

On Distributed Program Specification and Synthesis in Architecture with Cycles

Julien Bernet, David Janin

▶ **To cite this version:**

Julien Bernet, David Janin. On Distributed Program Specification and Synthesis in Architecture with Cycles. Internationnal Conference on Formal Methods for Networked and Distributed Systems (FORTE), Sep 2006, France. pp.175-190. hal-00306412

HAL Id: hal-00306412
https://hal.science/hal-00306412

Submitted on 25 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On Distributed Program Specification and Synthesis in Architectures with Cycles

Julien Bernet and David Janin

LaBRI, Université de Bordeaux I,
351, cours de la libération,
F-33 405, Talence Cedex, France
bernet|janin@labri.fr

Abstract. In this paper, we consider discrete distributed synthesis problems, as defined by Pnueli and Rosner [17], on possibly cyclic architectures with zero-delay semantics and global specifications.
We describe a uniform (and complete) translation of these problems into distributed games problems. We prove the correctness of this translation and we also obtain, in this setting, a characterization of distributed architectures with decidable synthesis problems.
It shall be noted that, as opposed to former approaches, zero-delay semantics requires a specific treatment for modeling instantaneous value propagation. Moreover, cyclic dependencies with zero-delay semantics involve equations with potentially many solutions. Accordingly, several variants of the distributed synthesis problem are proposed and studied.

Introduction

Automatic or semi-automatic synthesis of programs from specifications has been for long a challenging research goal in formal methods.

In the context of distributed discrete events systems, Pnueli and Rosner gave one of the first abstract definitions of this problem, proved its general undecidability, and characterized a decidable class of problems: distributed synthesis on the pipeline architecture [17].

Since then, distributed program synthesis has received a lot of attention: in the framework defined by Pnueli and Rosner [5, 10–12], in control theory [2, 13, 1, 4, 20], or in the framework of true concurrency [7, 6]. Many variations of this problem have been considered and solved: cyclic or acyclic architectures, from synchronous to asynchronous communications, interleaved or true concurrent models, with or without zero-delay semantics, with point to point or broadcast communication channels.

From a theoretical point of view, solving a distributed synthesis problem - where programs to be synthesized only have local knowledge of the global state of the system - amounts to solving a multiplayer game with partial information. This general problem has been defined and studied already in [16, 19].

More recently, a regain of interest for distributed synthesis has led to a specialized version of multiplayer games, *distributed games*, that aims at defining a

common framework where distributed synthesis problems can be encoded and solved [14]. In particular, these games have been equipped with various automata based tools [3] that help in this process.

Objective of this Work

In this paper, we first aim at illustrating the relevance of this unifying approach by giving a clear, uniform and complete reduction of distributed synthesis problems into distributed games. As a subgoal, we expect this reduction to be efficient, both in the sense that decidable cases are strictly preserved, and in the sense that the complexity of solving these problems does not increase through the reduction.

In order to do so, we study the distributed synthesis problem in the case of architectures with cycles, zero-delay semantics and branching time global specifications; this case has not been considered so far. It occurs that zero-delay semantics require a specific treatment for modeling instantaneous value propagation. Moreover, cyclic dependencies with zero-delay semantics involve equations with potentially many solutions. Accordingly, several variants of the distributed synthesis problem are proposed and studied.

The relevance of studying cyclic architecture with zero-delay semantics first comes from digital circuits design as illustrated, for instance, by the classical R/S flip flop [8]. In this setting, one may consider extensions of Hardware Description Languages such as VHDL [9] with modules for automatic program synthesis. Studying zero-delay semantics also makes sense at the application level.

Investigating furthermore the relevance of this approach in various application fields like telecommunication or web services, distributed database or parallel scientific computing, etc..., is however not the purpose of this paper. Following Pnueli and Rosner original works and motivations [17], we stick to a level of abstraction that is application independent.

There is little doubt that application of automatic distributed synthesis still requires a lot more research efforts for exhibiting both richer decidable frameworks and more tractable solutions.

Organization of the Paper

A model of zero-delay synchronous behaviors is presented in the first section. On architecture with cycles, it is shown that global behaviors may not be uniquely defined by sets of local behaviors. Accordingly, three variants of the distributed synthesis problem are defined: the angelic, the strict and the demonic variant.

The notion of hierarchical architectures is defined and studied in the second section. It is equivalent to a similar notion defined in [5]. Our main result is then stated: the angelic and strict variant of the distributed synthesis problem with arbitrary MSO specification are decidable if and only if the underlying architecture is hierarchical.

A proof of this theorem is given in the fourth part. More precisely, after briefly reviewing the definition of distributed games, we prove that both variants of the

distributed synthesis problem can be encoded into distributed games that are decidable when the underlying architecture is hierarchical.

Some open problems are presented as a conclusion.

Notations

A word on an alphabet A is a partial function $w : \omega \to A$ with downward-closed domain, i.e. $w(i)$ is the $i+1$th letter of word w. When $dom(w)$ is finite, we say w is a finite word, otherwise w is an infinite word. The length $|w|$ of a word w is the cardinality of its domain. The set of finite words (resp. finite non empty words) over alphabet A is written A^* (resp. A^+), the set of infinite words is written A^ω, the set of finite or infinite words is written A^∞. The empty word is written ε. The concatenation of a word $u \in A^*$ and a word $v \in A^\infty$ is written $u.v$.

Given alphabet B, a B-labeled A-tree is a partial function $t : A^* \to B$ with prefix-closed domain.

Given two sets A and B, we write π_A (resp. π_B) for the left (resp. right) projection from $A \times B$ to A (resp. from $A \times B$ to B). These notations are extended to any subset of $A \times B$ and words on $A \times B$. Given $P \subseteq A \times B$, (resp. $w \in (A \times B)^\infty$), we may also write $P[1]$ for $\pi_A(P)$ and $P[2]$ for $\pi_B(P)$ (resp. $w[1]$ for $\pi_A(w)$ and $w[2]$ for $\pi_B(w)$). These notations are generalized to larger products.

1 Distributed Program Synthesis with Zero-Delay Semantics

In this section, we rephrase Pnueli and Rosner's distributed synthesis problem for zero-delay semantics and arbitrary - possibly cyclic - architectures. In this context, behavior semantics and architecture structures are studied and interrelated one with the other.

1.1 Models of Behaviors

Programs considered in this paper produce a sequence of output events from a sequence of input events. We assume moreover that programs are *synchronous* and *zero-delay*: no output event is produced prior to any input event and every input event produces one and only one output event.

Definition 1. *A* synchronous zero-delay behavior *with input alphabet A and output alphabet B is a mapping $f : A^* \to B^*$ such that there exists a mapping $k_f : A^* \to (A \to B)$, called the* kernel *of f such that $f(\varepsilon) = \varepsilon$ and, for every $u \in A^*$ and $a \in A$, $f(u.a) = f(u).k_f(u)(a)$.*

In the remainder of the text, a synchronous zero-delay behavior $f : A^* \to B^*$ is simply called a sequential function *and, for every $u \in A^*$, $k_f(u) : A \to B$ is called the* one-step behavior *of function f after input u.*

A sequential function $f : A^* \to B^*$ *has* finite memory *when its kernel* $k_f : A^* \to (A \to B)$ *is eventually periodic, i.e. there are some integers* m *and* n *such that, for every* $u \in A^*$ *with* $|u| > m$*, for every* $v \in A^*$ *with* $|v| = n$*,* $k_f(u.v) = k_f(u)$*.*

One can easily check that each sequential function has a unique kernel and, conversely, each mapping $k : A^* \to (A \to B)$ is the kernel of a unique sequential function. Specifying or synthesizing sequential functions thus amounts to specifying and synthesizing their kernels. Moreover, since kernels are infinite $A \to B$-labeled A-trees, Monadic Second Order Logic (MSO) - or any of its sub logics such as LTL, CTL or the mu-calculus - is available for specification purposes and the related infinite tree-automata theory [18] can be applied for synthesis algorithms.

Another interesting characteristic of this notion of kernel, especially for distributed synthesis, is the good behavior of kernels w.r.t. function composition since, in some sense, it commutes with it. More precisely, writing $f;g$ for the composition $g \circ f$, for all sequential functions $f : A^* \to B^*$ and $g : B^* \to C^*$ and for every input sequence $u \in A^*$, one has $k_{f;g}(u) = k_f(u); k_g(f(u))$.

In other words, the one-step behavior of the sequential composition of function f with function g after some time is just the sequential composition of the one-step behaviors of f with the next step behavior of g after the same amount of time.

Remark. In the setting defined by Pnueli an Rosner [17] and considered in subsequent works [5, 14, 10], a sequential behavior $f : A^* \to B^*$ is generated by a A-branching B-labeled A-tree $h_f : A^* \to B$ (with irrelevant root value) by $f(\varepsilon) = \varepsilon$ and for every $u \in A^*$ and $a \in A$, $f(u.a) = f(u).h_f(u.a)$. In this model, for every $u \in A^+$, $h_f(u) \in B$ is the last output produced after input sequence u.

It shall be clear that these two approaches are equivalent in the sense that they both define the same sequential functions. However, dealing with the latter definition is much harder when composing functions. In fact, for all sequential functions $f : A^* \to B^*$ and $g : B^* \to C^*$ and for every input sequence $u \in A^*$, one has $h_{f;g}(u) = h_g(f(u))$. This difficulty entails for instance, in the approach presented in [14], an *asynchronous* encoding of *synchronous* distributed synthesis problems into distributed games. This somehow artificial asynchronism is not necessary as shown in the present paper.

Remark. A sequential function is, by definition, zero-delay. Still, we can provide a semantical definition of a one-delay behavior. A sequential function $f : A^* \to B^*$ is *one delayed* when, for every $u \in A^*$, every a_1 and $a_2 \in A$, $f(u.a_1) = f(u.a_2)$, i.e. the output event produced after a given input event only depends on the previous input events.

Observe that one-delay sequential functions have a very simple characterization in terms of their functional kernel. In fact, a sequential function $f : A^* \to B^*$ is one-delay if and only if, for every $u \in A^*$, the one-step behavior $k_f(u)$ is a constant function.

1.2 Distributed Architectures

Our definition of distributed architecture is adapted from Pnueli and Rosner's definition [17] allowing single write/multiple read channels as in [5].

Definition 2. *A* distributed architecture $\mathcal{H}$ *is defined as a tuple*

$$\mathcal{H} = \langle I, S, r, \{A_c\}_{c \in I \cup S} \rangle,$$

with a finite set I of (global) input channels, a disjoint finite set S of process sites (identified with output channels), a mapping $r : S \to \mathcal{P}(I \cup S)$ that maps every process $p \in S$ to the set of channels $r(p)$ where process p read input values, and, for every channel $c \in I \cup S$, the finite alphabet A_c of possible events on channel c.

We always assume that alphabets are pairwise disjoint. We also always assume that $I \subseteq \bigcup\{r(p) : p \in S\}$, i.e. any input is read by at least one process.

As a notation, we write A for the alphabet of all possible channel events at a given time, i.e. $A = \Pi_{c \in I \cup S}$. For every set of channels $X \subseteq I \cup S$, we write A_X for the product alphabet $\Pi_{c \in X} A_c$. In particular, $A_{r(p)}$ is the input alphabet of process p on the bigger channel formed by all channels of $r(p)$.

Given any sequence $w \in A^+$ of channel input/output events in the architecture $\mathcal{H}$, we write $in(w)$ for the corresponding sequence of events $\pi_{A_I}(w)$ on architecture input channels and we write $out(w)$ for the corresponding sequence of events $\pi_{A_S}(w)$ on architecture output channels. Similarly, for every process $p \in S$, we also write $in_p(w)$ for the corresponding sequence of events $\pi_{A_{r(p)}}(w)$ on process p input channels, and $out_p(w)$ for the corresponding sequence of events $\pi_{A_p}(w)$ on process p output -.

As a particular case, when $r(p) = \emptyset$, we define $A_{r(p)}$ to be a singleton alphabet, say $A_{r(p)} = \{1\}$, with, in this case, $in_p(w) = 1^{|w|}$. The intuition behind this case is that, with no input channels, a process still receives time clicks.

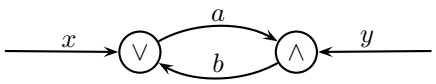
Remark. Two processes that read on the same set of channels (and thus share the same information) can be seen as a single process that writes on two distinct channels.

1.3 Distributed Behaviors

In presence of loops, giving zero-delay semantics to a distributed architecture is a priori non-trivial. Following Pnueli and Rosner [17], the intuitive idea would be to define distributed behavior of an architecture as the global behavior resulting from the composition of local sequential behaviors (one per process).

However, with zero-delay semantics, loops may create cyclic dependencies between the values of the output channels, i.e. systems of equations that may have several solutions. Zero, one or more global behaviors may be compatible with a given set of local behaviors. Consider for instance the system drawn below, where the left-hand process (resp. right-hand process) writes at each time the logical OR (resp. AND) of the last values it reads on its two inputs. Suppose

now that the value of x is 0 (in short $x = 0$) and that $y = 1$. Then, one can either have $a = 0$ and $b = 0$, or $a = 1$ and $b = 1$; hence there are several global behaviors corresponding to this set of local behaviors.



Thus the notion of distributed realization of a global behavior defined by Pnueli and Rosner [17] is no longer functional.

Definition 3. *Let $\mathcal{H} = \langle I, S, r, \{A_c\}_{c \in I \cup S} \rangle$ be a distributed architecture. A* global behavior $f : A_I^* \to A_S^*$ *of architecture $\mathcal{H}$ is* realizable *by a set of* local behaviors $f_p : A_{r(p)}^* \to A_p^*$, *one per process $p \in S$, when the following condition is satisfied:*

> *for every global input/output $u \in A^*$ with $out(w) = f(in(w))$, for every $p \in S$, one has $out_p(w) = f_p(in_p(w))$.*

The set of local behaviors $\{f_p\}_{p \in S}$ is incoherent *(resp.* ambiguous*) when it realizes no (resp. more than one) global behavior.*

In order to solve distributed synthesis problem, we need a more local definition of realizable behavior.

Definition 4 (One-step realizability). *A* global one-step behavior $k : A_I \to A_S$ *is* one-step realized *by a set of* local one-step behaviors $k_p : A_{r(p)} \to A_p$, *one per process $p \in S$, when the following condition is satisfied:*

> *for every global input/output events $a \in A$, such that $out(a) = k(in(a))$, one has, for every process $p \in S$, $out_p(a) = k_p(out_p(a))$.*

A set of local one-step behavior $\{k_p\}_{p \in S}$ is called incoherent *(resp.* ambiguous*) when it realizes no (resp. more than one) global one-step behavior.*

Remark. From this definition, one may be tempted to (re)define realizable global behaviors as sequential functions $f : A_I^* \to A_S^*$ such that, for every $u \in A_I^*$, the one-step global behavior $k_f(u)$ has a one-step realization.

Unfortunately, such a definition would be wrong as it would miss the fact that, for every process p, after any sequence of global input/output $w \in A^*$ with $out(w) = f(in(w))$, the one-step behavior of every process p can only depend on the input sequence $in_p(w)$ actually read by process p.

Both definitions of one-step and general realizability are still related as follows:

Lemma 1. *A* global behavior $f : A_I^* \to A_S^*$ *of architecture $\mathcal{H}$ is* realizable *by a set of* local behaviors $f_p : A_{r(p)}^* \to A_p^*$, *one per process $p \in S$, if and only if, for every global input/output sequence of events $w \in A^*$ with $out(w) = f(in(w))$, the set of one-step local behaviors $\{k_{f_p}(in_p(w))\}_{p \in S}$ realizes the global one-step behavior $k_f(in(w))$.*

Proof. For any $w \in A^*$, for any $v \in A$ with $f(in(w.v)) = out(w.v)$, for any process $p \in S$, one has $f_p(in_p(w.v)) = out_p(w.v)$, thus :

$$k_{f_p}(in_p(w))(in_p(v)) = out_p(w).out_p(v)$$

Since $f_p(in_p(w)) = out_p(w)$, it is clear that the global one-step behavior $k_f(in(w))$ is realized by the set of local one-step behaviors $\{k_{f_p}(in_p(w))\}_{p \in S}$. The converse is clearly true. □

Remark. Observe that, on *acyclic* architecture, a set of *zero-delay* local behaviors is always coherent and non ambiguous. Observe also that, on *arbitrary* architecture, a set of *one-delay* local behaviors is also always coherent and non ambiguous.

1.4 Distributed Synthesis Problems

The distributed synthesis problem is the following: given a specification of an expected global behavior find a distributed realization of it, *i.e.* a set of local behaviors, one for each process site, such that the corresponding global behaviors meet the specification.

With arbitrary zero-delay local behaviors several cases are possible. This leads us to consider three possible semantics for the synthesis problem.

Definition 5 (Distributed synthesis problem). *Given an architecture $\mathcal{H} = \langle I, S, r, \{A_c\}_{c \in I \cup S} \rangle$, given a specification φ of sequential functions with input alphabet A_I and output alphabet A_S, the angelic, strict or, resp. demonic distributed synthesis problem for $\langle \mathcal{H}, \varphi \rangle$ is to find a set of finite memory local sequential behavior $\{f_p\}_{p \in S}$ such that :*

- angelic case: *there is at least one function f realized by $\{f_p\}_{p \in S}$ such that $f \models \varphi$,*
- strict case: *there is a unique function f realized by $\{f_p\}_{p \in S}$ and, moreover, $f \models \varphi$,*
- *or,* demonic case: *the set of local behaviors $\{f_p\}_{p \in S}$ is coherent and for every function f realized by $\{f_p\}_{p \in S}$, one has $f \models \varphi$.*

Remark. The intuition behind these definitions is the following. In the angelic case, the programmer has the opportunity to add extra (hopefully limited) control channels in the architecture that allow control over the choice of the global behavior to be realized. In the strict case, these extra control channels described above are no longer needed: the architecture and the global specification are permissive enough to allow their (implicit) encoding within the architecture itself. Last, in the demonic case, extra control is just not available.

Observe that a distributed synthesis problem that has a solution with strict semantics also has a solution with demonic semantics. The main issues about these three semantics is the decidability problem.

It occurs that, as shown in the next section, both angelic and strict distributed synthesis problem are, as in the one-delay or the acyclic case, decidable on architectures called hierarchical. The demonic case remains an intriguing open problem.

2 Distributed Synthesis on Hierarchical Architectures

We review here the notion of knowledge of a process in an architecture. This leads to define hierarchical architecture and to state our main result in the angelic and strict case.

2.1 Process Knowledge

A similar notion is defined by Finkbeiner and Schewe in [5]. Both lead to equivalent notions of hierarchical architectures on the class of architecture common to both approaches. However, since this notion is somehow subtle and for the sake of completeness, we give here our own definition and related intuition.

Definition 6. *Given architecture $\mathcal{H}$ as above, for every process $p \in S$, we define the* knowledge *of process p to be the* greatest *set of channels $K_p \subseteq I \cup S$ such that:*

> *for all $q \in K_p$, either $q \in I$ and $q \in r(p)$, or $q \in S$ with $q \neq p$ and $r(q) \subseteq K_p$.*

The knowledge relation $\preceq_{\mathcal{H}}$ *is then defined on S to be the relation defined by $p \preceq_{\mathcal{H}} q$ when $q \in K_p$, meaning, informally, that* process p potentially knows more than process q.

One can check that the knowledge relation $\preceq_{\mathcal{H}}$ is a preorder, i.e. it is reflexive and transitive. In the sequel, we write $\simeq_{\mathcal{H}}$ for the induced equivalence relation, i.e. $p \simeq_{\mathcal{H}} q$ when $p \preceq_{\mathcal{H}} q$ and $q \preceq_{\mathcal{H}} p$.

At every moment in an running distributed architecture, the *immediate knowledge* of a process p is just the sequence of inputs it is receiving on channels of $r(p)$ and the sequence of outputs it is producing on channel p. The intended meaning of the knowledge relation is to capture a notion of *deducible knowledge* process p may have from its own immediate knowledge.

The following lemma gives a semantical characterization of the knowledge relation defined above:

Lemma 2. *For every process p, K_p is the set of channels q such that, for every $k : A_I \to A_S$ that is one-step realizable, for every a_1 and $a_2 \in A$, such that $out(a_1) = k(in(a_1))$ and $out(a_2) = k(in(a_2))$, if $in_p(a_1) = in_p(a_2)$ then $in_q(a_1) = in_q(a_2)$.*

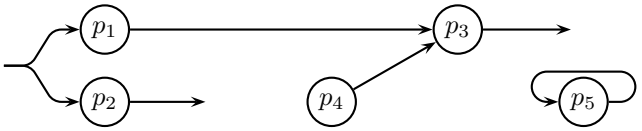
Proof. The full proof is omitted here due to space restrictions. Essentially, it suffices to remark that each process site q that is not in K_p is such that there exists an input channel $x \notin r(p)$ and a path from x to q that avoids p. Using the fact that there is a one-step realization of k, one can show that $in_p(a_1) = in_p(a_2)$ and $in_q(a_1) \neq in_q(a_2)$ if and only if q satisfies this path condition. □

2.2 Hierarchical Architectures

We (re)define here the notion of hierarchical architectures that is induced by the notion of process knowledge.

Definition 7. *An architecture is called* hierarchical *when the knowledge relation is total, i.e. for every p and $q \in S$, either $p \preceq_{\mathcal{H}} q$ or $q \preceq_{\mathcal{H}} p$. Equivalently, an architecture is hierarchical when the quotient set $S/\simeq_{\mathcal{H}}$ is linearly ordered by the relation $\preceq_{\mathcal{H}}$.*

It shall be clear that, on architectures that are common to both definitions, the definition presented here and the one of Finkbeiner and Schewe [5] are the same.



In the above example, one has $p_1 \simeq p_2 \preceq p_3 \preceq p_4 \simeq p_5$: it is therefore hierarchical.

2.3 Main Result

Theorem 1. *The angelic or strict distributed synthesis problem for architecture $\mathcal{H}$ is decidable with arbitrary global MSO specification φ if and only if the architecture $\mathcal{H}$ is hierarchical.*

Proof. In section 3, both angelic and demonic distributed synthesis problem on hierarchical architectures are encoded into pipeline distributed games (in the sense of [14, 3]) that are thus decidable.

Conversely, any non hierarchical architecture contains an undecidable pattern (in the sense of [11]) or an information fork (in the sense of [5]) hence it is undecidable (even with one-delay semantics). □

Since the one-delay semantics is a particular case of (say strict) zero-delay semantics (the one-delay assumption can be encoded into the global specification) this result generalizes previous result for distributed synthesis problems on architecture with global specification.

3 Game Encodings of Distributed Synthesis Problems

In this section, we show that (strict or angelic) distributed synthesis problems can be encoded into distributed games. On hierarchical architecture one gets pipeline games. Since these games are decidable, this induces a decision procedure for the distributed synthesis problem for hierarchical architectures (for both strict and angelic semantics). .

3.1 Distributed Games

Distributed games [14] are a special kind of multiplayer games with partial information [15] extended to infinite plays. In short, two-player games are played on a bipartite graph, in which each position belongs to either the first player (called the Process) or to the second player (called the Environment). Distributed games are an extension of two-player games, where n Process players play together against one Environment player.

Definition 8 (Game arenas). *A one-Process (or two players) game arena is a tuple $\mathcal{G} = \langle P, E, T, s\rangle$ where P is a set of Process position, E is a set of Environment position, $T \subseteq P \times E \cup E \times P$ is a set of possible transition moves, and $s \in P \cup E$ is an initial position.*

Given n one-Process game arenas $\mathcal{G}_i = \langle P_i, E_i, T_i, s_i\rangle$ for $i \in [1, n]$, a synchronous distributed game arena *$\mathcal{G}$ built from the local game arenas $\mathcal{G}_1$, ..., $\mathcal{G}_n$, is a game arena $\mathcal{G} = \langle P, E, T, s\rangle$ with $P = \prod_i P_i$, $E = \prod_i E_i$ and $s = (s_1, \ldots, s_n)$, and such that the set of moves T satisfies the following conditions: for every $u \in P$ and $v \in E$:*

- *Process team: $(u, v) \in T$ if and only if for every $i \in [1, n]$, $(u[i], v[i]) \in T_i$,*
- *Environment: if $(v, u) \in T$ then for every $i \in [1, n]$, $(v[i], u[i]) \in T_i$.*

Remark. Observe that there is a unique distributed game arena built from the local arenas $\mathcal{G}_1$, ..., $\mathcal{G}_n$ with maximal set of Environment moves. This arena, written $\mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$, is called the free synchronous product of the arenas $\mathcal{G}_1$, ..., $\mathcal{G}_n$.

Observe that any other distributed arena $\mathcal{G}$ built from the same local arenas can just be seen as a subgame of the free product obtained by possibly disallowing some Environment moves. It follows that, in the sequel, we will use the notation $\mathcal{G} \subseteq \mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$ to denote this fact.

Remark. In [14] or in [3], a more general notion of distributed with *asynchronous* moves is defined. For the study presented here, the additional expressiveness gained with asynchronism is not used in this paper. Since we are essentially establishing lower bounds result, this fact makes our result even stronger.

Definition 9. *Given a two player game arena $\mathcal{G} = \langle P, E, T, s\rangle$, a* strategy *for the Process player (resp. a* strategy *for the Environment player) is a mapping $\sigma : P^+ \to E$ (resp. a mapping $\tau : E^+ \to P$).*

From the initial position $s \in E + T$, the play induced by strategies σ and τ from position s, written $\sigma * \tau$ is defined to be the maximal word $w \in (P + E)^\infty$ such that $w(0) = s$ and for every $i \in dom(w)$ with $i > 0$, $(w(i-1), w(i)) \in T$ and, given $w' = w(0) \cdots w(i-1)$, if $w(i-1) \in P$ then $w(i) = \sigma \circ \pi_P(w')$ and if $w(i-1) \in E$ then $w(i) = \tau \circ \pi_E(w')$.

Strategy σ for Process is non blocking *when, for every counter strategy τ, if $\sigma * \tau$ is finite then it ends in an Environment position.*

Given an n-process game arena $\mathcal{G} \subseteq \mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$, a Process strategy $\sigma : P^+ \to E$ is a distributed strategy *where there is a set of local process strategies $\{\sigma_i :$*

$P_i^+ \to E\}_{i\in[1,n]}$ *such that, for every word* $w \in P^+$, $\sigma(w) = (\sigma_1 \circ \pi_{P_1}(w), \cdots, \sigma_n \circ \pi_{P_n}(w)$.

In other words, a Process strategy is distributed when every local Process player only plays following its own local view of the global play.

Until now, we didn't specify how to win in such a game. For any play w that ends in a position without successor, the convention we use is to declare that w is won by the Processes (resp. by the Environment) if and only if the last position of w belongs to E (resp. to P). This corresponds to the idea that whenever a player may not make a game move anymore, it looses the play.

Since loops are allowed in the game arena, a Process strategy may also induce infinite plays. A *winning condition* for a distributed arena $\mathcal{G} = \langle P, E, T, s\rangle$ should therefore be defined as a set Acc of infinite plays won by the Process, *i.e.* infinite words from $(P+E)^\omega$. In order to describe such a set in a finite way, one can either use some decidable logic over infinite words (either monadic second order logic MSO, or one of its specializations, such as μ-calculus or LTL). In the scope of this paper, however, we do not want to be specific about the formalism chosen for the specification; therefore we simply assume that Acc is an ω-regular language (equivalently a language definable in MSO).

Definition 10 (Distributed games and winning strategies). *A* distributed game *is a tuple* $\mathcal{G} = \langle P, E, T, s, Acc\rangle$ *where* $\langle P, E, T, s\rangle$ *is a distributed arena, and* $Acc \subseteq (P + E)^\omega$ *is an* ω*-regular winning condition.*

A distributed Process strategy σ *in game* $\mathcal{G}$ *is a winning strategy when it is non blocking and for any strategy* τ *for the Environment, if the play* $\sigma * \tau$ *(from initial position* s*) is infinite then it belongs to* Acc.

Observe that a distributed game arena can just be seen as a distributed game with infinitary winning condition $Acc = (P+E)^\omega$. *In this case, winning strategies and non blocking strategies are the same.*

3.2 Distributed Games for the Strict Case

We prove here that unambiguous realizable behaviors can be encoded as distributed non blocking strategies in distributed arenas.

Definition 11. *Let* $\mathcal{H} = \langle I, S, r, \{A_c\}_{c\in I\cup S}\rangle$ *with* $S = \{1, \ldots, n\}$ *a* n*-process distributed architecture. We define the* n*-process* strict distributed arena $\mathcal{G}^S_\mathcal{H} = \langle P, E, T, s\rangle$ *from a free synchronous product arena* $\mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$ *of the game arenas* $\mathcal{G}_i$ *as follows.*

For every $p \in \{1, \cdots, n\}$, *the game arena* $\mathcal{G}_p = \langle P_p, E_p, T_p, s_p\rangle$ *is defined by taking*

1. $P_p = \{*_p, \bot_p\} \cup A_{r(p)}$
2. $E_p = (A_{r(p)} \to A_p)$,
3. T_p *is the union of the sets* $(P_p - \{\bot_p\}) \times E_p$ *and* $E_p \times (P_p - \{*_p\})$,
4. *and* $s_p = *_p$,

The intended meaning of this game arena is that, at every step, Process p chooses a one-step local behavior (in $A_{r(p)} \to A_p$) and Environment answers by choosing one local direction.

The distributed arena $\mathcal{G}^S_{\mathcal{H}}$ is then defined from the free product by restricting Environment global moves as follows: from an Environment distributed position $e = (k_p)_{p\in[1,n]}$,

1. *Environment checks that the set of one-step local behaviors $\{k_p\}_{p\in[1,n]}$ is an unambiguous* one-step realization *of a global one-step behavior k_e,*
2. *if so, Environment chooses a global input event $a = (a_c)_{c\in I}$, compute the corresponding global output event $(a_c)_{c\in S} = k_e(a)$, and distribute back to processes their corresponding local inputs, i.e. Environment moves to Process distributed position $(b_p)_{p\in[1,n]}$ with $b_p = (a_c)_{c\in r(p)}$), otherwise Environment moves to the final position $(\bot_1, \bot_2, \cdots, \bot_n)$.*

Unambiguous distributed behaviors of architecture $\mathcal{H}$ are encoded as non blocking distributed strategies in the distributed arena $\mathcal{G}^S_{\mathcal{H}}$ as follows. Every process $p \in S$ defines, step by step, in game $\mathcal{G}_p$, the local behavior process p will have in architecture $\mathcal{H}$. The environment player checks that these choices are compatible one with another in such a way that the resulting global behavior is well defined (and thus has a coherent and unambiguous distributed realization).

Theorem 2. *A distributed strategy $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$ is non blocking in the game arena $\mathcal{G}^S_{\mathcal{H}}$ if and only if the set $\{f_{\sigma_p}\}_{p\in S}$ of the behaviors defined on local games $\mathcal{G}_1$, ..., $\mathcal{G}_n$ by strategies σ_1, ..., σ_n is the distributed realization of an unambiguously realizable sequential function $f_\sigma : A^*_I \to A^*_S$.*

In particular, a strategy σ is finite memory if and only if the sequential function f_σ is finitely generated (i.e. it has a finite memory kernel).

Proof. Let $\sigma : P^* \to E$ be a non blocking distributed strategy for the process team with $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$.

By definition, from every coherent and unambiguous $e \in E$ there is a unique mapping $k_e : A_I \to A_S$ locally realized by e. Moreover, for every input value $a \in A_I$, there is one and only one position $p_{e,a} \in P$ where environment player can move to and, moreover, value a can be read in values stored in $p_{e,a}$ hence all positions $\{p_{e,a}\}_{a\in A_I}$ are distinct one from the other.

It follows that there is a unique mapping $h_\sigma : A^*_I \to P^+$ such that $h_\sigma(\varepsilon) = (*_1, \cdots, *_n)$ and, for every $u \in A^*_I$, for every $a \in A_I$, given $e = \sigma(h_\sigma(u))$, one has $h(u.a) = h_\sigma(u).p_{e,a}$.

We define then the mapping $k_\sigma : A^*_I \to (A_I \to A_S)$, the *functional kernel* of the sequential function induced by strategy σ, by taking, for every $u \in A_I$, $k_\sigma(u) = k_e$ with $e = \sigma(h_\sigma(u))$.

By construction, k_σ is the functional kernel of an unambiguously realizable behavior f_σ of architecture $\mathcal{H}$. In fact, for every $u \in A^*_I$, the environment position $\sigma(h_\sigma(u))$ is the local realization of $k_\sigma(u)$ since it is a coherent and unambiguous position hence Lemma 1 applies.

Conversely, let $f : A_I^* \to A_S^*$ be a distributed architecture behavior realized by a coherent and unambiguous set of local process behaviors $\{f_p\}_{p \in S}$. Given, for every $p \in S$ a non blocking strategy σ_p in game $\mathcal{G}_p$ that corresponds to behavior f_p, it is not hard to see that the distributed strategy $\sigma_f = \sigma_1 \otimes \cdots \otimes \sigma_n$ is non blocking in the distributed arena $\mathcal{G}_\mathcal{H}$. □

3.3 Distributed Games for the Angelic Case

We prove here that coherent realizable behaviors can be encoded as non blocking distributed strategies in distributed arenas.

Definition 12. *Again, let $\mathcal{H} = \langle I, S, r, \{A_c\}_{c \in I \cup S} \rangle$ with $S = \{1, \ldots, n\}$ an n-process distributed architecture. We define the $n+1$-process* angelic distributed arena $\mathcal{G}_\mathcal{H}^A = \langle P, E, T, s \rangle$ *from a free synchronous product arena $\mathcal{G}_0 \otimes \mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$ as follows.*

For every $p \in \{1, \cdots, n\}$, the game arena $\mathcal{G}_p = \langle P_p, E_p, T_p, s_p \rangle$ is defined as in the strict case (see Definition 12) and the game arena $\mathcal{G}_0 = \langle P_0, E_0, T_0, s_0 \rangle$ is defined as follows:

1. $P_0 = \{*_0, \bot_0\} \cup A_I$
2. $E_0 = (A_I \to A_S)$,
3. T_0 *is defined to be the union of the sets* $(P_0 - \{\bot_0\}) \times E_0$ *and* $E_0 \times (P_0 - \{*_0\})$,
4. *and* $s_0 = *_0$.

The intended meaning of this game arena is that, at every step, Process 0 *chooses a one-step* global behavior *and Environment answers by choosing one global input.*

The distributed arena $\mathcal{G}_\mathcal{H}^A$ is then defined from the free product by restricting Environment global moves as follows: from an Environment distributed position $e = (k_p)_{p \in [0,n]}$,

1. *Environment checks that the set of one-step local behaviors $\{k_p\}_{p \in [1,n]}$ is a* one-step realization *of the global one-step behavior k_0,*
2. *if so, Environment chooses a global input event $a = (a_c)_{c \in I}$, compute the corresponding global output event $(a_c)_{c \in S} = k_e(a)$, and distributes back to processes their corresponding local inputs, i.e. Environment moves to Process position $(b_p)_{p \in [0,n]}$ with $b_p = (a_c)_{c \in r(p)}$ when $p \in [1, n]$ and $b_p = a$ when $p = 0$, otherwise Environment moves to the final position $(\bot_0, \bot_1, \bot_2, \cdots, \bot_n)$.*

Coherent distributed behaviors of architecture $\mathcal{H}$ are encoded as non blocking distributed strategies in the distributed game $\mathcal{G}_\mathcal{H}$ as follows. Every process $p \in S$ defines, step by step, in game $\mathcal{G}_p$, the local behavior process p will have in architecture $\mathcal{H}$, and process 0 defines, step by step, the intended global realizable behavior. The environment player checks that these choices are compatible one with the other in such a way that the chosen global behavior defined by player 0 is realizable by the coherent (though possibly ambiguous) set of local behaviors that are built by the other players.

Theorem 3. *A distributed strategy $\sigma = \sigma_0 \otimes \sigma_1 \otimes \cdots \otimes \sigma_n$ is non blocking in game $\mathcal{G}^A_{\mathcal{H}}$ if and only if the set $\{f_{\sigma_p}\}_{p\in[1,n]}$ of the local behaviors defined on local games $\mathcal{G}_1, \ldots, \mathcal{G}_n$ by strategies $\sigma_1, \ldots, \sigma_n$ is a distributed realization of the global behavior $f_{\sigma_0} : A_I^* \to A_S^*$ defined on local game $\mathcal{G}_0$.*

In particular, strategy σ is finite memory if and only if the sequential function f_{σ_0} is finitely generated (i.e. it has a finite memory kernel).

Proof. The argument are essentially the same as in the proof of Theorem 2. □

3.4 Distributed Synthesis Problem in Distributed Games

Now we show that any n-process strict distributed synthesis problem on hierarchical architecture can be encoded into a n-process pipeline distributed game.

The first step is to prove that:

Lemma 3. *If architecture $\mathcal{H}$ is hierarchical then both distributed games $\mathcal{G}^S_{\mathcal{H}}$ and $\mathcal{G}^A_{\mathcal{H}}$ are pipeline game arenas in the sense of [14, 3].*

Proof. This follows immediately from the definition of hierarchical architecture (see section 2.2), Lemma 2 and Definitions 11 or Definition 12. In fact, Environment always transmit local inputs to Process players. It follows that any linearization of the knowledge order on processes will give an order that process the game is a pipeline game [14].

In the angelic case, Process 0 knows the global input and the global behavior. It follows that he also knows every other process inputs. It is thus already a leader (see [14, 3]) and can be added as the least element in this total order. □

It follows:

Theorem 4. *For every hierarchical architecture $\mathcal{H} = \langle I, S, r, \{A_c\}_{c\in I\cup S}\rangle$ with n Process players and every MSO specification φ of (kernel of) sequential function from A_I^* to A_S^* there is an $n+1$-process for the strict case (resp. $n+2$-process for the angelic case) decidable distributed game $\mathcal{G}^S_{\langle\mathcal{H},\varphi\rangle}$ (resp. $\mathcal{G}^A_{\langle\mathcal{H},\varphi\rangle}$) such that there is an unambiguously realizable (resp. realizable) behavior for $\mathcal{H}$ that satisfies specification φ if and only if there is a (finite memory) distributed winning strategy for the process team in game $\mathcal{G}^S_{\langle\mathcal{H},\varphi\rangle}$ (resp. $\mathcal{G}^A_{\langle\mathcal{H},\varphi\rangle}$).*

Proof. First, one can easily translate the global specification φ to a global strategy specification ψ that is satisfied only by global strategies that encode global behaviors that satisfy φ. Then the result immediately follows from Theorem 2, Theorem 3, Lemma 3 and the fact that, as described in [3], pipeline games with such external conditions are decidable. □

4 Conclusion

We have shown that strict and angelic distributed synthesis problem are decidable on hierarchical architectures with zero-delay semantics.

The demonic case remain, so far, an open problem. For proving decidability, one may try to adapt the above proof to this case by letting Environment (instead of player 0 in the angelic case) choose any behaviors realized by the local behaviors built by Processes. But this would break the pipeline structure of the resulting game so this approach would be non conclusive.

On the other side, for proving undecidability in the presence of a cycle in the architecture, one may try to force - by means of the external specification - some subset of processes to choose coherent, but ambiguous, local behaviors that would induce equations with multiple solutions. Then, following demonic semantics, Environment could pick arbitrary values among these solutions, creating thus another arbitrary input in the architecture that could behave such as a typical undecidable architecture. But this approach is inconclusive too since we do not know, so far, how to force in a global specification such a kind of set of ambiguous local behaviors.

Less directly related with our proposal, one may observe that, in the case of non hierarchical architecture, with the notable exception of local specifications[12], hardly any restriction on the global specification have been established for increasing the class of decidable distributed synthesis problems. This is certainly a open research direction that could be followed. The notion of process knowledge could be tuned to take into account the global specification. Distributed games, where both architecture and specification have been merged, could serve as a tool to achieve new results in this direction.

Even more distant from our present work, but still related, one can also observe that asynchronous behaviors, with fairness assumption that guarantees only finitely many output events are produced after every single input event, can be encoded by extending the notion of kernels to mapping from A^* to $A \rightarrow B^*$. Though the resulting vertex labeling would be on an infinite alphabet, the distributed game techniques that are used here could be extended to this case.

Acknowledgment

We are grateful to anonymous referees. Their comments strongly helped us revising former versions of this work. We would also like to thank Dietmar Berwanger for his suggestions for the final version of this paper.

References

1. L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang. The control of synchronous systems, part II. In *CONCUR*, volume 2154 of *LNCS*, pages 566–582, 2001.
2. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controlers with partial observation. *Theoretical Comp. Science*, 303(1):7–34, 2003.
3. J. Bernet and D. Janin. Tree automata and discrete distributed games. In *Foundation of Computing Theory*, volume 3623 of *LNCS*, pages 540–551. Springer-Verlag, 2005.
4. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

5. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 321–330, 2005.
6. P. Gastin, B. Lerman, and M. Zeitoun. Causal memory distributed games are decidable for series-parallel systems. In *Proceedings of FSTTCS'04*, LNCS. Springer-Verlag, 2004. To appear.
7. P. Gastin, B. Lerman, and M. Zeitoun. Distributed games and distributed control for asynchronous systems. In *Proceedings of LATIN'04*, volume 2976 of *LNCS*, pages 455–465. Springer-Verlag, 2004.
8. J. L. Hennessy and D. A. Patterson. *Computer organization and design (2nd ed.): the hardware/software interface.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
9. IEEE Std 1076-1993. *IEEE Standard VHDL*, 1993.
10. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 389–398, 2001.
11. P. Madhusudan. *Control and Synthesis of Open Reactive Systems.* PhD thesis, University of Madras, 2001.
12. P. Madhusudan and P.S. Thiagarajan. Distributed control and synthesis for local specifications. In *Int. Call. on Aut. and Lang. and Programming (ICALP)*, volume 2076 of *LNCS*. Springer-Verlag, 2001.
13. P. Madhusudan and P.S. Thiagarajan. A decidable class of asynchrnous distributed controlers. In *CONCUR'02*, volume 2421 of *LNCS*, pages 145–160. Springer-Verlag, 2002.
14. S. Mohalik and I. Walukiewicz. Distributed games. In *Found. of Soft. tech and Theor. Comp. Science*, volume 2914 of *LNCS*, pages 338–351. Springer-Verlag, 2003.
15. G.L. Peterson and J.H. Reif. Multiple-person alternation. In *20th Annual IEEE Symposium on Foundations of Computer Sciences*, pages 348–363, october 1979.
16. G.L. Peterson, J.H. Reif, and S. Azhar. Decision algorithms for multiplayer noncooperative games of incomplete information. *Computers and Mathematics with Applications*, 43:179–206, january 2002.
17. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *IEEE Symposium on Foundations of Computer Science*, pages 746–757, 1990.
18. M. O. Rabin. Decidability of second order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.
19. J.H. Reif. Universal games of incomplete information. In *11th Annual ACM Symposium on Theory of Computing*, pages 288–308, 1979.
20. K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.