{date}} =_{\text{def}} \mathbf{R}_+$, the set of non-negative real numbers. Thus a tag $\tau \in \mathcal{T}_{\text{date}}$ assigns a date, and we define

$$\tau \sqcup \tau' =_{\text{def}} \max(\tau, \tau').$$

Hence, \sqcup is here a total function. Composing two systems has the effect that the two components wait for the latest date of occurrence for each shared variable.

² We insist: “preorder”, not “partial order”—this should not be a surprise, since the superposition of two partial orders generally yields a preorder.

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.

Timed systems. Various classes of timed automata models have been proposed since their introduction by Alur and Dill [2]. In timed automata, dates of events are subject to constraints of the form $C : \tau \in \cup_{i \in I} \llbracket s_i, t_i \rrbracket$, where I is some finite set whose cardinality depends on the considered event, and $\llbracket = [$ or $($, and symmetrically for \rrbracket . The classes of timed automata differ by the type of constraints that can be expressed, and therefore they differ in their decidability properties. Nevertheless, they can all be described by the following kind of tagged system³.

Take $\mathcal{T}_{\text{timed}} =_{\text{def}} \text{Pow}(\mathbf{R}_+) \setminus \{\emptyset\}$, where Pow denotes powerset. Thus, a tag $\tau \in \mathcal{T}_{\text{timed}}$ assigns to each event a constraint on its possible dates of occurrence. Then, several definitions are of interest:

- $\tau_1 \bowtie \tau_2$ iff $\tau_1 \cap \tau_2 \neq \emptyset$, and $\tau_1 \sqcup \tau_2 = \tau_1 \cap \tau_2$. This is the straightforward definition that consists in regarding tags as constraints and combining them by taking their conjunction.
- the unification of tags is a total function, defined as follows: $\tau_1 \sqcup \tau_2 = \{\max(t_1, t_2) \mid t_1 \in \tau_1, t_2 \in \tau_2\}$. In this case, events are synchronized by waiting for the latest one.

3 Heterogeneous Systems

In this section, we first define the algebra of tag structures. Then, we formally define heterogeneous systems based on this algebra. Heterogeneous systems are obtained via the parallel composition of heterogeneous components by means of communication media with appropriate tagged structures. We first define heterogeneous parallel composition for two components. Since this composition lacks associativity, the definition of composition for three or more subsystems is non trivial. We resort to the use of morphisms in the tag algebra and a result on pullbacks from category theory to define general heterogeneous composition.

3.1 The Algebra of Tag Structures

Tag structures will be denoted either explicitly by a triple $(\mathcal{T}, \leq, \sqcup) = (\text{set}, \text{partial order}, \text{unification order})$, or simply by the symbol \mathcal{T} if the other items are understood.

³Our framework of tagged systems handles (infinite) behaviours and is not suited to investigate decidability properties. This explains why we can subsume all variants of timed automata into a unique tagged systems model.

Morphisms and desynchronization. Given two tag structures $\mathcal{T}, \mathcal{T}'$, call *morphism* a total map $\rho : \mathcal{T} \mapsto \mathcal{T}'$ which is increasing for both orders \leq and \sqsubseteq . Morphisms compose. Morphisms satisfy $\rho(\tau_1 \sqcup \tau_2) = \rho(\tau_1) \sqcup' \rho(\tau_2)$. Thus, morphisms preserve and possibly increase unifiability as they make more pairs of behaviours unifiable under parallel composition. 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 $\rho \circ \rho' = Id_{\mathcal{T}'}$, where \circ denotes the composition of functions. We do not distinguish isomorphic tag structures. Morphisms induce a preorder on tag structures:

$$\mathcal{T}' \preceq \mathcal{T} \quad \text{iff there exists a morphism } \rho : \mathcal{T} \mapsto \mathcal{T}'.$$

We now discuss how to use morphism to address the desynchronization problem of great interest in practical applications (i.e., the problem of mapping synchrony to asynchrony). Desynchronization is defined here as the operation of making the synchronization constraints less stringent. Since synchronization constraints are captured by a tag structure that forces events to belong to reactions, desynchronization corresponds to remove, at least partially, the reaction tags. Indeed, a complete desynchronization consists of mapping a synchronous system into an asynchronous one, that is, of mapping the tag structure of the original system (it is convenient to think of the case where the original synchronous system is the specification that captures the behaviour we wish to obtain) into the trivial tag structure of the asynchronous system that we can consider a very efficient (minimal overhead) implementation. In general, it is very rare that this radical desynchronization has the same behaviour of the specification. Hence, we are interested in the case where we can map the specification into a partially desynchronized system. Distributed architectures of practical interest have this characteristic. The nice point is that desynchronization can be captured by morphisms as defined above.

Examples. Take $\mathcal{T} = \mathcal{T}_{\text{synch}}$, $\mathcal{T}' = \mathcal{T}_{\text{triv}}$, and ρ is the morphism that maps all $\tau \in \mathcal{T}$ to the unique tag constituting \mathcal{T}' ; morphism ρ models desynchronization.

Take $\mathcal{T} = \mathbf{R}_+$, $\mathcal{T}' = \mathbf{N}$ with \bowtie being equality and \leq being the usual order. The integer part $\mathbf{R}_+ \ni x \mapsto \lfloor x \rfloor \in \mathbf{N}$ is a morphism. More generally, sampling mechanisms are morphisms.

A useful type of morphism is that of the canonical projections associated with products of tag structures, defined as follows: $(\mathcal{T}_1, \leq_1, \sqsubseteq_1) \times (\mathcal{T}_2, \leq_2, \sqsubseteq_2) =_{\text{def}} (\mathcal{T}_1 \times \mathcal{T}_2, \leq_1 \times \leq_2, \sqsubseteq_1 \times \sqsubseteq_2)$.

Fibred product. For $\mathcal{T}_1 \xrightarrow{\rho_1} \mathcal{T} \xleftarrow{\rho_2} \mathcal{T}_2$ two morphisms, define the *fibred product*:

$$\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2 =_{\text{def}} \{ (\tau_1, \tau_2) \in \mathcal{T}_1 \times \mathcal{T}_2 \mid \rho_1(\tau_1) = \rho_2(\tau_2) \}, \quad (8)$$

also written $\mathcal{T}_1 \times_{\mathcal{T}} \mathcal{T}_2$. The fibred product is equipped with the restriction of the product orders $\leq_1 \times \leq_2$ and $\sqsubseteq_1 \times \sqsubseteq_2$. The map $\rho : (\tau_1, \tau_2) \in \mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2 \mapsto \rho_1(\tau_1) \in \mathcal{T}$ defines the canonical morphism associated with this fibred product.

Examples. Note that $\mathcal{T} \xrightarrow{Id \times Id} \mathcal{T} = \mathcal{T}$. 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, $\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2$ is the product $\mathcal{T}'_1 \times \mathcal{T} \times \mathcal{T}'_2$.

Multiple fibred product. Formula (8) defines the fibred product for two tag structures. How can we generalize it for more than two? For instance,

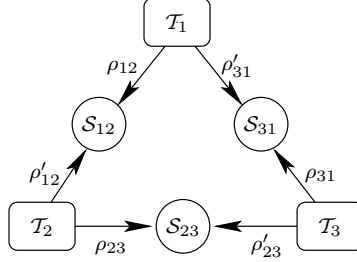


Figure 1: Morphism triangle.

consider the “triangle” shown in Fig. 1. This induces several fibred products. For instance:

$$(\mathcal{T}_1 \times_{\mathcal{S}_{12}} \mathcal{T}_2) \times_{\mathcal{S}_{23}} (\mathcal{T}_3 \times_{\mathcal{S}_{31}} \mathcal{T}_1) \quad \text{and} \quad ((\mathcal{T}_1 \times_{\mathcal{S}_{12}} \mathcal{T}_2) \times_{\mathcal{S}_{23}} \mathcal{T}_3) \times_{\mathcal{S}_{31}} \mathcal{T}_1.$$

Lemma 1 below related to ([20]–ch. 3, “pullback lemma”), ensures that

$$\text{these seemingly different fibred products are in fact all isomorphic.} \quad (9)$$

The same holds for every network of tags sets like in Fig. 1. This fact is essential to the modeling of heterogeneous architectures that we discuss in the next section.

The following lemma says that “the fibred product (8) defines a *pullback* [20] in the category of tag structures”. This result is essential in defining the heterogeneous composition of more than two components.

Lemma 1 *For any two morphisms $\rho_i : \mathcal{T}_i \mapsto \mathcal{T}, i = 1, 2$, let $(\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2)$ be their fibred product and ρ the associated canonical morphism. Then, for every two morphisms $\rho'_i : \mathcal{T}' \mapsto \mathcal{T}_i, i = 1, 2$ such that $\rho_1 \circ \rho'_1 = \rho_2 \circ \rho'_2$, there exists a unique morphism $\rho' : \mathcal{T}' \mapsto (\mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2)$ such that $\rho_i \circ \rho'_i = \rho \circ \rho'$, for $i = 1, 2$.*

Proof: First observe that pullbacks are defined in the category of sets exactly by using formula (8). But objects and morphisms in the category of tag structures induce objects and morphisms in the category of sets, by ignoring the partial order structure and unification orders on tag sets and related morphisms. This shows the existence of ρ' as a function satisfying the above factorizations. It remains to see that ρ' is in fact a morphism of tag structures, i.e., that $\rho' : 1/$ is increasing, and $2/$ commutes with the unification order. This is immediate.

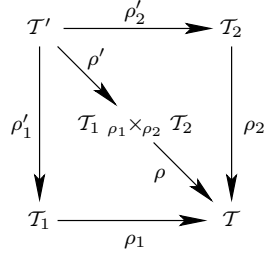


Figure 2: Pullback diagram for Lemma 1.

3.2 Heterogeneous Parallel Composition

In this section we define the composition of two tagged systems $P_1 = (V_1, \mathcal{T}_1, \Sigma_1)$ and $P_2 = (V_2, \mathcal{T}_2, \Sigma_2)$ when $\mathcal{T}_1 \neq \mathcal{T}_2$.

Given a morphism $\rho : \mathcal{T} \mapsto \mathcal{T}'$ and a behaviour $\sigma \in V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D)$, replacing τ by $\rho(\tau)$ in σ defines a new behaviour having \mathcal{T}' as tag structure. This behaviour is denoted as σ_ρ , or (with some abuse of notation) as $\sigma \circ \rho$. Performing this for every behaviour of a tagged system P with tag structure \mathcal{T} yields the tagged system

$$P_\rho, \text{ also denoted by } P_{\mathcal{T}'}. \quad (10)$$

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 (σ_1, σ_2) a pair satisfying $\sigma_1 \rho_1 \bowtie_{\rho_2} \sigma_2$, define

$$\sigma_1 \rho_1 \sqcup_{\rho_2} \sigma_2 \quad (11)$$

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, τ_i, x) is an event of σ_i , and 2/ (v, n, τ, 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 Σ_1 and Σ_2 by:

$$\Sigma_1 \wedge_{\rho_1 \rho_2} \Sigma_2 =_{\text{def}} \{ \sigma_1 \rho_1 \sqcup_{\rho_2} \sigma_2 \mid \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2, \sigma_1 \rho_1 \bowtie_{\rho_2} \sigma_2 \} \quad (12)$$

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

$$P_1 \parallel_{(\rho_1 \parallel \rho_2)} P_2 = (V_1 \cup V_2, \mathcal{T}_1 \times_{\rho_1 \times \rho_2} \mathcal{T}_2, \Sigma_1 \wedge_{\rho_1 \rho_2} \Sigma_2). \quad (13)$$

For convenience, when the morphisms ρ_1 and ρ_2 are understood, we prefer the following notation instead of (13), where \mathcal{T} is such that $\mathcal{T}_1 \xrightarrow{\rho_1} \mathcal{T} \xleftarrow{\rho_2} \mathcal{T}_2$:

$$P_1 \parallel_{\mathcal{T}} P_2 \quad (14)$$

Examples: GALS and Hybrid Timed/Untimed Systems. (The reader is referred to Section 2.2 for the below mentioned tag structures.)

For GALS, take $\mathcal{T}_1 = \mathcal{T}_2 = \mathcal{T}_{\text{synch}}$, where $\mathcal{T}_{\text{synch}} = \mathbf{N}$ is the tag structure of synchronous systems, and consider $P_1 \parallel_{\mathcal{T}_{\text{triv}}} P_2$, where $\mathcal{T} = \mathcal{T}_{\text{triv}}$ is the tag structure of asynchronous ones.

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 $P \parallel_{\mathcal{T}_{\text{triv}}} A$.

To model the composition of timed systems with untimed system, consider the heterogeneous system $P \parallel_{\mathcal{T}_{\text{synch}}} Q$, where $P = (V, \mathcal{T}_{\text{synch}} \times \mathcal{T}_{\text{date}}, \Sigma)$ is a synchronous timed system while $Q = (W, \mathcal{T}_{\text{synch}}, \Sigma')$ is a synchronous but untimed system.

3.3 Heterogeneous Systems and Architectures

So far we only defined heterogeneous parallel composition of two components. To capture more complex architectures we need to define it for a heterogeneous network of components. Fact (9) will be instrumental in doing this. The expression

$$P_1 \parallel_{\mathcal{S}_{12}} P_2 \parallel_{\mathcal{S}_{23}} P_3 \parallel_{\mathcal{S}_{31}} P_1, \quad (15)$$

where the repeated symbol P_1 refers to the same component, gives raise to several possible interpretations. We give only two of them:

$$(P_1 \parallel_{\mathcal{S}_{12}} P_2) \parallel_{\mathcal{S}_{23}} (P_3 \parallel_{\mathcal{S}_{31}} P_1) \text{ and } ((P_1 \parallel_{\mathcal{S}_{12}} P_2) \parallel_{\mathcal{S}_{23}} P_3) \parallel_{\mathcal{S}_{31}} P_1. \quad (16)$$

Thanks to (9), the two interpretations yield isomorphic tagged systems, i.e., tagged systems that are equal up to an isomorphism between their tag structures. This property is a restricted form of associativity⁴. It ensures that *expression (15) is well defined*.

Fig. 3 proposes a graphical notation for heterogeneous systems. In this

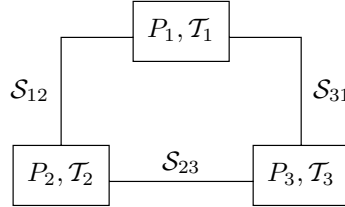


Figure 3: *A graphical notation for heterogeneous systems.*

figure, the pair (P_1, \mathcal{T}_1) specifies that the corresponding box refers to system P_1 having tag structure \mathcal{T}_1 . The label \mathcal{S}_{12} sitting aside the link between (P_1, \mathcal{T}_1) and (P_2, \mathcal{T}_2) denotes a tag structure such that $\mathcal{S}_{12} \preceq \mathcal{T}_i, i = 1, 2$. The presence of this label indicates that these two systems are composed via $P_1 \parallel_{\mathcal{S}_{12}} P_2$. In general, the components of the considered system are indexed by some set I (in Fig. 3, $I = \{1, 2, 3\}$). Collect the tag structures of the different communication

⁴Full fledged associativity cannot be considered, however: the first expression of (16) implicitly assumes for \mathcal{S}_{31} that $\mathcal{S}_{31} \preceq \mathcal{T}_3$ and $\mathcal{S}_{31} \preceq \mathcal{T}_1$; on the other hand, the second expression of (16) requires the strictly weaker conditions that $\mathcal{S}_{31} \preceq ((\mathcal{T}_1 \times_{\mathcal{S}_{12}} \mathcal{T}_2) \times_{\mathcal{S}_{23}} \mathcal{T}_3)$ and $\mathcal{S}_{31} \preceq \mathcal{T}_1$.

media into the matrix $\mathbf{S} =_{\text{def}} (\mathcal{S}_{ij})_{(i,j) \in I \times I}$. The heterogeneous architecture shown in Fig. 3 is denoted by

$$\mathbf{P} = \parallel_{\mathbf{S}, i \in I} P_i, \quad \text{or simply} \quad \mathbf{P} = \parallel_{i \in I} P_i, \quad (17)$$

if \mathbf{S} is understood. For \mathbf{P} as in (17), a tag structure \mathcal{T} is called **\mathbf{P} -consistent** iff $\mathcal{T} \preceq \mathcal{S}_{ij}$ for every pair (i, j) . Given a \mathbf{P} -consistent tag structure \mathcal{T} , denote by

$$\mathbf{P}_{\mathcal{T}} \quad (18)$$

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

3.4 A Simple Example

Here we illustrate the above constructions by means of a toy example and its many variants.

The base case: synchronous systems. Let P and Q be two synchronous systems involving the same set of variables: b of type boolean with values in $\{\text{F}, \text{T}\}$, and x of type integer. Each system possesses only a single behaviour, shown on the right hand side of $P : \dots$ and $Q : \dots$, respectively. Each behaviour 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.

P	:	<table style="border-collapse: collapse; border: none;"> <tr> <td style="padding: 2px 5px;">b</td> <td style="padding: 2px 5px;">:</td> <td style="padding: 2px 5px;">T</td> <td style="padding: 2px 5px;">F</td> <td style="padding: 2px 5px;">T</td> <td style="padding: 2px 5px;">F</td> <td style="padding: 2px 5px;">T</td> <td style="padding: 2px 5px;">F</td> <td style="padding: 2px 5px;">...</td> </tr> <tr> <td style="padding: 2px 5px;">x</td> <td style="padding: 2px 5px;">:</td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;"></td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;"></td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;"></td> <td style="padding: 2px 5px;">...</td> </tr> </table>	b	:	T	F	T	F	T	F	...	x	:	1		1		1		...
b	:	T	F	T	F	T	F	...												
x	:	1		1		1		...												
Q	:	<table style="border-collapse: collapse; border: none;"> <tr> <td style="padding: 2px 5px;">b</td> <td style="padding: 2px 5px;">:</td> <td style="padding: 2px 5px;">T</td> <td style="padding: 2px 5px;">F</td> <td style="padding: 2px 5px;">T</td> <td style="padding: 2px 5px;">F</td> <td style="padding: 2px 5px;">T</td> <td style="padding: 2px 5px;">F</td> <td style="padding: 2px 5px;">...</td> </tr> <tr> <td style="padding: 2px 5px;">x</td> <td style="padding: 2px 5px;">:</td> <td style="padding: 2px 5px;"></td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;"></td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;"></td> <td style="padding: 2px 5px;">1</td> <td style="padding: 2px 5px;">...</td> </tr> </table>	b	:	T	F	T	F	T	F	...	x	:		1		1		1	...
b	:	T	F	T	F	T	F	...												
x	:		1		1		1	...												

The single behaviour of P can be expressed formally in our framework as

$$\begin{aligned} \sigma(b)(2n-1) &= (2n-1, \text{T}) & , & & \sigma(b)(2n) &= (2n, \text{F}) \\ \sigma(x)(n) &= (2n-1, 1) \end{aligned} \quad (19)$$

where we take $\mathcal{T}_{\text{synch}} = \mathbf{N}$ to index the successive reactions. Note the absence of x at tag $2n$. Similarly, for Q we have the following where x is absent at tag $2n-1$:

$$\begin{aligned} \sigma(b)(2n-1) &= (2n-1, \text{T}) & , & & \sigma(b)(2n) &= (2n, \text{F}) \\ \sigma(x)(n) &= (2n, 1) \end{aligned} \quad (20)$$

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

Desynchronizing the base case. Informally, the *desynchronization* of a synchronous system like P or Q consists in (i) removing the synchronization barriers separating the successive reactions, and, then, (ii) compressing the sequence of values for each variable, individually. This yields:

$$P_\alpha = Q_\alpha : \frac{\begin{array}{cccccccc} b & : & \text{T} & \text{F} & \text{T} & \text{F} & \text{T} & \text{F} & \dots \end{array}}{\begin{array}{cccccccc} x & : & 1 & 1 & 1 & . & . & . & \end{array}}$$

where the subscript α refers to asynchrony. The reader may think that events having identical index for different variables are aligned, but this is not the case. In fact, as the absence of vertical bars in the diagram suggests, there is *no alignment* at all between events associated with different variables.

Formally, we express asynchrony by taking $\mathcal{T} = \mathcal{T}_{\text{triv}}$, the trivial set with a single element. The reason is that we do not need any additional time stamping information. Thus, the single behaviour of $P_\alpha = Q_\alpha$ is written as

$$\sigma_\alpha(b)(2n-1) = \text{T}, \sigma_\alpha(b)(2n) = \text{F}, \text{ and } \sigma_\alpha(x)(n) = 1. \quad (21)$$

Regarding desynchronization, the following comments are in order. Note that $P \neq Q$ but $P_\alpha = Q_\alpha$. Next, the synchronous system R defined by $R = P \cup Q$, the nondeterministic choice between P and Q , possesses two behaviours. However, its desynchronization R_α equals P_α , and possesses only one behaviour. Now, since $P_\alpha = Q_\alpha$, then $P_\alpha \parallel Q_\alpha \stackrel{\text{def}}{=} P_\alpha \cap Q_\alpha = P_\alpha = Q_\alpha \neq \emptyset$. Thus, for the pair (P, Q) , desynchronization does not preserve the semantics of parallel composition, in any reasonable sense.

Adding causalities. Suppose that some analysis of the considered program allows us to add the following causalities to P and Q :

$$\begin{array}{l} P_c : \\ Q_c : \end{array} \begin{array}{|c|c|c|c|c|c|c|c|} \hline b & : & \text{T} & \text{F} & \text{T} & \text{F} & \text{T} & \text{F} & \dots \\ \hline & & \downarrow & & \downarrow & & \downarrow & & \dots \\ \hline x & : & 1 & & 1 & & 1 & & \dots \\ \hline \hline b & : & \text{T} & \text{F} & \text{T} & \text{F} & \text{T} & \text{F} & \dots \\ \hline & & & \downarrow & & \downarrow & & \downarrow & \dots \\ \hline x & : & & 1 & & 1 & & 1 & \dots \\ \hline \end{array}$$

For example, in accordance to the above causalities, the meaning of P could be: if $b = t$ then get x (and similarly for Q). By using the compound tag structure $\mathcal{T}_{\text{synch}} \times \mathcal{T}_{\text{dep}}$, where \mathcal{T}_{dep} is the set of dependencies defined in (6), we can express formally the behaviour of P_c as

$$\begin{aligned} \sigma(b)(2n-1) &= ([2n-1, (x, 0)], \text{T}) & , & \sigma(b)(2n) &= ([2n, (x, 0)], \text{F}) \\ \sigma(x)(n) &= ([2n-1, (b, 2n-1)], 1) \end{aligned}$$

and the behaviour of Q_c as

$$\begin{aligned} \sigma(b)(2n-1) &= ([2n-1, (x, 0)], \text{T}) & , & \sigma(b)(2n) &= ([2n, (x, 0)], \text{F}) \\ & & , & \sigma(x)(n) &= ([2n, (b, 2n)], 1) \end{aligned}$$

As for the base case and for the same reason, $P_c \parallel Q_c = \emptyset$.

Then, desynchronizing. Removing the synchronization barriers from P_c and Q_c yields

$$\begin{array}{l}
P_c^\alpha : \quad \begin{array}{cccccccc}
\hline
b & : & T & F & T & F & T & F & \dots \\
& & \downarrow & & \downarrow & & \downarrow & & \dots \\
x & : & 1 & & 1 & & 1 & & \dots \\
\hline
\end{array} \\
Q_c^\alpha : \quad \begin{array}{cccccccc}
\hline
b & : & T & F & T & F & T & F & \dots \\
& & & \downarrow & & \downarrow & & \downarrow & \dots \\
x & : & & 1 & & 1 & & 1 & \dots \\
\hline
\end{array}
\end{array}$$

We insist that, again, desynchronizing consists in (i) removing the synchronization barriers, and then (ii) compressing the sequence of values for each variable, individually⁵. Formally, we project tags on \mathcal{T}_{dep} , this yields, for P_c^α :

$$\begin{aligned}
\sigma(b)(2n-1) &= ((x, 0), T) & , & & \sigma(b)(2n) &= ((x, 0), F) \\
\sigma(x)(n) &= ((b, 2n-1), 1)
\end{aligned}$$

and, for Q_c^α we have

$$\begin{aligned}
\sigma(b)(2n-1) &= ((x, 0), T) & , & & \sigma(b)(2n) &= ((x, 0), F) \\
& & & & \sigma(x)(n) &= ((b, 2n), 1)
\end{aligned}$$

According to the max rule (6) these two behaviours are unifiable and yield the dependency $(b, 2n)$. In fact, the reader can check that $P_c^\alpha \parallel Q_c^\alpha = Q_c^\alpha$. Thus P_c and Q_c did not include enough causalities for desynchronization to properly preserve the semantics.

Adding more causalities. Suppose that ‘‘oblique’’ causalities are added, from each previous occurrence of x to the current occurrence of b :

$$\begin{array}{l}
P_{cc} : \quad \begin{array}{cccccccc}
\hline
b & : & T & F & T & F & T & F & \dots \\
& & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & \nearrow & \dots \\
x & : & 1 & & 1 & & 1 & & \dots \\
\hline
\end{array} \\
Q_{cc} : \quad \begin{array}{cccccccc}
\hline
b & : & T & F & T & F & T & F & \dots \\
& & & \downarrow & \nearrow & \downarrow & \nearrow & \downarrow & \dots \\
x & : & & 1 & & 1 & & 1 & \dots \\
\hline
\end{array}
\end{array}$$

These supplementary causalities are conform to the synchronous model since they agree with the increasing reaction index. Formally, the single behaviour of P_{cc} is written

$$\begin{aligned}
\sigma(b)(2n-1) &= ([2n-1, (x, 0)], T) & , & & \sigma(b)(2n) &= ([2n, (x, n)], F) \\
\sigma(x)(n) &= ([2n-1, (b, 2n-1)], 1)
\end{aligned}$$

and the one of Q_{cc} is

$$\begin{aligned}
\sigma(b)(2n-1) &= ([2n-1, (x, n-1)], T) & , & & \sigma(b)(2n) &= ([2n, (x, 0)], F) \\
& & & & \sigma(x)(n) &= ([2n, (b, 2n)], 1)
\end{aligned}$$

Again, $P_{cc} \parallel Q_{cc} = \emptyset$.

⁵This last step is not shown on the drawing, just because it is a lot easier to draw vertical arrows.

Then, again desynchronizing. Removing the synchronization barriers from P_{cc} and Q_{cc} yields

$$\begin{array}{l}
 P_{cc}^\alpha : \\
 \hline
 b : \quad T \quad F \quad T \quad F \quad T \quad F \quad \dots \\
 \quad \quad \downarrow \quad \nearrow \quad \downarrow \quad \nearrow \quad \downarrow \quad \nearrow \quad \dots \\
 x : \quad 1 \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad \dots \\
 \hline
 Q_{cc}^\alpha : \\
 \hline
 b : \quad \quad T \quad F \quad \quad T \quad F \quad \quad T \quad F \quad \dots \\
 \quad \quad \quad \quad \downarrow \quad \nearrow \quad \quad \downarrow \quad \nearrow \quad \quad \downarrow \quad \dots \\
 x : \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad \dots \\
 \hline
 \end{array}$$

In our framework, for P_{cc}^α we have

$$\begin{aligned}
 \sigma(b)(2n-1) &= ((x, 0), T) & , & \quad \sigma(b)(2n) = ((x, n), F) \\
 \sigma(x)(n) &= ((b, 2n-1), 1)
 \end{aligned}$$

and, for Q_{cc}^α we have

$$\begin{aligned}
 \sigma(b)(2n-1) &= ((x, n-1), T) & , & \quad \sigma(b)(2n) = ((x, 0), F) \\
 & & , & \quad \sigma(x)(n) = ((b, 2n), 1)
 \end{aligned}$$

However, now the composed behaviour does not coincide with Q_{cc}^α but is

$$P_{cc}^\alpha \parallel Q_{cc}^\alpha = \begin{array}{l} \hline b : \quad \quad T \quad F \quad \quad T \quad F \quad \quad T \quad F \quad \dots \\ \quad \quad \quad \quad \downarrow \quad \nearrow \quad \quad \downarrow \quad \nearrow \quad \quad \downarrow \quad \dots \\ x : \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad \dots \\ \hline \end{array}$$

The reason for the double causality between x and F-occurrences of b is that the n -th x causes the $(2n)$ -th b (i.e. the n -th F-occurrence of b) in P_{cc} whereas the $(2n)$ -th b causes the n -th x in Q_{cc} . Formally, by the max rule (6), the composed behaviour of $P_{cc}^\alpha \parallel Q_{cc}^\alpha$ is written

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

In conclusion, $P_{cc}^\alpha \parallel Q_{cc}^\alpha$ possesses causality loops and may be considered pathological and thus “rejected” in accordance with the original semantics $P \parallel Q = \emptyset$.

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.

4.1 Preserving Semantics: Formalization

Diagram (a) in Fig. 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 set \mathcal{T} . Let $W \stackrel{\text{def}}{=} V_1 \cap V_2$ be the set of shared variables of P_1 and P_2 . To prepare for distributed deployment we wish to distinguish the shared variables as they are seen respectively

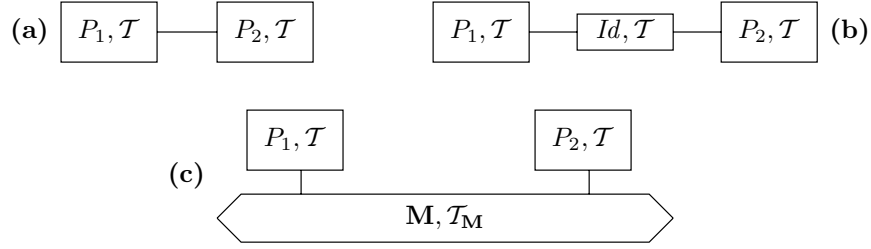


Figure 4: *A specification and its actual deployment: (a) specification, (b) same but with an explicit “identity” channel, (c) the deployment over a (possibly heterogeneous) communication medium with tag family \mathbf{T} .*

from P_1 or P_2 . 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 “identity” channel with tag set \mathcal{T} , implementing $w_1 = w_2$ for each $w \in W$: $Id = (W_1 \uplus W_2, \mathcal{T}, \Sigma)$, where Σ is the set of behaviours 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) tagged system as in (17). Two semantics can be considered:

$$\begin{aligned} \text{the specification semantics} & : \mathbf{S} = P_1 \parallel Id \parallel P_2 \\ \text{the deployment semantics} & : \mathbf{D} = P_1 \parallel \mathbf{M} \parallel P_2 \end{aligned} \quad (22)$$

where the latter involve morphisms since $P_i, i = 1, 2$ on the one hand, and \mathbf{M} on the other hand, possess in general different tag sets. In addition, \mathbf{M} may be a heterogeneous system like in (17).

Definition 1 (semantics preserving) *We say that $P_1 \parallel \mathbf{M} \parallel P_2$ simulates $P_1 \parallel Id \parallel P_2$, written $P_1 \parallel \mathbf{M} \parallel P_2 \leq P_1 \parallel Id \parallel P_2$, iff each pair of behaviours unifiable in the deployment is also unifiable in the specification. Formally, $P_1 \parallel \mathbf{M} \parallel P_2 \leq P_1 \parallel Id \parallel P_2$ iff $\forall (\sigma_1, \sigma_2) \in \Sigma_1 \times \Sigma_2$, (i) \Rightarrow (ii) holds, where:*

$$\begin{aligned} (i) & : \exists \sigma' \in \Sigma_{\mathbf{D}} \text{ s.t. } \mathbf{proj}_{V_1}(\sigma') = \sigma_1 \text{ and } \mathbf{proj}_{V_2}(\sigma') = \sigma_2 \\ (ii) & : \exists \sigma \in \Sigma_{\mathbf{S}} \text{ s.t. } \mathbf{proj}_{V_1}(\sigma) = \sigma_1 \text{ and } \mathbf{proj}_{V_2}(\sigma) = \sigma_2 \end{aligned}$$

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

$$P_1 \parallel \mathbf{M} \parallel P_2 \equiv P_1 \parallel Id \parallel P_2 \quad (23)$$

iff both $P_1 \parallel \mathbf{M} \parallel P_2 \leq P_1 \parallel Id \parallel P_2$ and $P_1 \parallel \mathbf{M} \parallel P_2 \geq P_1 \parallel Id \parallel P_2$ hold.

For $\mathcal{T}' \prec \mathcal{T}$ and $\rho : \mathcal{T} \mapsto \mathcal{T}'$, the heterogeneous parallel composition $P_1 \parallel_{\mathcal{T}'} P_2$ can be seen as a particular case of deployment: we simply write

$$P_1 \parallel_{\mathcal{T}'} P_2 \equiv P_1 \parallel P_2 \quad (24)$$

iff $P_1 \parallel (Id, T') \parallel P_2 \equiv P_1 \parallel Id \parallel P_2$ holds (note the heterogeneous parallel composition on the left hand side).

The simple example of Section 3.4 revisited. Regarding semantics preserving deployment, the following comments can be stated on our simple example. The synchronous parallel composition of P and Q , defined by intersection: $P \parallel Q =_{\text{def}} P \cap Q$, is empty. The reason is that P and Q disagree on where to put absences for the variable x . On the other hand, since $P_\alpha = Q_\alpha$, then $P_\alpha \parallel Q_\alpha =_{\text{def}} P_\alpha \cap Q_\alpha = P_\alpha = Q_\alpha \neq \emptyset$. Thus, for the pair (P, Q) , desynchronization does not preserve the semantics of parallel composition, in any reasonable sense. \diamond

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 [27]. We cast their approach in the framework of tagged systems and we generalize it.

4.2 General Results on Correct Deployment

We first analyze requirement (24), and then we consider the more general case of (23).

Analyzing requirement (24). Here we investigate conditions ensuring (24). See (10) for the notation $P_{T'}$ used in the following theorem:

Theorem 1 *The pair (P_1, P_2) satisfies condition (24) if it satisfies the following two conditions:*

$$\forall i \in \{1, 2\} : (P_i)_{T'} \text{ is in bijection with } P_i \quad (25)$$

$$(P_1 \parallel P_2)_{T'} = (P_1)_{T'} \parallel (P_2)_{T'} \quad (26)$$

Comments. The primary application of this general theorem is when P and Q are synchronous systems, and $T' = \mathcal{T}_{\text{triv}}$ is the tag set for asynchrony. This formalizes GALS deployment. Thus, Theorem 1 provides sufficient conditions to ensure correct GALS deployment. See [6, 7, 29] for the concepts of *endochrony* and *isochrony*, which are related to conditions (25) and (26), respectively.

Proof: For the proof we use the notations involving morphisms instead of tag sets, see (10) and (13). Inclusion \supseteq in (23) always hold, meaning that every pair of behaviours unifiable in the right hand side of (23) is also unifiable in the left hand side. Thus it remains to show that, if the two conditions of Theorem 1 hold, then inclusion \subseteq in (23) does too. Now, assume (25) and (26). Pick a pair (σ_1, σ_2) of behaviours which are unifiable in $P_1 \parallel_{(\rho)} P_2$. Then, by definition of $\parallel_{(\rho)}$, the pair $((\sigma_1)_\rho, (\sigma_2)_\rho)$ is unifiable in $(P_1)_\rho \parallel (P_2)_\rho$. Next, (26) guarantees that $(\sigma_1)_\rho \sqcup (\sigma_2)_\rho$ is a behaviour of $(P_1 \parallel P_2)_\rho$. Hence

there must exist some pair (σ'_1, σ'_2) unifiable in $P_1 \parallel P_2$, such that $(\sigma'_1 \sqcup \sigma'_2)_\rho = (\sigma_1)_\rho \sqcup (\sigma_2)_\rho$. Using the same argument as before, we derive that $((\sigma'_1)_\rho, (\sigma'_2)_\rho)$ is also unifiable with respect to its associated (asynchronous) parallel composition, and $(\sigma'_1)_\rho \sqcup (\sigma'_2)_\rho = (\sigma_1)_\rho \sqcup (\sigma_2)_\rho$. 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 $(\sigma'_i)_\rho = (\sigma_i)_\rho$ for $i = 1, 2$ follows. Finally, using (25), we know that if (σ'_1, σ'_2) is such that, for $i = 1, 2$: $(\sigma'_i)_\rho = (\sigma_i)_\rho$, then: $\sigma'_i = \sigma_i$. Hence (σ_1, σ_2) is unifiable in $P_1 \parallel P_2$. \diamond

Corollary 1 *Let P_1 and P_2 be synchronous systems whose behaviours are equipped with some equivalence relation \sim , and assume that P_1 and P_2 are closed with respect to \sim . Then, the pair (P_1, P_2) satisfies condition (23) if it satisfies the following two conditions:*

$$\forall i \in \{1, 2\} : (P_i)_{\mathcal{T}'} \text{ is in bijection with } P_i / \sim \quad (27)$$

$$(P_1 \parallel P_2)_{\mathcal{T}'} = (P_1)_{\mathcal{T}'} \parallel (P_2)_{\mathcal{T}'} \quad (28)$$

where P_i / \sim is the quotient of P_i modulo \sim .

Proof: Identical to the proof of Theorem 1 until the paragraph starting with “Finally”. Finally, using (27), we know that if (σ'_1, σ'_2) is such that, for $i = 1, 2$: $(\sigma'_i)_\rho = (\sigma_i)_\rho$, then: $\sigma'_i \sim \sigma_i$. Hence (σ_1, σ_2) is unifiable in $P_1 \parallel P_2$, since all synchronous systems we consider are closed under \sim . \diamond

This result is of particular interest when \sim is the equivalence modulo stuttering defined in (5).

Analyzing requirement (23). Here we investigate conditions ensuring (23). Using notation (17), decompose the communication medium as $\mathbf{M} = \parallel_{\mathbf{T}, i \in I} M_i$. Let $\mathcal{T}_{\mathbf{M}}$ be a $(P_1 \parallel \mathbf{M} \parallel P_2)$ -consistent tag set, and let \mathcal{T} be the common tag set of P_i , for $i = 1, 2$. Note that $\mathcal{T} \succeq \mathcal{T}_{\mathbf{M}}$. The following theorem complements Theorem 1 and its corollary:

Theorem 2 *The triple (P_1, \mathbf{M}, P_2) satisfies condition (23) if it satisfies the following two conditions:*

$$P_1 \parallel_{\mathcal{T}_{\mathbf{M}}} P_2 \equiv P_1 \parallel P_2 \quad (29)$$

$$\mathbf{M} \text{ is in bijection with } \mathbf{M}_{\mathcal{T}_{\mathbf{M}}}, \text{ and } \mathbf{M}_{\mathcal{T}_{\mathbf{M}}} = (Id, \mathcal{T}_{\mathbf{M}}) \quad (30)$$

where $\mathbf{M}_{\mathcal{T}_{\mathbf{M}}}$ is defined in (18) and equality in (30) means that the two systems possess identical sets of behaviours when restricted to the variables of P_1 or P_2 and local variables of \mathbf{M} being hidden.

Note that condition (29) is handled by Theorem 1 or Corollary 1.

Proof: First, note that, by using the renaming convention of (22), condition (29) rewrites $P_1 \parallel_{\mathcal{T}_M} (Id, \mathcal{T}_M) \parallel_{\mathcal{T}_M} P_2 \equiv P_1 \parallel Id \parallel P_2$. Thus

$$P_1 \parallel_{\mathcal{T}_M} \mathbf{M}_{\mathcal{T}_M} \parallel_{\mathcal{T}_M} P_2 \equiv P_1 \parallel Id \parallel P_2 \quad (31)$$

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

$$P_1 \parallel \mathbf{M} \parallel P_2 \preceq P_1 \parallel_{\mathcal{T}_M} \mathbf{M}_{\mathcal{T}_M} \parallel_{\mathcal{T}_M} P_2, \quad (32)$$

cf. Definition 1. Then, (31,32) and the first statement of (30) together complete the proof. \diamond

Theorem 2 is a “*separation theorem*”: condition (29) does (almost) not involve the communication medium, since only the greatest lower tag set \mathcal{T}_M and associated morphism $\rho : \mathcal{T} \mapsto \mathcal{T}_M$ play a role. On the other hand, condition (30) involves only the medium, not the application.

4.3 Examples

Here we propose some examples to illustrate Theorem 1. Theorem 2 will be illustrated in the next section.

The simple example of Section 3.4. Since P and Q possess a single behaviour, they clearly satisfy condition (25). However, the alternative condition (26) 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, Q) is not isochronous in the sense of [6, 7].

More examples and counter-examples. Our simple example was a counter-example where condition (26) is violated. For the following counter-example, condition (25) is violated: P possesses a local signal named x , and emits to Q two signals with names y and z . Signal z is emitted each time x occurs, and signal y is emitted by P if and only if $x > 0$ (assuming, say, x integer). Signals y and z are awaited by Q . Formally:

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

$$Q : \begin{cases} \sigma(y)(n) &= (l(n), -) \\ \sigma(z)(n) &= (k(n), -) \end{cases}$$

In (33), symbol $-$ 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 satisfies (25) but Q does not. The desynchronization α is not semantics preserving for this pair (P, Q) .

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

$$\begin{aligned}
P' & : \begin{cases} \sigma(x)(n) & = (n, -) \\ \sigma(b)(n) & = \text{if } \sigma(x)(n)(n) > 0 \text{ then } (n, t) \text{ else } (n, f) \\ \sigma(y)(n) & = (m(n), -) \\ & \text{where } m(n) = \min\{i \mid i > m(n-1) \wedge \sigma(x)(i) > 0\} \\ \sigma(z)(n) & = (n, -) \end{cases} \\
Q' & : \begin{cases} \sigma(b)(n) & = (k(n), -) \\ \sigma(y)(n) & = (l(n), -) \\ & \text{where } l(n) = \min\{k(i) \mid k(i) > l(n-1) \wedge \sigma(b)(i) = t\} \\ \sigma(z)(n) & = (k(n), -) \end{cases}
\end{aligned} \tag{34}$$

The guard b explicitly says when y must be awaited by Q' , this guarantees that Q' satisfies (25) (and so does P'). On the other hand, the pair (P', Q') satisfies (26). Thus the modified pair (P', Q') is semantics preserving, for desynchronization. The modification, from (P, Q) to (P', Q') , was obtained by adding the explicit guard b . This can be made systematic, as outlined in [7].

5 Deploying Timed Synchronous Specifications over LTTA

Loosely Time-Triggered Architectures (LTTA) were introduced in [10] as a less constrained version of H. Kopetz' TTA [24]. The reader is referred to references [10, 13] for the motivations behind LTTA. In this section, we provide the complete foundations for correct-by-construction deployment over LTTA. This complements the partial analysis provided in [10]. 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 [16], 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 where the three watches indicate a different time (*they are not synchronized*).

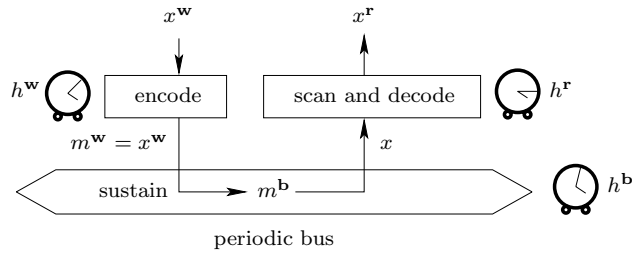


Figure 5: *The LTTA-protocol.*

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. Set \mathbf{N} will serve to index the successive events of each individual signal, exactly as in our model of Section 2.1.

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

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

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 :

$$m^b(t) = m^w(t^b(n)), \text{ where } n = \sup\{n' \mid t^b(n') < t\} \quad (36)$$

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 $x^r(n)$:

$$x^r(n) = m^b(t^r(n)) \quad (37)$$

Call *LTTA-protocol* the protocol defined by formulas (35,36,37).

5.2 A Formal Tagged System Model of LTTA

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

The problem. The situation is illustrated on Fig. 6. In this figure, we show

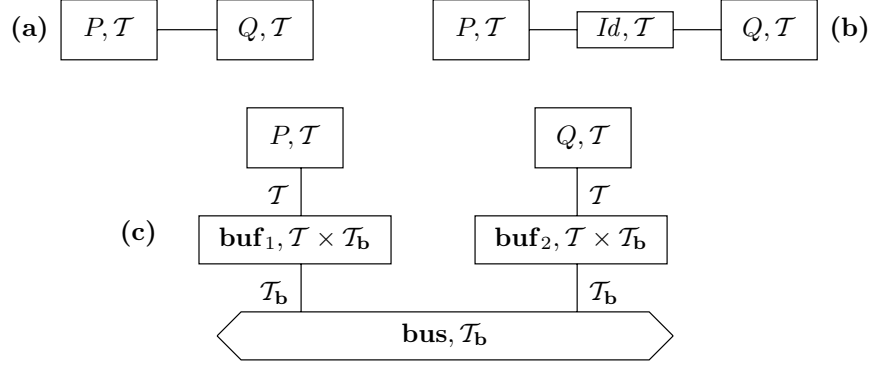


Figure 6: LTTA deployment depicted as in Fig. 4, with the conventions of Fig. 3. In all diagrams, $\mathcal{T} = \mathcal{T}_{\text{synch}} = \mathbf{N}$ captures logical time and $\mathcal{T}_b = \mathcal{T}_{\text{tta}} = \mathbf{R}_+$ models physical time from time-triggered systems, as introduced in Section 2.2.

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 tagged system model of LTTA.

Since the trivial tag structure $\mathcal{T}_{\text{triv}}$ associated to full asynchrony satisfies $\mathcal{T}_{\text{triv}} \preceq \mathcal{T}_{\text{synch}}$ and $\mathcal{T}_{\text{triv}} \preceq \mathcal{T}_{\text{tta}}$ and no other “known” tag structure satisfies these two conditions, $\mathcal{T}_{\text{triv}}$ is a natural candidate for a \mathcal{L} -consistent tag structure. Informally, switching from logical time to physical time, and then back to logical time, destroys synchronization information. It is therefore non trivial that LTTA deployment can preserve semantics.

The tagged 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, i.e., 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 re-interpreted 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:

$$\mathcal{L}^{w \rightarrow r} = P^w \parallel P^b \parallel P^r. \quad (38)$$

As the reader will see, (38) is a heterogeneous parallel composition, since the components of this parallel composition possess different tag structures. These components are:

P^w is the writer buffer: The writer buffer is a hybrid synchronous/timed tagged 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 (in the synchronous sense) of w is a sub-clock of the activation clock of P^w .
- *Behaviours:*

$$\sigma(w)(k) = ((m_k, t_{m_k}), x_k), \text{ where } t_m = \lambda^w m + \varphi^w \quad (39)$$

where $(\lambda^w, \varphi^w) = (\text{period}, \text{phase})$ of the writer buffer periodic clock. In (39), 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:* $\mathcal{T}^r = \mathcal{T}_{\text{synch}} \times \mathcal{T}_{\text{tta}} = \text{logical time} \times \text{physical time of TTA}$.
- *Variables:* the reader has a single variable r
- *Behaviours:*

$$\sigma(r)(\ell) = ((p_\ell, t_{p_\ell}), x_\ell), \text{ where } t_p = \lambda^r p + \varphi^r \quad (40)$$

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

$$\begin{aligned} \sigma(w)(k) &= (t_k^w, x_k) \\ \sigma(\xi)(n) &= (t_n^\xi, x[\sigma(w)(k_n)]), \text{ where } k_n = \max\{k \mid t_k^w < t_n^\xi\} \\ \sigma(r)(n) &= (t_n^r, x[\sigma(\xi)(n_\ell)]), \text{ where } n_\ell = \max\{n \mid t_n^\xi < t_n^r\} \end{aligned} \quad (41)$$

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 LTTA

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, out_P denotes the set of outputs of P .

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

$$P \parallel \underbrace{\left(\parallel_{v \in \text{out}_P \cap V_Q} Id^{v_P \rightarrow v_Q} \right) \parallel \left(\parallel_{v \in V_P \cap \text{out}_Q} Id^{v_Q \rightarrow v_P} \right)}_{Id: \text{ a bundle of directed synchronous identity channels}} \parallel Q. \quad (42)$$

Note that \mathcal{S} defined in (42) is a (purely) synchronous system.

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

$$P \parallel \underbrace{\left(\parallel_{v \in \text{out}_P \cap V_Q} \mathcal{L}^{v_P \rightarrow v_Q} \right) \parallel \left(\parallel_{v \in V_P \cap \text{out}_Q} \mathcal{L}^{v_Q \rightarrow v_P} \right)}_{\mathcal{L}: \text{ a bundle of directed Ltta channels}} \parallel Q, \quad (43)$$

Here, \mathbf{D} defined in (43) is a hybrid system, consisting of two synchronous systems P and Q communicating via synchronous/timed system \mathcal{L} . The model given in (43) is somewhat inaccurate. It implicitly assumes that a bundle of 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. But this time-division multiplexing is easily handled by a proper choice of the pair (period, phase).

Preserving semantics. We shall use the general results of Section 4. Our communication medium is \mathcal{L} , it is a heterogeneous tagged 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 \mathbf{D} -consistent tag structure. Using Theorem 2, we derive the following sufficient conditions for the LTTA deployment to preserve semantics:

$$P_1 \parallel_{\mathcal{T}_{\text{triv}}} P_2 \equiv P_1 \parallel P_2 \quad (44)$$

$$\mathcal{L} \text{ is in bijection with } \mathcal{L}_{\mathcal{T}_{\text{triv}}}, \text{ and } \mathcal{L}_{\mathcal{T}_{\text{triv}}} = (Id, \mathcal{T}_{\text{triv}}) \quad (45)$$

Condition (44) involves only the robustness of deploying the pair (P_1, P_2) over a GALs architecture, it does not depend on LTTA. Condition (45) involves only LTTA, not the considered application. Since condition (44) has already been addressed elsewhere [6, 7, 29], we focus on (45).

To this end, recall the following result from [10], we rephrase it slightly differently for convenience:

Theorem 3 ([10]) *Assume the following condition for the respective periods of the writing/bus/reading systems:*

$$\lambda^w \geq \lambda^b \quad , \text{ and } \quad \left\lfloor \frac{\lambda^w}{\lambda^b} \right\rfloor \geq \frac{\lambda^r}{\lambda^b} \quad , \quad (46)$$

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 : $x_{k_n}^r = x_n^w$ and $\forall k : k_n \leq k < k_{n+1} \Rightarrow x_k^r = x_{k_n}^r$.

(In fact, a stronger result is proved in [10], 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. Note that the k_n sequence is not periodic in general. The original presentation of the protocol in [10] involved this flag from the beginning. We show here that its very reason is to enforce condition (45).

6 Conclusion

We developed a compositional theory of heterogeneous reactive systems based on tagged systems, an extension of 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 behavioural theory of heterogeneous architectures. We use it to study formally how to deploy a specification into an implementation, often a distributed architecture, so that their behaviours are equivalent. This framework make it relatively straightforward to study formally the correctness of the design principles in use at Airbus, based on the LTTA 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. We are about to extend the approach to an agent-based framework so that tools could be developed effectively to generate correct deployments.

References

- [1] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas and O. Sokolsky. Hierarchical Modeling and Analysis of Embedded Systems. *Proc. of the IEEE*, 91(1), 11–28, Jan. 2003.
- [2] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comp. Science*, 126(2), 183–235, Apr. 1994.

- [3] ARTIST Network of Excellence. Roadmap on Hard Real-Time Development Environments. Available in may 2003 from url <http://www.systemes-critiques.org/ARTIST/>.
- [4] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone and A. Sangiovanni-Vincentelli. Metropolis: an Integrated Electronic System Design Environment. *IEEE Computer* 36(4):45-52, Apr 2003.
- [5] A. Benveniste. Some synchronization issues when designing embedded systems from components. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT'01*, T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 32-49, Springer.
- [6] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J.C.M. Baeten and S. Mauw, Eds., *CONCUR'99, Concurrency Theory, 10th Intl. Conference*, LNCS 1664, pages 162-177. Springer, Aug. 1999.
- [7] A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in dataflow synchronous languages: specification & distributed code generation. *Information and Computation*, 163, 125-171 (2000).
- [8] A. Benveniste, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli. Heterogeneous reactive systems modeling and correct-by-construction deployment. In R. Alur and I. Lee, Eds., *Proc. of the Third Intl. Conf. on Embedded Software, EMSOFT 2003*, LNCS 2855, Springer, 2003.
- [9] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The Synchronous Language Twelve Years Later. *Proc. of the IEEE*, 91(1):64-83, January 2003.
- [10] A. Benveniste, P. Caspi, P. Le Guernic, H. Marchand, J-P. Talpin and S. Tripakis. A Protocol for Loosely Time-Triggered Architectures. In *Embedded Software. Proc. of the 2nd Intl. Workshop, EMSOFT 2002*, A. Sangiovanni-Vincentelli and J. Sifakis Eds., Grenoble, France, Oct. 2002. LNCS vol. 2491, 252-265, Springer Verlag.
- [11] G. Berry. The Foundations of Esterel. MIT Press, 2000.
- [12] J. Burch, R. Passerone and A. L. Sangiovanni-Vincentelli. Overcoming Heterophobia: Modeling Concurrency in Heterogeneous Systems, *Proc. of the 2nd. Intl. Conf. on Application of Concurrency to System Design*. June 2001
- [13] G. Buttazzo. Scalable Applications for Energy-Aware Processors. In *Embedded Software. Proc. of the 2nd Intl. Workshop, EMSOFT 2002*, A. Sangiovanni-Vincentelli and J. Sifakis Eds., Grenoble, France, Oct. 2002. LNCS vol. 2491, 153-165, Springer Verlag.

- [14] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of Latency-Insensitive Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, September 2001.
- [15] P. Caspi. Clocks in Dataflow languages. *Theor. Comp. Science*, 94:125–140, 1992.
- [16] P. Caspi. Embedded control: from asynchrony to synchrony and back. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT’01*, T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 80–96, Springer.
- [17] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. A concurrent model for de-synchronization. In *Proc. Intl. Workshop on Logic Synthesis*, May 2003.
- [18] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, J. Ludwig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity—The Ptolemy approach. *Proc. of the IEEE*, 91(1), 127–144, Jan. 2003.
- [19] L. de Alfaro and T.A. Henzinger. Interface Theories for Component-Based Design. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT’01*, T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 32–49, Springer, 2001.
- [20] R. Goldblatt. *Topoi, the categorical analysis of logic*. Studies in logic and the foundations of mathematics, Vol. 98, North-Holland, 1984.
- [21] E.A. Lee and Y. Xiong. System-Level Types for Component-Based Design. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT’01*, T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 32–49, Springer, 2001.
- [22] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. *Proc. of the IEEE*, 91(1), 127–144, Jan. 2003.
- [23] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Data Flow Programming Language LUSTRE. *Proc. of the IEEE*, 79(9):1305–1320, Sep. 1991.
- [24] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers. 1997. ISBN 0-7923-9894-7.
- [25] L. Lamport, “Specifying concurrent program modules”, *ACM Trans. on Prog. Lang. and Sys.*, 5(2):190-222, 1983.
- [26] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with SIGNAL. *Proc. of the IEEE*, 79(9):1326–1333, Sep. 1991.
- [27] P. Le Guernic, J.-P. Talpin, J.-C. Le Lann, Polychrony for system design. *Journal for Circuits, Systems and Computers*. World Scientific, April 2003.

- [28] E.A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 17(12), 1217–1229, Dec. 1998.
- [29] D. Potop-Butucaru, B. Caillaud and A. Benveniste. Concurrency in Synchronous Systems. In Proc. of the 4th Int. Conf. on Applications of Concurrency in System Design (ACSD) Hamilton, Canada, June 2004