Compositional design of isochronous systems

Jean-Pierre Talpin, Julien Ouy, Loïc Besnard, Paul Le Guernic

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Compositional design of isochronous systems

Jean-Pierre Talpin — Julien Ouy — Loïc Besnard — Paul Le Guernic

N° ????

June 2007

Thème COM

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

Compositional design of isochronous systems

Jean-Pierre Talpin, Julien Ouy, Loïc Besnard, Paul Le Guernic

Thème COM — Systèmes communicants
Projets Espresso

Rapport de recherche n° ???? — June 2007 — 21 pages

Abstract: The synchronous modeling paradigm provides strong execution correctness guarantees to embedded system design while making minimal environmental assumptions. In most related frameworks, global execution correctness is achieved by ensuring endochrony: the insensitivity of (logical) time in the system from (real) time in the environment. Interestingly, endochrony can be statically checked, making it fast to ensure design correctness. Unfortunately, endochrony is not preserved by composition, making it difficult to exploit with component-based design concepts in mind.

Compositionality can be achieved by weakening the objective of endochrony but at the cost of an exhaustive state-space exploration. These observations raise a tradeoff between performance and precision. Our aim is to balance this tradeoff by proposing a formal design methodology that adheres to a weakened global design objective, namely, the non-blocking composition of weakly endochronous processes, while preserving local endochrony objectives. This yields an ad-hoc yet cost-efficient approach to compositional synchronous modeling.

Key-words: formal methods, embedded systems, program analysis, synchronous paradigm

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Mise en œuvre compositionelle de systèmes isochrones

Résumé : Le paradigme synchrone met en œuvre des méthodes formelles permettant de garantir la correction d'un programme en faisant peu d'hypothèse sur son environnement (d'exécution). Cette correction est assurée par la propriété d'endochronie: l'insensibilité du temps logique, dans le programme, au temps r 'eel, dans l'environnement. L'endochronie peut être vérifiée par analyse statique. elle donne donc facilement et rapidement l'assurance qu'un programme est correct. Malheureusement, elle n'est pas stable par composition, cela complique donc son utilisation dans le cas d'une conception modulaire de programmes.

La compositionalié peut cependant être obtenue en affaiblissant la propriété d'endochronie, mais au prix d'une mise en œuvre plus coûteuse car nécéssitant l'exploration de l'espace détats du programme. Cela nous confronte à un dilème entre performance et précision, entre simplicité et compositionalité. Notre objectif est d'aller au delà de cet équilibre en proposant une méthodologie de conception fondée sur l'objectif global de composer des modules non-blocants et sur l'objectif local d'assurer l'endochronie (de chaque module). La propriété obtenue est compositionelle et le coût de sa vérification est faible.

Mots-clés : méthodes formelles, systèmes embarqués, analyse de programmes, paradigme synchrone

1 Introduction

Embedded system design using synchronous modeling paradigms provides strong execution correctness guarantees while requiring minimal assumptions on the execution environment. In most synchronous formalisms, this is achieved by locally verifying that computation (in the system) is insensitive to communication delays (from the environment), i.e., that the system is endochronous (literally, time is defined from inside the system).

$$x : (t_1, 1)\ (t_2, 0)\ (t_3, 0)\ (t_4, 1) \quad \rightarrow \boxed{y = \mathsf{filter}(x)} \rightarrow y : \qquad (t_2, 1) \qquad (t_4, 1)$$

Figure 1: an endochronous filtering process receives an input signal x and emits a y every time the value of x. Output tags are timely related to input tags.

In the data-flow formalism Signal, for instance, design is driven by the safety objective of endochrony: endochrony guarantees a synchronization of computations and communications that is independent of possible network latency.

Unfortunately, endochrony is not a compositional property: it is not preserved by synchronous composition.

$$\begin{array}{llll} c : (t_0, 0) & (t_2, 1) & (t_4, 1) & (t_7, 0) \\ y : & (t_2, 1) & (t_4, 1) & \\ z : (t_0, 1) & & & (t_7, 0) \end{array} \rightarrow \boxed{d = \mathsf{merge}(c, y, z)} \rightarrow d : (t_0, 1) \quad (t_2, 1) \quad (t_4, 1) \quad (t_7, 0)$$

Figure 2: the synchronous composition of the filter with an endochronous merging process is no longer endochronous: time tags of the output d and no longer depend from one of the input signals x or c.

It is shown, in [17], that compositionality can be achieved by weakening the objective of endochrony: a weakly endochronous system is a deterministic system that can perform independent communications in any order as long as this does not alter its state (i.e. it satisfies the diamond property). In [16], it is further shown that the non-blocking composition of weakly endochronous processes is isochronous.

$$\begin{array}{l} x : 1\ 0\ 0\ 1 \\ c : 0\ 1\ 1\ 0 \\ z : 1\ 0\ 1\ 0 \end{array} \rightarrow \boxed{x = \mathsf{filter}(y) \parallel d = \mathsf{merge}(c, y, z)} \rightarrow \quad d : 1\ 1\ 1\ 0$$

Figure 3: the composition of the filter and the sample is isochronous: the tag-less asynchronous behavior is the same as the tagged synchronous behavior

However, checking that a system is weakly endochronous requires an exhaustive exploration of its state-space to guarantee that its behavior is independent from the order of inbound communications. This raises a tradeoff between performance (incurred by state-space exploration) and flexibility (gained from compositionality). We aim at balancing this trade-off by proposing a formal design methodology that weakens the global design

objective (non-blocking composition) and preserves design objectives secured locally (by accepting endochronous components).

Our approach consists in maintaining a compositional global design objective (non-blocking composition) while preserving properties secured locally (endochrony). This yields a less general yet cost-efficient approach to compositional modeling that is able to embrace most of the practical engineering situations and practices. This approach is particularly aimed at efficiently reusing most of the existing program analysis and compilation algorithms of Signal. To support the present design methodology, we have designed a simple controller synthesis and code generation scheme in Signal [15].

Paper outline The article starts in Section 2 with an introduction to Signal and its polychronous model of computation. Section 3 defines the necessary analysis framework. Our contribution is given in Section 4. We review the related work in Section 5 and conclude.

2 An introduction to Polychrony

In Signal, a process (written P or Q) consists of the synchronous composition (noted $P \,|\, Q$) of equations on signals (written $x = y\ f\ z$). A signal x represents an infinite flow of values. It is sampled according to the discrete pace of its clock, noted $\hat{x}$. An equation $x = y\ f\ z$ defines the output signal x by the relation of its input signals y and z through the operator f. A process defines the simultaneous solution of the equations it is composed of.

$$P, Q \ ::= \ x = y\ f\ z \mid P \,|\, Q \mid P/x \qquad \text{(process)}$$

There are several kinds of equations. A functional equation $x = y\ f\ z$ defines an (arithmetic or boolean) relation f between the operands y, z and the result x. A delay $x = y \,\mathsf{pre}\, v$ defines x by the value that the signal y had last time it was evaluated. Initially, x is defined by the value v. In a delay equation, the signals x and y are required to be synchronous: simultaneously present or simultaneously absent at all times.

A sampling $x = y \,\mathsf{when}\, z$ defines x by y when z is true and both y and z are present. In a sampling equation, the output signal x is present iff both input signals y and z are present and z holds the value $true$. A merge $x = y \,\mathsf{default}\, z$ defines x by y when y is present and by z otherwise. In a merge equation, the output signal is present iff either of the input signals y or z is present.

The process P/x restricts the lexical scope of the signal x to the process P. In the remainder, we write $\mathcal{V}(P)$ for the set of free signal names x of P (all x that occur in an equation of P and whose scope is not bound by restriction P/x). A free signal is an output iff it occurs on the left hand-side of an equation. Otherwise, it is an input signal.

Example Our first example is a filtering function. It receives a boolean input signal y. It produces an output signal x every time the value of the input changes. The local signal

y pre true holds the previous value of y, initially defined by true. If y and its previous value y pre true differ then x is given the value true. Otherwise x is absent.

$$x = \mathsf{filter}(y) \stackrel{\text{def}}{=} (x = \mathsf{true\ when}\,(y \neq (y\,\mathsf{pre\ true}\,)))$$

2.1 Model of computation

We describe the semantics of Signal in the polychronous model of computation (see [9] for more detail). In this model, symbolic tags t or u denote periods in time during which execution takes place. Time is defined by a partial order relation $\leq$ on tags: $t \leq u$ stipulates that t occurs before u. A chain is a totally ordered set of tags. It corresponds to the clock of a signal: it samples its values over a series of totally related tags. The domains for events, signals, behaviors and processes are defined as follows:

- an *event* is a pair consisting of a tag $t \in \mathbb{T}$ and a value $v \in \mathbb{V}$,
- a *signal* is a function from a *chain* of tags to a set of values,
- a *behavior* b is a function from a set of signal names to signals,
- a *process* p is a set of behaviors that have the same domain.

Example To illustrate these definitions, consider a possible behavior of the signals x and y defined by the process $x = \mathsf{filter}(y)$. The time tags at which the output x is present correspond to the time tags at which the previous and present value of y differ. Hence, evenly tagged values belong both to x and y.

$y \mapsto$	$(t_1,$ true$)$	$(t_2,$ false$)$	$(t_3,$ false$)$	$(t_4,$ true$)$	$(t_5,$ true$)$	$(t_6,$ false$)$
$x \mapsto$		$(t_2,$ true$)$		$(t_4,$ true$)$		$(t_6,$ true$)$

Notations We write $\mathcal{T}(s)$ for the chain of tags of a signal s and $\min s$ and $\max s$ for its minimal and maximal tag. We write $\mathcal{V}(b)$ for the domain of a behavior b (a set of signal names). The restriction of a behavior b to X is noted $b|_X$ (i.e. $\mathcal{V}(b|_X) = X$). Its complementary $b_{/X}$ satisfies $b = b|_X \uplus b_{/X}$ (i.e. $\mathcal{V}(b_{/X}) = \mathcal{V}(b) \setminus X$). We overload the use of $\mathcal{T}$ and $\mathcal{V}$ to talk about the tags of a behavior b and the set of signal names of a process p.

Synchrony The synchronization of a behavior b with a behavior c is noted $b \leq c$ and is defined as the effect of "stretching" its timing structure. A behavior c is a *stretching* of a behavior b, written $b \leq c$, iff $\mathcal{V}(b) = \mathcal{V}(c)$ and there exists a bijection f on tags s.t.

$\forall t, u, t \leq f(t) \wedge (t < u \Leftrightarrow f(t) < f(u))$

$\forall x \in \mathcal{V}(b), \mathcal{T}(c(x)) = f(\mathcal{T}(b(x))) \wedge \forall t \in \mathcal{T}(b(x)), b(x)(t) = c(x)(f(t))$

b and c are *clock-equivalent*, written $b \sim c$, iff there exists a behavior d s.t. $d \leq b$ and $d \leq c$. The synchronous composition $p \,|\, q$ of two processes p and q is defined by combining behaviors $b \in p$ and $c \in q$ that are identical on $I = \mathcal{V}(p) \cap \mathcal{V}(q)$, the interface between p and q.

$$p \,|\, q = \{b \cup c \,|\, (b, c) \in p \times q \ \wedge\ b|_I = c|_I \ \wedge\ I = \mathcal{V}(p) \cap \mathcal{V}(q)\}$$

Asynchrony Desynchronization is defined as the effect of "relaxing" the timing structure of a behavior: a behavior c is a *relaxation* of b, written $b \sqsubseteq c$, iff $\mathcal{V}(b) = \mathcal{V}(c)$ and, for all $x \in \mathcal{V}(b)$, $b|_x \leq c|_x$. Two behaviors b and c are *flow-equivalent*, written $b \approx c$, iff there exists a behavior d s.t. $b \sqsupseteq d \sqsubseteq c$. The asynchronous composition $p \parallel q$ of two processes p and q is defined by the set of behaviors d that are flow-equivalent to behaviors $b \in p$ and $c \in q$ along the interface $I = \mathcal{V}(p) \cap \mathcal{V}(q)$.

$$p \parallel q = \left\{ d \,|\, (b,c) \in p \times q \wedge b_{/I} \cup c_{/I} \leq d_{/I} \wedge b|_I \sqsubseteq d|_I \sqsupseteq c|_I \wedge I = \mathcal{V}(p) \cap \mathcal{V}(q) \right\}$$

2.2 Semantics of Signal

The semantics $[\![P]\!]$ of a Signal process P is a set of behaviors that are inductively defined by the concatenation of reactions. A reaction r is a behavior with (at most) one time tag t. We write $\mathcal{T}(r)$ for the tag of a non empty reaction r. An empty reaction of the signals X is noted $\emptyset|_X$. The empty signal is noted $\emptyset$. A reaction r is concatenable to a behavior b iff $\mathcal{V}(b) = \mathcal{V}(r)$, and, for all $x \in \mathcal{V}(b)$, $\max(b(x)) < \mathcal{T}(r(x))$. If so, concatenating r to b is defined by

$$\forall x \in \mathcal{V}(b), \forall u \in \mathcal{T}(b) \cup \mathcal{T}(r), (b \cdot r)(x)(u) = \text{if } u \in \mathcal{T}(r(x)) \text{ then } r(x)(u) \text{ else } b(x)(u)$$

Initially, we assume that $\emptyset|_{\mathcal{V}(p)} \in [\![P]\!]$. The semantics of a delay $x = y \,\mathsf{pre}\, v$ is defined by appending a reaction r of tag t to a behavior b. It initially defines x by the value v (when b is empty) and then by the previous value of y (i.e. $b(y)(u)$ where u is the maximal tag of b).

$$[\![x = y \,\mathsf{pre}\, v]\!] = \left\{ b \cdot r \left| \begin{array}{l} b \in [\![x = y \,\mathsf{when}\, z]\!], \\ u = \max(\mathcal{T}(b(y))),\ r(x) = \\ t = \mathcal{T}(r), \end{array} \right| \begin{array}{ll} t \mapsto b(y)(u), & r(y) \neq \emptyset \wedge b \neq \emptyset_{xy} \\ t \mapsto v, & r(y) \neq \emptyset \wedge b = \emptyset_{xy} \\ \emptyset, & r(y) = \emptyset \wedge b = \emptyset_{xy} \end{array} \right\}$$

Similarly, the semantics of a sampling $x = y \,\mathsf{when}\, z$ defines x by y when z is true.

$$[\![x = y \,\mathsf{when}\, z]\!] = \left\{ b \cdot r \left| \begin{array}{l} b \in [\![x = y \,\mathsf{when}\, z]\!], \\ u = \max(\mathcal{T}(b(y))),\ r(x) = \\ t = \mathcal{T}(r), \end{array} \right| \begin{array}{ll} r(y), & r(z)(t) = \mathsf{true} \\ \emptyset, & r(z)(t) = \mathsf{false} \\ \emptyset, & r(z) = \emptyset \end{array} \right\}$$

Finally, $x = y \,\mathsf{default}\, z$ defines x by y when y is present and by z otherwise.

$$[\![x = y \,\mathsf{default}\, z]\!] = \left\{ b \cdot r \,\middle|\, b \in [\![x = y \,\mathsf{default}\, z]\!], r(x) = \left| \begin{array}{ll} r(y), & r(y) \neq \emptyset \\ r(z), & r(y) = \emptyset \end{array} \right. \right\}$$

The meaning of the synchronous composition $P \,|\, Q$ is the synchronous composition $[\![P \,|\, Q]\!] = [\![P]\!] \,|\, [\![Q]\!]$ of the meaning of P and Q. The meaning of restriction is defined by $[\![P/x]\!] = \{c \,|\, b \in [\![P]\!] \wedge c \leq (b_{/x})\}$.

Example The meaning of the equation $x = \mathsf{true\ when}\,(y \neq (y\,\mathsf{pre\ true}))$ consists of a set of behaviors with two signals x and y. On line one, below, we choose a behavior for the input signal y of the equation. On line two, we define the signal for the expression $y\,\mathsf{pre\ true}$ by application of the function $[\![]\!]$. Notice that y and $y\,\mathsf{pre\ true}$ are synchronous (they have the same set of tags). On line three, the output signal x is defined at the time tags t_i when y and $y\,\mathsf{pre\ true}$ hold different values, as expected in the previous example.

$$\begin{array}{rccccccc} y \mapsto & (t_1,\,\mathsf{true}) & (t_2,\,\mathsf{false}) & (t_3,\,\mathsf{false}) & (t_4,\,\mathsf{true}) & (t_5,\,\mathsf{true}) & (t_6,\,\mathsf{false}) \\ y\,\mathsf{pre\ true} \mapsto & (t_1,\,\mathsf{true}) & (t_2,\,\mathsf{true}) & (t_3,\,\mathsf{false}) & (t_4,\,\mathsf{false}) & (t_5,\,\mathsf{true}) & (t_6,\,\mathsf{true}) \\ x \mapsto & & (t_2,\,\mathsf{true}) & & (t_4,\,\mathsf{true}) & & (t_6,\,\mathsf{true}) \end{array}$$

2.3 Formal properties

The formal properties considered in the remainder pertain the insensitivity of timing relations in a process p (its local clock relations) to external communication delays. The property of endochrony, Definition 1, guarantees that the synchronization performed by a process p is independent from latency in the network. Formally, let I be a set of input signals of p, whenever the process p admits two input behaviors $b|_I$ and $c|_I$ that are assumed to be flow equivalent (timing relations have been altered by the network) then p always reconstructs the same, clock-equivalent, timing relations in b and c.

Definition 1 *A process p is* endochronous *iff* $\exists I \subset \mathcal{V}(p), \forall b, c \in p,\ b|_I \approx c|_I \Rightarrow b \sim c$.

Example To prove that the filter is an endochronous process, first consider two possible behaviors, b and c. Next, choose flow-equivalent input signals $b(y) = (t_1, \mathsf{true})(t_2, \mathsf{false})(t_3, \mathsf{false})(t_4, \mathsf{true})$ and $c(y) = (u_1, \mathsf{true})(u_2, \mathsf{false})(u_3, \mathsf{false})(u_4, \mathsf{true})$: they share no tags, but carry the same values in the same order. Given these input signals, the filter necessarily constructs the output signals $b(x) = (t_2, \mathsf{true})(t_4, \mathsf{true})$ and $c(x) = (u_2, \mathsf{true})(u_4, \mathsf{true})$. One notice that b and c are then equivalent by the bijection $(t_i \mapsto u_i)_{0<i<5}$: they are clock-equivalent. Hence, the filter is endochronous.

The weaker definition of endochrony requires a definition for the union, written $r \sqcup s$, of two reactions r and s that have the same tag t and disjoint domains (independent reactions).

$$\forall x \in \mathcal{V}(r), (r \sqcup s)(x) = \text{if } r(x) \neq \emptyset \text{ then } r(x) \text{ else } s(x)$$

Definition 2, below, formulates weak endochrony in the polychronous model of computation. Informally, process p is weakly endochronous iff it is deterministic and can perform independent reactions r and s in any order. Note that, by Definition 1, endochrony implies weak-endochrony (e.g. the filter is weakly endochronous).

Definition 2 *A process p is* weakly-endochronous *iff*

1. p is deterministic: $\exists I \subset \mathcal{V}(p), \forall b, c \in p,\ b|_I = c|_I \Rightarrow b = c$.

2. p satisfies the diamond property for all independent reactions r and s:

(a) if $b \cdot r \cdot s \in p$ *then* $b \cdot s \in p$
(b) if $b \cdot r \in p$ *and* $b \cdot s \in p$ *then* $b \cdot (r \sqcup s) \in p$
(c) if $b \cdot (r \sqcup s) \in p$ *and* $b \cdot (r \sqcup t) \in p$ *then* $b \cdot r \cdot s \in p$ *and* $b \cdot r \cdot t \in p$

Last, [17] proves that two weakly endochronous processes p and q are *isochronous* iff they are non-blobking (a locally synchronous reaction initiated by p or q yields a globally asynchronous execution of $p \parallel q$).

Definition 3 *Two processes* p *and* q *are* isochronous *iff* $p \,|\, q \approx p \parallel q$

A process p is non-blocking iff, in any reachable state (characterized by a behavior b), it has a path to a stuttering state (characterized by a reaction r).

Definition 4 *A process* p *is* non-blocking *iff* $\forall b \in p, \exists r, b \cdot r \in p$

3 Formal analysis

For the purpose of program analysis and program transformation, the control-flow tree and the data-flow graph of multi-clocked Signal specifications are constructed. These data structures manipulate clocks and signal names.

3.1 Clock and scheduling relations

The clock c of a signal denotes a series of instants (a chain of time tags). The clock $\hat{x}$ of a signal x denotes the instants at which the signal x is present. The clock $[x]$ (resp. $[\neg x]$) denotes the instants at which x is present and holds the value true (resp. false). A clock expression e is either the empty clock, noted 0, a signal clock c, or the conjunction $e_1 \wedge e_2$, the disjunction $e_1 \vee e_2$, the symmetric difference $e_1 \setminus e_2$ of two clock expressions e_1 and e_2.

$$c ::= \hat{x} \,|\, [x] \,|\, [\neg x] \quad \text{(clock)} \qquad e ::= 0 \,|\, c \,|\, e_1 \wedge e_2 \,|\, e_1 \vee e_2 \,|\, e_1 \setminus e_2 \quad \text{(expression)}$$

Signals and clocks are related by synchronization and scheduling relations, noted R. A scheduling relation $a \rightarrow^c b$ specifies that the calculation of the node b, a signal or a clock, cannot be scheduled before that of the node a when the clock c is present. A clock relation $c = e$ specifies that the signal clock c is present iff the clock expression e is true. Just as ordinary processes P, relations R are subject to composition $R \,|\, S$ and to restriction R/x. The semantics of synchronization and scheduling relations in the polychronous model of computation, introduced in [9], is recalled in appendix.

$$a, b ::= x \,|\, \hat{x} \quad \text{(node)} \qquad R, S ::= c = e \,|\, a \rightarrow^c b \,|\, (R \,|\, S) \,|\, R/x \quad \text{(timing relations)}$$

3.2 Clock inference

The inference system $P : R$ associates a process P with its implicit timing relations R. It is defined by induction on the structure of P: the deduction for composition $P \,|\, Q$ and for P/x are defined by induction on the deductions $P : R$ and $Q : S$ for P and for Q.

$$\frac{P : R \quad Q : S}{P \,|\, Q : R \,|\, S} \qquad \frac{P : R}{P/x : R/x}$$

Deduction starts from the assignment of clock relations to primitive equations. In a delay equation $x = y \,\mathsf{pre}\, v$, the input and output signals are synchronous, written $\hat{x} = \hat{y}$, and do not have any scheduling relation. In a sampling equation $x = y \,\mathsf{when}\, z$, the clock of the output signal x is defined by that of $\hat{y}$ and sampled by $[z]$. The input y is scheduled before the output when both $\hat{y}$ and $[z]$ are present, written $y \rightarrow^{\hat{x}} x$. In a merge equation $x = y \,\mathsf{default}\, z$ the output signal x is present if either of the input signals y or z are present. The first input signal y is scheduled before x when it is present, written $y \rightarrow^{\hat{y}} x$. Otherwise z is scheduled before x, written $z \rightarrow^{\hat{z}\setminus\hat{y}} x$. A functional equation $x = y\, f\, z$ synchronizes and serializes its input and output signals.

$$x = y \,\mathsf{pre}\, v : (\hat{x} = \hat{y}) \qquad x = y \,\mathsf{default}\, z : (\hat{x} = \hat{y} \vee \hat{z} \;|\; y \rightarrow^{\hat{y}} x \,|\, z \rightarrow^{\hat{z}\setminus\hat{y}} x)$$
$$x = y \,\mathsf{when}\, z : (\hat{x} = \hat{y} \wedge [z] \;|\; y \rightarrow^{\hat{x}} x) \qquad x = y\, f\, z : (\hat{x} = \hat{y} = \hat{z} \;|\; y \rightarrow^{\hat{x}} x \,|\, z \rightarrow^{\hat{x}} x)$$

We write $R \models S$ to mean that R satisfies S in the Boolean algebra in which timing relations are expressed: composition $R \,|\, S$ stands for conjunction and restriction R/x for existential quantification (some examples are given below). For all boolean signal x in $\mathcal{V}(R)$, we assume that $R \models \hat{x} = [x] \vee [\neg x]$ and $R \models [x] \wedge [\neg x] = 0$.

Example To outline the use of clock and scheduling relation analysis in Signal, we consider the specification and analysis of a one-place buffer. Process buffer implements two functionalities: alternate and current. The process alternate synchronizes the signals x and y to the true and false values of an alternating boolean signal t. The process current stores the value of an input signal y and loads it into the output signal x upon request.

$$x = \mathsf{buffer}(y) \stackrel{\text{def}}{=} (x = \mathsf{current}(y) \,|\, \mathsf{alternate}(x, y))$$
$$\mathsf{alternate}(x, y) \stackrel{\text{def}}{=} (s = t \,\mathsf{pre}\, \mathsf{true} \,|\, t = \mathsf{not}\, s \,|\, \hat{x} = [t] \,|\, \hat{y} = [\neg t])\, /s\, t$$
$$x = \mathsf{current}(y) \stackrel{\text{def}}{=} (r = y \,\mathsf{default}\, (r \,\mathsf{pre}\, \mathsf{false}) \,|\, x = r \,\mathsf{when}\, \hat{x} \,|\, \hat{r} = \hat{x} \vee \hat{y})\, /r$$

The process buffer has three clock equivalence classes. The clocks $\hat{s}$, $\hat{t}$ and $\hat{r}$ are synchronous (i.e. $\hat{r} = \hat{s} = \hat{t} = \hat{x} \vee \hat{y}$). Two other synchronization classes, $\hat{x} = [t]$ and $\hat{y} = [\neg t]$, are samples of the signal t.

$$R_{\mathsf{buffer}} \stackrel{\text{def}}{=} \big((s \rightarrow^{\hat{s}} t \,|\, \hat{x} = [t] \,|\, \hat{y} = [\neg t]) \;|\; (y \rightarrow^{\hat{y}} r \,|\, r \rightarrow^{\hat{x}} x \,|\, \hat{r} = \hat{x} \vee \hat{y})\big)\, /r s t$$

From R, we can deduce $R_{\mathsf{buffer}} \models (\hat{r} = \hat{t})$. Since t is a Boolean signal, $R_{\mathsf{buffer}} \models \hat{t} = [t] \vee [\neg t]$ (a signal is always true or false when present). By definition of R, $R_{\mathsf{buffer}} \models \hat{x} = [t]$ and $R_{\mathsf{buffer}} \models \hat{y} = [\neg t]$ (x and y are sampled from t). Hence, we have $R_{\mathsf{buffer}} \models (\hat{r} = \hat{x} \vee \hat{y})$ and can deduce that $R_{\mathsf{buffer}} \models (\hat{r} = \hat{t})$.

3.3 Clock hierarchy and scheduling graph

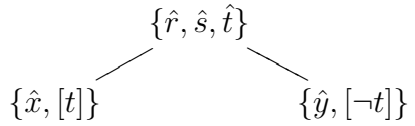
The internal data-structures manipulated by the Signal compiler for program analysis and code generation consist of a clock hierarchy and of a scheduling graph. The clock hierarchy represents the control-flow of a process by a partial order relation. The scheduling graph defines a fine-grained scheduling of otherwise synchronous signals.

Definition 5 *The* hierarchy $\preceq$ *of a process* $P : R$ *is the transitive closure of the maximal relation defined by the following axioms and rules:*

1. *for all boolean signals* x, $\hat{x} \preceq [x]$ *and* $\hat{x} \preceq [\neg x]$
2. *if* $R \models b = c$ *then* $b \preceq c$ *and* $c \preceq b$, *written* $b \sim c$
3. *if* $R \models b_1 = c_1 \, f \, c_2$, $f \in \{\wedge, \vee, \setminus\}$, $b_2 \preceq c_1$, $b_2 \preceq c_2$ *and* b_2 *is minimal*[1] *then* $b_2 \preceq b_1$

We refer to $c_\sim$ *as the clock equivalence class of* c *in the hierarchy* $\preceq$

Example The hierarchy of the buffer is constructed by application of the first and second rules of Definition 5. Rule 2 defines three clock equivalence classes $\{\hat{r}, \hat{s}, \hat{t}\}$, $\{\hat{x}, [t]\}$ and $\{\hat{y}, [\neg t]\}$. Rule 1 places the first class above the two others. This yields the following structure



A well-formed hierarchy is such that no relation $b \preceq c$ contradicts Definition 5. For instance, the hierarchy of the process $x = y \text{ and } z \parallel z = y \text{ when } y$ is ill-formed, since $\hat{y} \sim [y]$. A process with an ill-formed hierarchy may block.

Definition 6 *A hierarchy* $\preceq$ *is* ill-formed *iff* either $\hat{x} \succeq [x]$ *or* $\hat{x} \succeq [\neg x]$, *for any* x, *or* $b_1 \preceq b_2$ *for any* $b_1 = c_1 \, f \, c_2$ *such that* $c_1 \succeq b_2 \preceq c_2$ *and* $b_2 \preceq b_1$

The elimination of symmetric difference relations from R (that are temporarily represented by negative clocks $\overline{c}$) is obtained by repeated application of the following rules:

$$\begin{array}{llllll} c \setminus d & \stackrel{\text{def}}{=} c \wedge \overline{d} & \overline{c \vee d} & \stackrel{\text{def}}{=} \overline{c} \wedge \overline{d} & \overline{[\neg x]} \stackrel{\text{def}}{=} \hat{x} \vee [x] & c \wedge (\hat{x} \vee \overline{\hat{x}}) \stackrel{\text{def}}{=} c \\ \overline{c \wedge d} & \stackrel{\text{def}}{=} \overline{c} \vee \overline{d} & \overline{[x]} & \stackrel{\text{def}}{=} \overline{\hat{x}} \vee [\neg x] & \hat{x} \wedge \overline{\hat{x}} \stackrel{\text{def}}{=} 0 & \text{if } d \sim \hat{x}, \overline{\hat{x}} \stackrel{\text{def}}{=} \overline{d} \end{array}$$

In [2], it is shown that a symmetric difference $c \setminus d$ has a disjunctive form if c and d have a common minimum b (i.e. $c \succeq b \preceq d$). The relations R are said in disjunctive form iff it has no clock defined by a symmetric difference relation. For instance, suppose that $d \sim [x]$ and that $c \succeq b \preceq d$. Then, the clock expression $c \setminus d$ can be eliminated because it can be expressed with $c \wedge [\neg x]$.

[1] i.e. for any b such that $b \preceq c_1$ and $b \preceq c_2$, $b_2 \preceq b$

Definition 7 *A process P of timing relations R and hierarchy $\preceq$ is* well-clocked *iff $\preceq$ is well-formed and R is disjunctive.*

The transitive closure of scheduling relations in R is the largest relation $\twoheadrightarrow$ defined by, 1) if $R \models a \rightarrow^c b$ then $a \twoheadrightarrow^c b$, 2) if $a \twoheadrightarrow^{e_1} b$ and $a \twoheadrightarrow^{e_2} b$ then $a \twoheadrightarrow^{e_1 \vee e_2} b$, and 3) if $a \twoheadrightarrow^{e_1} c$ and $c \twoheadrightarrow^{e_2} b$ then $a \twoheadrightarrow^{e_1 \wedge e_2} b$

Definition 8 *A process P of timing relations R is* acyclic *iff the transitive closure $\twoheadrightarrow$ of its scheduling relations R satisfy, for all nodes a, if $a \twoheadrightarrow^e a$ then $R \models e = 0$.*

4 Contribution

We formulate a decision procedure that uses the clock hierarchy and the scheduling graph of a Signal process to compositionally check the property of isochrony.

4.1 Compilability

We start by considering the class of Signal processes P that are reactive and deterministic.

Definition 9 *A process P is* compilable *iff it is acyclic and its relations R are well-clocked.*

Property 1 *A compilable process P is reactive and deterministic.*

Proof An immediate consequence of Property 5, in [18], where a well-clocked and acyclic process is proved to be deterministic.

4.2 Roots of a hierarchy

Next, we consider the structure of a compilable Signal specification. It is possibly paced by several, independent, input signals. It necessarily corresponds to a hierarchy $\preceq$ that has several roots. To represent them, we refer to $\preceq^\circ$ as the minimal clock equivalence classes of $\preceq$, and to $\preceq^c$ as the tree of root c in the hierarchy $\preceq$.

$$\preceq^\circ = \{c_\sim \mid c \in \min \preceq\} \qquad \preceq^c = \{(c,d)\} \cup \preceq^d \mid c \preceq d$$

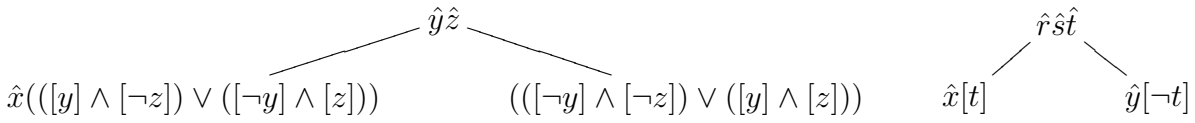
When the clock hierarchy of a process has a unique root, it is endochronous, since the presence of any clock is solely determined by the presence and values of clocks above it in the hierarchy.

Definition 10 *A process p is* hierachic *iff its hierarchy has a unique root $\preceq^\circ$.*

Property 2 *A compilable and hierarchic process p is endochronous.*

Proof A detailed proof appears in [18].

Example For instance, the hierarchies of the filter $x = \mathsf{filter}(y) \stackrel{\text{def}}{=} (x = \mathsf{true\ when}\,(y \neq z) \mid z = y\,\mathsf{pre\ true}\,)))/z$, left, and of the buffer, right, are both hierarchic: they are endochronous.



By contrast, a process with several roots necessarily defines concurrent threads of execution. Indeed, and by definition of a hierarchy, its roots cannot be expressed or calculated one in terms of the others. As a consequence, they can neither be synchronized nor sampled one with respect to the others. Therefore, they naturally define the source of concurrency which is the subject of verification for the satisfaction of weak endochrony.

4.3 Model checking weak endochrony

Checking that a compilable process p is weakly endochronous reduces to proving that the roots of a process hierarchy satisfy property (2a) of definition 2 by using bounded model checking. Property (2a) can be formulated as an invariant in Signal and submitted to its model checker Sigali [10].

(1) $i = \mathsf{StateIndependent}\,(x, y) \stackrel{\text{def}}{=}$

$$\left(\begin{array}{r} [cx_{t+1}] = \hat{x} \,\|\, cx_t \;=\; cx_{t+1}\,\mathsf{pre\ false} \\ \mid [cy_{t+1}] = \hat{y} \,\|\, cy_t \;=\; cy_{t+1}\,\mathsf{pre\ false} \\ \mid \qquad i = (\,\mathsf{not}\,cx_t\,\mathsf{or}\,cy_t)\,\mathsf{or}\,(cx_{t+1}\,\mathsf{or\ not}\,cy_{t+1})\,\mathsf{or}\,(cx_t\,\mathsf{and}\,cy_t) \end{array}\right) \Big/ \begin{array}{l} cx_t, cx_{t+1} \\ cy_t, cy_{t+1} \end{array}$$

The invariant returned by $\mathsf{StateIndependent}\,(x, y)$ is defined for all pairs of root clock equivalence classes. It says that, if x is present and y absent at time t (i.e. $cx_t \wedge \neg cy_t$) and if y is present and x absent at time $t+1$ (i.e. $\neg cx_{t+1} \wedge cy_{t+1}$) then x and y can both be present at time t (i.e. $cx_t \wedge cy_t$), written $(\neg cx_t \vee cy_t) \vee (cx_{t+1} \vee \neg cy_{t+1}) \vee (cx_t \wedge cy_t)$.

Properties (2b-2c) can similarly be checked with the properties $\mathsf{OrderIndependent}$ and $\mathsf{FlowIndependent}$. Property $\mathsf{OrderIndependent}$ is defined by $(cx_t \wedge \neg cy_t) \wedge (cy_t \wedge \neg cx_t) \Rightarrow (cx_t \wedge cy_t)$. It means that x and y are independently available at all times.

(2) $i = \mathsf{OrderIndependent}\,(x, y) \stackrel{\text{def}}{=}$

$$\left(\;[cx_t] = \hat{x} \,|\, [cy_t] = \hat{y} \,|\, i = (\,\mathsf{not}\,cx_t\,\mathsf{or}\,cy_t)\,\mathsf{or}\,(cx_t\,\mathsf{or\ not}\,cy_t)\,\mathsf{or}\,(cx_t\,\mathsf{and}\,cy_t)\;\right)/cx_t, cy_t$$

Property FlowIndependent is defined for any signal $z \in \mathcal{V}(p)$ by $cz_t \wedge ((cx_t \wedge \neg cy_t) \wedge (cy_t \wedge \neg cx_t)) \Rightarrow cz_t \wedge ((cx_{t+1} \wedge \neg cy_{t+1}) \vee (cy_{t+1} \wedge \neg cx_{t+1}))$.

$$(3)\quad i = \mathsf{FlowIndependent}\,(x,y,z) \stackrel{\text{def}}{=} \left(\begin{array}{l} [cx_{t+1}] = \hat{x} \;\|\; cx_t = cx_{t+1}\ \mathsf{pre}\ \mathsf{false} \\ \mid [cy_{t+1}] = \hat{y} \;\|\; cy_t = cy_{t+1}\ \mathsf{pre}\ \mathsf{false} \\ \mid [cz_{t+1}] = \hat{z} \;\|\; cz_t = cz_{t+1}\ \mathsf{pre}\ \mathsf{false} \\ \mid i = (\ \mathsf{not}\ cz_t\ \mathsf{or}\ ((\ \mathsf{not}\ cx_t\ \mathsf{or}\ cy_t)\ \mathsf{or}\ (cx_{t+1}\ \mathsf{or}\ \mathsf{not}\ cy_{t+1}))) \\ \qquad \mathsf{or}\ (cz_t\ \mathsf{and}\ ((cx_{t+1}\ \mathsf{and}\ \mathsf{not}\ cy_{t+1})\ \mathsf{or}\ (\ \mathsf{not}\ cx_{t+1}\ \mathsf{and}\ cy_{t+1}))) \end{array}\right) \Big/ \begin{array}{l} cx_t, cx_{t+1} \\ cy_t, cy_{t+1} \\ cz_t, cz_{t+1} \end{array}$$

When the clock hierarchy of a compilable process P consists of multiple roots, we can use the above properties to verify that it is weakly endochronous.

Property 3 *A compilable process P whose roots satisfy criteria (1-3) is weakly endochronous.*

Proof We observe that the formulation of properties $(1-3)$ directly translate Definition 2 in terms of timed Boolean equations. Since they are expressed in Signal, one can model-check them against the specification of the process P under consideration to verify that it is weakly endochronous.

4.4 Static checking isochrony

Unfortunately, model-checking is unaffordable for purposes such as program transformation or code generation. In the aim of generating sequential or concurrent code starting from weakly endochronous specifications, we would like to define a simple and cost-efficient criterion to allow for a large and easily identifiable class of weakly endochronous programs to be statically checked and compiled. To this end, we define the following formal design methodology.

Definition 11 *If P is compilable and hierarchic then it is* weakly hierachic. *If P and Q are weakly hierarchic and $P \,|\, Q$ is well-clocked and acyclic then $P \,|\, Q$ is weakly hierarchic.*

A process P is weakly hierarchic iff it is compilable and its hierarchy has roots $r_{1..n}$ such that, for all $1 \leq i < n$, $X_i = \mathcal{V}(\preceq^{r_i})$, $P_i = P|_{X_i}$ is weakly hierarchic and the pair $(\prod_{j=1}^{i} P_j, P_{i+1})$ is well-clocked and acyclic.

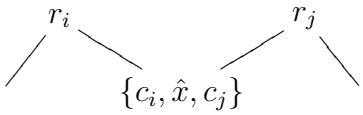
Property 4

1. *A weakly hierarchic process P is weakly endochronous.*
2. *If $P \,|\, Q$ is weakly hierarchic then $P \,|\, Q$ is non-blocking.*
3. *If $P \,|\, Q$ is weakly hierarchic then P and Q are isochronous.*

Proof

1. By definition, a weakly hierarchic process P consists in the composition of a series of processes P_i that are individually compilable and hierarchic, hence endochronous. Since endochrony implies weak endochrony, and since weak endochrony is preserved by composition, the composition P of the P_is is weakly endochronous.

2. Consider the hierarchy of any pair of endochronous processes P_i and P_j in $P \,|\, Q$ that share a common signal x of clock $\hat{x}$. The processes P_i and P_j have roots r_i and r_j and synchronize on $\hat{x}$ at a sub-clock c_i, computed using r_i (since P_i is hierarchic) and at a clock c_j, computed using r_j (since P_j is hierarchic).



   Since $P_i \,|\, P_j$ is well-clocked, the clocks c_i, c_j and hence $\hat{x}$ have a disjunctive form. Hence, it cannot be the case that $\hat{x}$ is defined by the symmetric difference of a clock under r_i and another (e.g. under r_j). Therefore, any reaction initiated in P_i to produce $\hat{x}$ can locally and deterministically decide to wait for a rendez-vous with a reaction of P_j consuming $\hat{x}$. Since P_i and P_j are well-formed, then it cannot be the case that $\hat{x} = 0$, which would mean that the rendez-vous would never happen. Finally, since $P_i \,|\, P_j$ is acyclic, the rendez-vous of c_i and c_j cannot deadlock. This holds for any pair of endochronous processes P_i and P_j in $P \,|\, Q$, hence $P \,|\, Q$ is non–blocking.

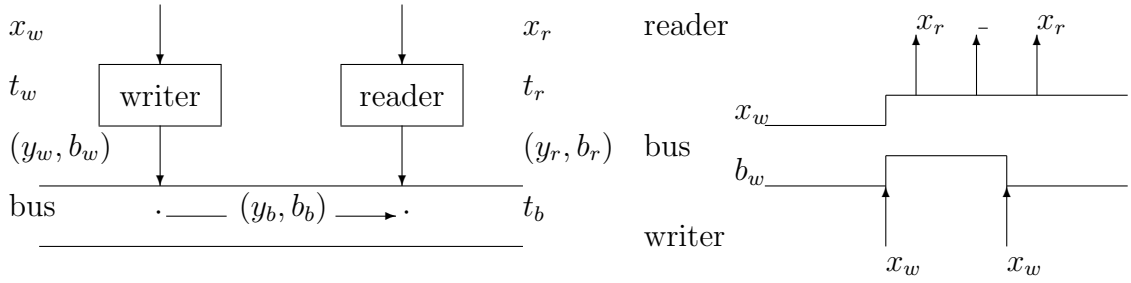
3. These conditions precisely correspond to the weak isochrony criterion of [17], namely, that non-blocking composition (2) of weakly endochronous processes (1) is isochronous. Consequently, the composition of P and Q is isochronous.

4.5 A design methodology for weakly hierarchic processes

The static criterion for checking the composition of endochronous processes weakly isochronous defines a cost-effective methodology for the integration of existing components in the aim of architecture exploration or simulation. Interestingly, this formal methodology meets most of the engineering practice and industrial usage of Signal: the real-time simulation of an embedded architecture (e.g. integrated modular avionics) starting from heterogeneous, possibly foreign, functional blocks (merely endochronous, data-flow functions) and architecture service models (e.g. the ARINC 653 real-time operating system [11]).

Example of a loosely time-triggered architecture To demonstrate this capability, we consider a realistic use case build upon examples we previously presented. We wish to design a simulation model for a loosely time-triggered architecture (LTTA). The LTTA is composed of three devices, a *writer,* a *bus,* and a *reader.* Each device is activated by its own clock.

At the nth clock tick (time $t_w(n)$), the *writer* generates the value $x_w(n)$ and an alternating flag $b_w(n)$. At any time $t_w(n)$, the writer's output buffer (y_w, b_w) contains the last value that was written into it. At $t_b(n)$, the *bus* bus fetches (y_w, b_w) to store in the input buffer of the reader, denoted by (y_b, b_b). At $t_r(n)$, the *reader* loads the input buffer (y_b, b_b) into the variables $y_r(n)$ and $b_r(n)$. Then, in a similar manner as for an alternating bit protocol, the reader extracts $y_r(n)$ iff $b_r(n)$ has changed.



A simulation model of the LTTA To model an LTT architecture in Signal, we consider two data-processing functions that communicate by writing and reading values on an LTT bus. In Signal, we model an interface of these functions that exposes their (limited) control. The writer accepts an input x_w and defines the boolean flag b_w that is carried along with it over the bus.

$$(y_w, b_w) = \mathsf{writer}(x_w, c_w) \stackrel{\text{def}}{=} \Big(\hat{x}_w = \hat{b}_w = [c_w] \,|\, y_w = x_w \,|\, b_w = \mathsf{not}\,(b_w \,\mathsf{pre}\, \mathsf{true}\,)\Big)$$

The reader loads its inputs y_r and b_r from the bus and filters x_r upon a switch of b_r.

$$x_r = \mathsf{reader}(y_r, b_r, c_r) \stackrel{\text{def}}{=} (x_r = y_r \,\mathsf{when}\, \mathsf{filter}(b_r) \,|\, \hat{y}_r = [c_r])$$

The bus buffers and forwards the inputs y_w and b_w to the reader. The clock c_b is not used since the buffers have local clocks.

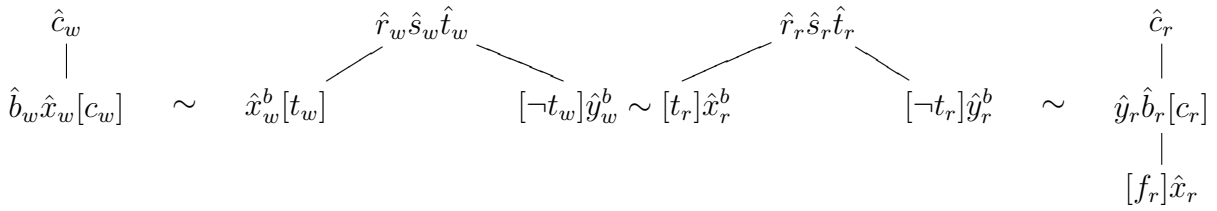
$$(y_r, b_r) = \mathsf{bus}(y_w, b_w, c_b) \stackrel{\text{def}}{=} ((y_r, b_r) = \mathsf{buffer}(\mathsf{buffer}(y_w, b_w)))$$

The process ltta is defined by its three components reader, bus and writer.

$$x_r = \mathsf{ltta}(x_w, c_w, c_b, c_r) \stackrel{\text{def}}{=} (x_r = \mathsf{reader}(\mathsf{bus}(\mathsf{writer}(x_w, c_w), c_b), c_r))$$

We observe that the hierarchy of the LTTA is composed of four trees. Each tree corresponds to an endochronous and separately compiled process, connected to the other at four rendezvous points (depicted by clock equivalence relations $\sim$). The LTT architecture itself is not endochronous, but it can be checked weakly isochronous by observing that its four components are endochronous and that the composition of these components is well-clocked

and acyclic.



A concurrent simulation code generation scheme In a related report [15], we describe a complementary separate compilation technique which accommodates the concurrent composition of endochronous processes by synthesizing rendez-vous protocols to interface communicating endochronous processes.

Together with the present verification technique and design methodology, it defines a new way to regard design using a synchronous multi-clocked model of computation by the component-based integration of endochronous functionalities and the modular exploration of the design space.

5 Related Work

In synchronous design formalisms, the design of an embedded architecture is achieved by constructing an endochronous model of the architecture and then by automatically synthesizing ad-hoc synchronization protocols between the elements of this model that will be physically distributed. This technique is called desynchronization and a thorough survey on it is proposed in [12]. In the case of Signal, automated distribution is proposed by Maffeis and Aubry in [14] and [1]. It consists in partitioning endochronous specifications and synthesizing inter-partition protocols to ensure preservation of endochrony.

In [13], Girault et al. propose a different approach for the synchronous languages Lustre and Esterel. It consists in replicating the generated code of an endochronous specification and applying an optimization technique. This technique consists in replacing replicated blocks of instructions by inter-partition communications. As it uses notions of bi-simulation to safely eliminate blocks, it leads to the construction of a distributed program that consists of endochronously connected programs. But again, distributed code generation is also driven by the global preservation of endochrony.

In [17], the so-called property of weak endochrony is proposed. Weak endochrony supports the compositional construction of globally asynchronous system by adhering to the global objective of weak-isochrony. In [19], we propose a related analysis of Signal programs to check this property. However, we observe that it is far more costly than necessary, at least for code generation purposes, as it requires an exhaustive state-space exploration. In [8], Dasgupta et al. also propose a technique to synthesize delay-insensitive protocols for synchronous circuits described with Pétri Nets.

In the model of latency-insensitive protocols [5], components are denoted by the notion of *pearl* ("intellectual property under a shell"). A pearl is required to satisfy an invariant of

patience (which, in turn, implies endochrony [18]) and a *latency-insensitive protocol* wraps the pearl with a generic client-side controller: a so-called relay station.

The relay station ensures the functional correctness of the pearl in the architecture by guaranteeing the preservation of signal flows (i.e. isochrony). It implements this function by suspending the pearl's incoming traffic as soon as it is reported to exceed its consumption capability. A technique proposed by Casu et al. in [7] refines this protocol to prevent unnecessary traffic suspension by controlling traffic through pre-determined periodic schedules.

The latency-insensitive protocol is a compositional approach, and can be seen as a "black-box" approach, in that no knowledge on the pearl (but its capability to be patient) is required. Just as desynchronization, Casu's variant is a "grey-box" approach, where knowledge on the pearl is needed to synthesize an an-hoc controller and, at the same time, ensure functional correctness.

The clock analysis at the core of our approach shares similarities with both approaches (desynchronization and latency insensitivity) but implements a different rendez-vous-based communication mechanism. It avoids the need for any suspension mechanism (apart from the implementation of rendez-vous) thanks to the determination of precise synchronization and scheduling relations.

6 Conclusions

We are concerned with a cost-effective technique for the compositional design of globally asynchronous architectures starting from synchronous modules. We propose a formal design methodology which balances the trade-off between cost (of verification) and compositionality (of design). Our approach maintains a compositional global design objective of isochrony and preserves properties secured locally (endochrony) by checking that their composition is non-blocking.

This yields a less general yet cost-efficient approach to compositional modeling embedded architectures. In addition, it meets the actual engineering practice and industrial usage of Signal. The commercial implementation of Signal, Sildex, commercialized by TNI, is widely used for the real-time simulation of embedded architectures starting from heterogeneous, possibly foreign, functional blocks (merely endochronous, data-flow functions) and architecture service models (e.g. the ARINC 653 real-time operating system [11]).

As an example, TNI has developed a real-time, hardware in-the-loop, simulator of all onboard electronic equipments for a car manufacturer. Our technique efficiently reuses most of existing compilation tool-suites available for Signal in order to implement our proposal, which justifies presenting it in sufficient details in the present article. We are currently upgrading the Polychrony toolset, that supports the Signal specification formalism, with a simple controller-synthesis and code generation scheme supporting the proposed methodology.

References

[1] Pascal Aubry. Mises en oeuvre distribuées de programmes synchrones. Thèse de l'Université de Rennes, 1997.

[2] Loïc Besnard. Compilation de Signal: horloges, dépendances, environnements. Thèse de l'Université de Rennes, 1992.

[3] Albert Benveniste, Benoit Caillaud, and Paul Le Guernic. Compositionality in dataflow synchronous languages: Specification and distributed code generation. *Information and Computation*, v. 163. Academic Press, 2000.

[4] Albert Benveniste, Paul Caspi, Stephen Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone. The Synchronous Languages Twelve Years Later. *Proceedings of the IEEE*, 2003.

[5] Luca Carloni, Ken McMillan, and Alberto Sangiovanni-Vincentelli. The theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 20(9). IEEE Press, 2001.

[6] Paul Caspi, Alain Girault, and Daniel Pilaud. Distributing Reactive Systems. International Conference on Parallel and Distributed Computing Systems. ISCA, 1994.

[7] Mario Casu, Luca Macchiarulo. A new approach to latency insensitive design. Design Automation Conference. ACM Press, 2004.

[8] Sohini Dasgupta, Dumitru Potop-Butucaru, Benoît Caillaud, Alex Yakovlev. Moving from Weakly Endochronous Systems to Delay-Insensitive Circuits. Formal Methods for GALS Design, Electronic Notes in Theoretica Computer Science. Elsevier, 2006.

[9] Paul Le Guernic, Jean-Pierre Talpin, and Jean-Christophe Le Lann. Polychrony for system design. *Journal of Circuits, Systems and Computers*. World Scientific, 2003.

[10] Hervé Marchand, Eric Rutten, Michel Le Borgne and M. Samaan. Formal Verification of programs specified with Signal : application to a power transformer station controller. Science of Computer Programming, v. 41(1), 2001.

[11] Abdoulaye Gamatié, Thierry Gautier. Synchronous Modeling of Avionics Applications using the SIGNAL Language. Real-Time and Embedded Technology and Applications Symposium. IEEE Press, 2003.

[12] Alain Girault. A survey of automatic distribution methods for synchronous programs. In International Workshop on Synchronous Languages, Applications and Programs. Electronic Notes in Theoretical Computer Science. Elsevier, 2005.

[13] Alain Girault, Xavier Nicollin, and Marc Pouzet. Automatic rate desynchronization of embedded reactive programs. ACM Transactions on Embedded Computing Systems, 5(3). ACM Press, 2006.

[14] Olivier Maffeïs. Ordonnancements de graphes de flots synchrones ; application à la mise en oeuvre de SIGNAL. Thèse de l'Universit de Rennes, 1993.

[15] Julien Ouy, Jean-Pierre Talpin, Loïc Besnard, and Paul Le Guernic. *Formal Methods for Globally Asynchronous Locally Synchronous Design*. Electronic Notes in Theoretical Computer Science, Elsevier, 2007.

[16] Dimitru Potop-Butucaru and Benoit Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. In *Application of concurrency to system design*. IEEE Press, 2005.

[17] Dimitru Potop-Butucaru, Benoit Caillaud, and Albert Benveniste. Concurrency in synchronous systems. In *Formal Methods in System Design*. Kluwer Academic Press, 2006.

[18] Jean-Pierre Talpin and Paul Le Guernic. An algebraic theory for behavioral modeling and protocol synthesis in system design. Formal Methods in System Design. Special Issue on formal methods for GALS design. Springer Verlag, 2006.

[19] Jean-Pierre Talpin, Dimitru Potop-Butucaru, Julien Ouy, and Benoit Caillaud. From multi-clocked synchronous specifications to latency-insensitive systems. In *Embedded Software Conference*. ACM Press, 2005.

Appendix - Clocks and scheduling relations

The appendix recall the semantics of synchronization and scheduling relations in the polychronous model of computation, presented in [9]. It is complementary material for information to reviewers. A scheduling structure can be added to the polychronous model of computation outlined in the present article to define a denotational semantics of scheduling relations $x \rightarrow^c y$.

Scheduling structure To render scheduling relations between events occurring at the same time tag t, we equip the domain of polychrony with a scheduling relation, noted $t_x \rightarrow t'_y$, defined on a domain of dates $\mathcal{D} = \mathcal{T} \times \mathcal{X}$, to mean that the event along the signal named y at t' may not happen before x at t. When no ambiguity is possible on the identity of b in a scheduling constraint, we write it $t_x \rightarrow t_y$. We constraint scheduling $\rightarrow$ to contain causality so that $t < t'$ implies $t_x \rightarrow^b t'_x$ and $t_x \rightarrow^b t'_x$ implies $\neg(t' < t)$.

The definitions for the partial order structure of synchrony and asynchrony in the polychronous model of computation extend point-wise to account for scheduling relations. We say that a behavior c is a *stretching* of b, written $b \leq c$, iff $\mathcal{V}(b) = \mathcal{V}(c)$ and there exists a bijection f on $\mathcal{T}$ which satisfies

$$\begin{array}{l}
\forall t, t' \in \mathcal{T}(b), t \leq f(t) \wedge (t < t' \Leftrightarrow f(t) < f(t')) \\
\forall x, y \in \mathcal{V}(b), \forall t \in \mathcal{T}(b(x)), \forall t' \in \mathcal{T}(b(y)), t_x \rightarrow^b t'_y \Leftrightarrow f(t)_x \rightarrow^c f(t')_y \\
\forall x \in \mathcal{V}(b), \mathcal{T}(c(x)) = f(\mathcal{T}(b(x))) \wedge \forall t \in \mathcal{T}(b(x)), b(x)(t) = c(x)(f(t))
\end{array}$$

Meaning of clocks The meaning $[\![e]\!]_b$ of a clock e is defined with respect to a given behavior b and consists of the set of tags satisfied by the proposition e in the behavior b. The meaning of the clock $x = v$ (resp. $x = y$) in b is the set of tags $t \in \mathcal{T}(b(x))$ (resp. $t \in \mathcal{T}(b(x)) \cap \mathcal{T}(b(y))$) such that $b(x)(t) = v$ (resp. $b(x)(t = b(y)(t))$. In particular, $[\![\hat{x}]\!]_b = \mathcal{T}(b(x))$ and $[\![[x]]\!]_b = [\![x = \mathsf{true}]\!]_b$. The meaning of a conjunction $e \wedge f$ (resp. disjunction $e \vee f$ and difference $e \setminus f$) is the intersection (resp. union and difference) of the meaning of e and f. Clock 0 has no tags.

$$\begin{array}{lll}
[\![1]\!]_b = \mathcal{T}(b) \quad [\![0]\!]_b = \emptyset & & [\![e \wedge f]\!]_b = [\![e]\!]_b \cap [\![f]\!]_b \\
[\![x = v]\!]_b = \{t \in \mathcal{T}(b(x)) \,|\, b(x)(t) = v\} & & [\![e \vee f]\!]_b = [\![e]\!]_b \cup [\![f]\!]_b \\
[\![x = y]\!]_b = \{t \in \mathcal{T}(b(x)) \cap \mathcal{T}(b(y)) \,|\, b(x)(t) = b(y)(t)\} & & [\![e \setminus f]\!]_b = b[\![e]\!]_b \setminus [\![f]\!]_b
\end{array}$$

Meaning of scheduling relations A scheduling specification $y \rightarrow x$ at clock e denotes the behaviors b on $\mathcal{V}(e) \cup \{x, y\}$ which, for all tags $t \in [\![e]\!]_b$, requires x to preceed y: if t is in $b(x)$ then it is necessarily in $b(y)$ and satisfies $t_y \rightarrow^b t_x$.

$$[\![y \rightarrow^c x]\!] = \{b \,|\, \mathcal{V}(b) = \mathcal{V}(c) \cup \{x, y\} \wedge \forall t \in [\![c]\!]_b,\, t \in \mathcal{T}(b(x)) \Rightarrow t \in \mathcal{T}(b(y)) \wedge t_y \rightarrow^b t_x\}$$

In [9], we finally show that, whenever a process P has graph R, then $[\![P]\!] \subseteq [\![R]\!]$.

Contents

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399