Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 180 (2007) 71–89
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

On the Expressiveness of Timed Coordination via Shared Dataspaces

Isabelle Linden and Jean-Marie Jacquet[1]

Institute of Informatics
University of Namur
Namur, Belgium

Abstract

Since Linda, many differents coordination models using shared dataspaces have been developped. However, a few only have incorporated the notion of time. This paper builds upon previous work to study the expressive power of two families of timed coordination models based on shared dataspaces. The first one relies on Linda's communication primitives whereas the second relies on the more general notion of multi-set rewriting, incorporated, for instance, in Gamma. We analyse the expressiveness increase provided by the primitives in each of the two families and also compare the expressiveness power of the two families.

Keywords: Expressiveness, Shared dataspaces, Timed coordination

1 Introduction

As motivated by the constant expansion of computer networks and illustrated by the development of distributed applications, the design of modern software systems centers on re-using and integrating software components. This induces a paradigm shift from stand-alone applications to interacting distributed systems, which, in turn, naturally calls for well-defined methodologies and tools aiming at integrating heterogeneous software components. One of these tools consist of coordination languages and models, which, following Gerlernter's Linda ([10]) proposal, have advocated the interest of clearly separating *interactional* and the *computational* aspects of software components.

In Linda, communication between agents does not cope with time. However, data rarely has an eternal life and most of services have to be provided in a bounded amount of time. For exemple, a request at a bank machine has to be satisfied in a reasonable amount of time. Moreover, time is considered as critical in areas such as

[1] Email: ili,jmj@info.fundp.ac.be

doi:10.1016/j.entcs.2006.10.047

traffic control and telecommunication switches. Finally, industrial proposals such as TSpaces and JavaSpaces have incorporated time constructs.

In our recent work [14,13], we have studied the introduction of time in Linda-like models in four differents ways, by using two notions of time, relative time and absolute time, and, for each notion, two types of features: delay mechanism and explicit deadlines on the validity of tuples and on the duration of suspension of communication operations. In addition to the description of the language primitives, elementary expressiveness results have been presented and implementation techniques have been detailed.

In this paper, we extend our study to the introduction of time in multiset based coordination languages. To that end, we shall only consider time in a relative manner. Following previous work, we shall use the so-called *two-phase functioning* approach to real-time systems illustrated by languages such as Lustre ([7]), Esterel ([2]) and Statecharts ([11]). This approach may be described as follows. In a first phase, elementary actions of statements are executed. They are assumed to be atomic in the sense that they take no time. Similarly, composition operators are assumed to be executed at no cost. In a second phase, when no actions can be reduced or when all the components encounter a special timed action, time progresses by one unit. Although simple, this approach has been proved to be effective for modelling reactive systems.

To our best knowledge, this paper is the first one to propose a time extension to multiset based coordination languages and to study their expressiveness.

Related proposals for the introduction of time in coordination-like languages mainly fall in the category of relative time languages and for variants of Linda languages. For instance, [18] introduces time in the concurrent constraint setting [2] ([20]) by identifying quiescent points in the computation where no new information is introduced and by providing an operator for delaying computations by one unit. At each quiescent point of time, the dataspace is reinitialized to an empty content. The paper [19] extends this framework, on the one hand, by introducing a primitive for checking the absence of information and reacting on this absence during the same unit of time and, on the other hand, by generalizing the delay mechanism in an *hence* A construct which states that A holds at every instant after the considered time. The resulting languages are called *tcc* and *tdcc*.

The paper [23] has shown that the language *tcc* can embed one classical representative of the state oriented synchronous languages, namely Argos ([15]), and one representative of the declarative class of dataflow synchronous languages, namely Lustre ([7]).

De Boer, Gabbrielli, and Meo have presented in [3] a timed interpretation of concurrent languages by fixing the time needed for the execution of parallel tell and ask operations as one unit and by interpreting action prefixing as the next operator. A delay mechanism is presented in Oz ([22]), a language which combines object oriented features with symbolic computation and constraints, and, (relative) time-outs

[2] Concurrent constraint languages may be viewed as a variant of Linda restricted to two communication primitives putting information of a dataspace and checking the presence of information on it

have been introduced in TSpaces ([24]) and JavaSpaces ([9]). A formal semantics of these time-outs and other mechanisms, different from our expressiveness study, is presented in [5].

Another piece of work on the expressiveness of timed constraint system is [17]. There, various extensions of the *tcc* languages have been studied: extension with replication and recursion static scoping. Decidability results are proved as well as several encodings, which are however not of the form of modular phased embeddings used in this paper.

Finally, [6] investigates the impact of various mechanisms for expired data collection on the expressiveness of coordination systems. However, the study is based on Random Access Machines, on ordered and unordered tells of timed data and on decidability results.

The rest of the paper is structured as follows. Section 2 introduces the two families of languages under study in the paper. Section 3 presents the framework for comparing the expressiveness of languages. To that end, we shall refine the notion of modular embedding proposed in [8] to our time context presented in phases. This will lead to the notion of phased-embedding. With these comparison tools, section 4 studies the expressive increasing provided by successive introduction of primitves *tell*, *ask*, *get*, *nask* and *delay* in each of the two families of languages. The two families are then compared in section 5. Finally, section 6 draws our conclusion.

2 The families of coordination languages

The families under study in this paper are described by first introducing the common syntax and rules, then by defining the family based on Linda like primitives and finally that based on multiset rewriting.

2.1 Common syntax and rules

We shall consider two families of languages $\mathcal{R}(\mathcal{X})$ and $\mathcal{M}(\mathcal{X})$ parameterised w.r.t. the set of communication primitives $\mathcal{X}$. The set is in turn a subset of a general set of communication primitives depending on the family under consideration. Assuming this general set, all the languages use sequential, parallel, and choice operators (see "General rule" in figure 1), whose meaning is defined by the usual rules (S), (P), and (C) in figure 2. There the configurations are of the form $\langle A \mid \sigma \rangle$ where A represents the agent under consideration and σ represents, the dataspace, subsequently called the store and which consists of a multiset of subscripted tokens.

As already said, we shall adopt the classsical *two-phase functioning* approach to real-time system. Accordingly, time needs to be taken into account explicitly in the transitions. This is achieved in two ways. First by the introduction of transition rules which define a transition relation $\leadsto$ to express the progress of time by one unit. In fact, the $\rightarrow$ reduction is used to model the first phase of the two-phase functioning approach to real-time while the $\leadsto$ relation is used to model the second phase of this approach. Second, as a result of the progress of time, delays under reduction, must be decreased by one unit.

General rule

$$A ::= C \mid A\,;\ A \mid A \,||\, A \mid A\ +\ A$$

$\mathcal{R}$ rule

$$C ::= tell_d(t) \mid ask_d(t) \mid get_d(t) \mid nask_d(t) \mid delay(d)$$

$\mathcal{M}$ rule

$$C ::= (\{M\}_d, \{M\}_d) \mid delay(d)$$
$$M ::= \lambda \mid\ +t \mid\ -t \mid M, M$$

Fig. 1. Comparative syntax of the languages.

A delay primitive is introduced in each family. As expected, the effect of $delay(d)$ is to postpone the computation of d units of time. The resolution of $delay(0)$ is described in rule (D).

2.2 The $\mathcal{R}$ family

The $\mathcal{R}$ family is based on Linda. Following our previous work, we shall use a slightly different modelling of the Linda primitives. We shall consider four primitives: *tell* to place a token on the store, *ask* to test the presence of a token, *get* to remove this token and *nask* to test the absence of a token. In our time setting, the effect of tell is additionnally a time to the token being told with the idea that the token will live for that amount of time. With respect to the other three primitives, the effect of time is to indicate the delay within which the request has to be satisfy.

Definition 2.1 Define *Stoken* as an enumerable set, the elements of which are called tokens and are typically represented by letters t and u. Define *Sduration* as the set composed of infinity, denoted ∞, and the positive integers. Elements of this set are subsequently called durations.

Definition 2.2 Define the set *Srcom* of time communication primitives as the one generated by the $\mathcal{R}$ rule of figure 1, where $t \in Stoken$ and d are durations. Moreover, for any sybset $\mathcal{X}$ of *Srcom*, define the language $\mathcal{R}(X)$ as the set of agents generated

<u>GENERAL RULES</u>

$$(S)\quad \frac{\langle A \mid \sigma\rangle \longrightarrow \langle A' \mid \sigma'\rangle}{\langle A\ ;\ B \mid \sigma\rangle \longrightarrow \langle A'\ ;\ B \mid \sigma'\rangle}$$

$$(P)\quad \frac{\langle A \mid \sigma\rangle \longrightarrow \langle A' \mid \sigma'\rangle}{\begin{array}{c}\langle A \mid\mid B \mid \sigma\rangle \longrightarrow \langle A' \mid\mid B \mid \sigma'\rangle \\ \langle B \mid\mid A \mid \sigma\rangle \longrightarrow \langle B \mid\mid A' \mid \sigma'\rangle\end{array}}$$

$$(C)\quad \frac{\langle A \mid \sigma\rangle \longrightarrow \langle A' \mid \sigma'\rangle}{\begin{array}{c}\langle A\ +\ B \mid \sigma\rangle \longrightarrow \langle A' \mid \sigma'\rangle \\ \langle B\ +\ A \mid \sigma\rangle \longrightarrow \langle A' \mid \sigma'\rangle\end{array}}$$

<u>TEMPORAL RULE</u>

$$(W)\quad \frac{A \neq E, A \neq A^- \text{ or } \sigma \neq \sigma^-, \langle A \mid \sigma\rangle \not\rightarrow}{\langle A \mid \sigma\rangle \rightsquigarrow \langle A^- \mid \sigma^-\rangle}$$

<u>DELAY RULE</u>

$$(D)\quad \langle delay(0) \mid \sigma\rangle \longrightarrow \langle E \mid \sigma\rangle$$

<u>$\mathcal{R}$ RULES</u>

$$(T0)\quad \langle tell_0(t) \mid \sigma\rangle \longrightarrow \langle E \mid \sigma\rangle$$

$$(Tr)\quad \frac{d > 0}{\langle tell_d(t) \mid \sigma\rangle \longrightarrow \langle E \mid \sigma \cup \{t_d\}\rangle}$$

$$(Ar)\quad \frac{d > 0}{\langle ask_d(t) \mid \sigma \cup \{t_k\}\rangle \longrightarrow \langle E \mid \sigma \cup \{t_k\}\rangle}$$

$$(Nr)\quad \frac{d > 0, \nexists k : t_k \in \sigma}{\langle nask_d(t) \mid \sigma\rangle \longrightarrow \langle E \mid \sigma\rangle}$$

$$(Gr)\quad \frac{d > 0}{\langle get_d(t) \mid \sigma \cup \{t_k\}\rangle \longrightarrow \langle E \mid \sigma\rangle}$$

<u>$\mathcal{M}$ RULE</u>

$$(CM0)\quad \frac{\begin{array}{c} pre^+ \subseteq \sigma^*,\ pre^- \cap \sigma^* = \emptyset, \\ \tau \subseteq \sigma,\ \tau^* = \sigma^* - post^-, d > 0\end{array}}{\langle (pre_d, post_0) \mid \sigma\rangle \longrightarrow \langle E \mid \tau\rangle}$$

$$(CM)\quad \frac{\begin{array}{c} pre^+ \subseteq \sigma^*,\ pre^- \cap \sigma^* = \emptyset, \\ \tau \subseteq \sigma,\ \tau^* = \sigma^* - post^-, d > 0 \\ \tau' = \{t_{d'} : t \in post^+\},\ d' > 0\end{array}}{\langle (pre_d, post_{d'}) \mid \sigma\rangle \longrightarrow \langle E \mid \tau \cup \tau'\rangle}$$

Fig. 2. Comparative semantics of the languages.

by the generale rules of figure 1 for $C \in \mathcal{X}$.

For any $\mathcal{X}$, computation in $\mathcal{R}(X)$ may be modelled by a transition system written in Plotkin's style. To easily express termination, we shall introduce a special terminating symbol E. For uniformity, we shall abuse language and qualify E as an agent. However, to meet the intuitive expectation, we shall always rewrite agent of the form $(E\ ;\ A)$, $(E \mid\mid A)$ and $(A \mid\mid E)$ as A. This is technicaly achieved by defining the extended set of agents as follows and by simplifying agents according to the bimonoid structure.

Definition 2.3 Define the extended set of agents *Seagent* by the following grammar

$$Ae ::= E \mid C \mid A\ ;\ A \mid A \mid\mid A \mid A\ +\ A$$

Moreover, we shall subsequently assert that the structure $(Seagent, E,\ ;\ ,\ \mid\mid\ )$ is a bimonoid and simplify elements of *Seagent* accordingly.

Definition 2.4

(i) Define the set of the stores *Ststore* as the set of the finite multisets of tokens of *Stoken* with duration as subscript.

(ii) Define the set of configurations $Sconf$ as $Seagent \times Ststore$. Configurations are denoted as $\langle A \mid \sigma \rangle$, where A is an (extended) timed agent and σ is a timed store.

When a temporal transition occures, agents and store content have to look older. This is achieved by the A^- and σ^- constructions.

Definition 2.5

(i) Given an agent $A \in \mathcal{R}(X)$, we denote by A^- the agent defined inductively as follows: [3]

$$
\begin{aligned}
tell_d(t)^- &= tell_d(t)\\
ask_d(t)^- &= ask_{max\{0,d-1\}}(t)\\
nask_d(t)^- &= nask_{max\{0,d-1\}}(t)\\
get_d(t)^- &= get_{max\{0,d-1\}}(t)\\
delay(d)^- &= delay(d-1)
\end{aligned}
\qquad
\begin{aligned}
(B \;;\; C)^- &= B^- \;;\; C\\
(B \mid\mid C)^- &= B^- \mid\mid C^-\\
(B \;+\; C)^- &= B^- \;+\; C^-
\end{aligned}
$$

(ii) Given a timed store σ, we denote by σ^- the new store obtained by decreasing the duration associated with the tokens by one unit and by removing those associated in σ with 1 unit of time: precisely, if all the notations are understood to relate to multi-sets: $\sigma^- = \{t_{d-1} : t_d \in \sigma, d > 1\}$

The operational semantics is defined by means of the transition relations $\longrightarrow$ and $\rightsquigarrow$ describing the two phase approach. These transition relations are defined by the transition rules in figure 2.

Rule (Tr) states that the primitive $tell_d(t)$ can be executed in any store σ, and that its execution results in adding the token t with duration d as subscript to the store σ. Rule (T0) states that telling a token for a zero duration succeeds without updating the store. The three others primitives are computable only for a stricly positive duration. Rules (Ar) and (Nr) state respectively that the atomic agents $ask_d(t)$ and $nask_d(t)$ can be executed in any store containing the token t and not containing t, and that their execution does not modify the current store. Rule (Gr) also states that the agent $get_d(t)$ acts similarly to the agent $ask_d(t)$ but deletes one occurrence of t from the store. Note that the symbol $\cup$ actually denotes multiset union.

In order to avoid that the computation infinitely tries to decrease blocked non-delay primitives, rule (W) requires that some progress can be made, namely that the agent A progresses, ie A^- differs from A, or that the store progresses, ie σ^- differs from σ.

The operational semantics is obtained by the integration of the two phase-relations in one relation.

[3] We extend classical arithmetic on natural numbers by $\infty - 1 = \infty$.

Definition 2.6

(i) Let δ^+ and δ^- be two fresh symbols denoting respectively success and failure. Define the set of final states $Sfstate$ as the set $Ststore \times \{\delta^+, \delta^-\}$.

(ii) Let $\mapsto$ be the relation defined by $\langle A \mid \sigma \rangle \mapsto \langle B \mid \tau \rangle$ iff $\langle A \mid \sigma \rangle \to \langle B \mid \tau \rangle$ or $\langle A \mid \sigma \rangle \rightsquigarrow \langle B \mid \tau \rangle$.

(iii) Define the *operational semantics* $\mathcal{O}_r : \mathcal{R}(Srcom) \to \mathcal{P}(Sfstate)$ as the following function: For any timed agent A,

$$\begin{aligned}\mathcal{O}_r(A) = &\{(\sigma, \delta^+) : \langle A \mid \emptyset \rangle \mapsto^* \langle E \mid \sigma \rangle\} \\ &\cup \{(\sigma, \delta^-) : \langle A \mid \emptyset \rangle \longmapsto^* \langle B \mid \sigma \rangle \not\mapsto, B \neq E\}\end{aligned}$$

2.3 The $\mathcal{M}$ family

The transition rules (Tr), (Ar), (Nr), and (Gr) suggest an alternative view of Linda-like communication primitives in terms of which conditions the current store should obey to allow the transitions to occur and which modifications these transitions make on the store.

A natural dual view of communication primitives is then to consider them as the rewriting of pre-conditions into post-conditions. We shall consequently examine, as a second family, languages based on multi-set rewriting. It is here worth noting that this approach has already been taken in [1,4,12,16].

Each communication primitive thus consists of a multi-set of pre-conditions and of a multi-set of post-conditions. Pre- and post-conditions are (possibly empty) multi-sets of positive and negative tuples. Intuitively speaking, the operational effect of a multi-set rewriting $(pre, post)$ is to insert all positive post-conditions and to delete all negative post-conditions from the current store σ, provided that σ contains all positive pre-conditions and does not contain any of the negative pre-conditions.

Time is introduced in this framework by associating *pre* and *post* sets with durations. The duration associated with the pre-condition indicates the delay within which the request has to be satisfy and the duration associated to the post-condition will be used as subscript for the tokens added to the store by the rewriting. For instance, the operational effect of the multi-set rewriting $(\{+r, -s, +t\}_2, \{+u, -t\}_3)$ is to add u_3 and delete t from the store σ provided that σ, in the current form or the one after one clock tick, contains r and t and does not contain s.

Given a multi-set rewriting $(pre_d, post_{d'})$ we shall denote by pre^+ the multi-set $\{t \mid +t \in pre\}$ and by pre^- the multi-set $\{t \mid -t \in pre\}$. The denotations $post^+$ and $post^-$ are defined analogously.

A multi-set rewriting $(pre_d, post_{d'})$ is *consistent* iff $pre^+ \cap pre^- = \emptyset$. A multi-set rewriting $(pre_d, post_{d'})$ is *valid* if $post^- \subseteq pre^+$, where $\subseteq$ denotes multi-set inclusion.

Definition 2.7 Define the set of multi-set communication primitives $Smscom$ as the set of C's engendered by the $\mathcal{M}$ rules of figure 1. On the point of notation, λ is used there to denote the empty sequence.

Given a subset $\mathcal{X}$ of $Smscom$, define the language $\mathcal{M}(\mathcal{X})$ as the set of the consistent and valid A's generated by the general rule of figure 1.

As a result of restricting to consistent and valid multi-set communication primitives, four basic pairs of pre and post-conditions are only possible: $(\{+t\}_d, \{\}_{d'})$, $(\{-t\}_d, \{\}_{d'})$, $(\{\}_d, \{+t\}_{d'})$, $(\{+t\}_d, \{-t\}_{d'})$. We shall respectively identify them to $ask_d(t)$, $nask_d(t)$, $tell_{d'}(t)$, and $get_d(t)$.

For our comparison purposes, given $\mathcal{X}$ a subset of communication primitives of $Srcom$, we shall abuse notations and denote by $\mathcal{M}(\mathcal{X})$ the language obtained by restricting multi-set rewriting pairs to component-wise multi-set unions of pairs associated with the communication primitives of $\mathcal{X}$. For instance, if $\mathcal{X} = \{ask, nask\}$, then the language $\mathcal{M}(\mathcal{X})$ only involves pairs of the form $(Pre_d, \{\}_{d'})$ where Pre may contain positive and negative tokens. Similarly, if $\mathcal{X} = \{tell, get\}$ then $\mathcal{M}(\mathcal{X})$ includes only pairs of the form $(Pre_d, Post_{d'})$ where Pre contains positive tokens only provided that each one is associated with one negative counterpart in $Post$ and $Post$ contains negative tokens provided each one is associated to one positive token in Pre as well as positive tokens (without restriction). Note that these notations fully agree with the one introduced in definition 2.7.

As in the case of the $\mathcal{R}$ family, the extention of the set of the agents with the termminating symbol E turns the structure $(\mathcal{M}(\mathcal{X}) \cup \{E\}, E, \ ; \ , \ || \ )$ into a bimonoid.

Similarly to definition 2.5, one has to define the transformation of an agent after one clock tick. This is achieved as follows.

Definition 2.8 Let A be an agent of $\mathcal{M}$. We denote by A^- the agent defined inductively as follows:

$$\begin{aligned}
(pre_d, post_{d'})^- &= (pre_{max\{0,d-1\}}, post_{d'}) \\
delay(d)^- &= delay(d-1) \\
(B \ ; \ C)^- &= B^- \ ; \ C \\
(B \ || \ C)^- &= B^- \ || \ C^- \\
(B \ + \ C)^- &= B^- \ + \ C^-
\end{aligned}$$

Pre-conditions do not care about the duration of the tokens on the store. They are only concerned with the presence or absence of some token. In order to capture this in the formalization of the rewriting transition, we introduce the following notation.

Definition 2.9 Let σ denote any store of $Ststore$. We denote by σ^* the set of the tokens of σ without their subscript.

The transition rules used in order to define the operational semantics of the $\mathcal{M}$ family of languages are provided by the general rules of figure 2 together with rules (CM) and (W) of that figure.

Rule (CM) states that a multi-set rewriting $(pre_d, post_{d'})$ can be executed in a store σ if the multi-set pre^+ is included in σ^* and if no negative pre-condition occurs in σ^*. If these conditions hold and if duration d is strictly positive, then the execution of the rewriting deletes, from σ, all the negative post-conditions, and adds, to σ, all the positive post-conditions with duration d' as subscript. The transition in rule (CM0) can be fired in similar conditions but with $d' = 0$. In this case, negative postconditions are deleted from the store but no tokens is added.

Definition 2.10 Define the *operational semantics* $\mathcal{O}_m : \mathcal{M}(Smscom) \to \mathcal{P}(Sfstate)$ as the following function: for any timed agent A,

$$\begin{aligned}\mathcal{O}_m(A) = &\{(\sigma, \delta^+) : \langle A \mid \emptyset\rangle \mapsto^* \langle E \mid \sigma\rangle\}\\ &\cup \{(\sigma, \delta^-) : \langle A \mid \emptyset\rangle \mapsto^* \langle B \mid \sigma\rangle \not\mapsto, B \neq E\}\end{aligned}$$

2.4 Normal Form

A classical result of concurrency theory is that modelling parallel composition by interleaving, as we do, allows agents to be considered in a normal form. We first define what this actually means, and then state the proposition that agents and their normal forms are equivalent in the sense that they yield the same computations.

Definition 2.11 Given a subset $\mathcal{X}$ of $Srcom$ or $Smscom$, the set $Snagent$ of agents in normal form is defined by the following rule, where N is an agent in normal form and c denotes a communication action of $\mathcal{X}$:

$$N ::= c \mid c\ ;\ N \mid N\ +\ N.$$

Proposition 2.12 *For any agent A, there is an agent N in normal form which has the same derivation sequences as A.*

3 Language comparison

3.1 Introduction

A natural question to ask is whether the timed multi-set rewriting extension we just introduced strictly increases the expressivity of languages of the $\mathcal{R}$ family and, if so, whether some of the timed primitives may be expressed in terms of others.

A basic approach to answer that question has been given by Shapiro in [21] as follows. Consider two languages L and L'. Assume given the semantics mappings (*observation criteria*) $\mathcal{S} : L \to Obs$ and $\mathcal{S}' : L' \to Obs'$, where Obs and Obs' are some suitable domains. Then, according to [21], L can *embed* L' if there exists a mapping $\mathcal{C}$ (*coder*) from the statements of L' to the statements of L, and a mapping $\mathcal{D}^e$ (*decoder*) from Obs to Obs', such that $\mathcal{D}^e(\mathcal{S}(\mathcal{C}(A))) = \mathcal{S}'(A)$, for every statement $A \in L'$. This approach is however too weak since, for instance, the above equation is satisfied by any pair of Turing-complete languages. To circumvent this problem, De Boer and Palamidessi have proposed in [8] to add three constraints on the coder $\mathcal{C}$ and on the decoder $\mathcal{D}^e$. First, $\mathcal{D}^e$ should be defined in an element-wise way w.r.t.

$$\begin{tikzcd}
\langle A \mid \emptyset \rangle \arrow[r, "*"] \arrow[d, "\mathcal{C}"'] & \langle A_1 \mid \alpha_1 \rangle \arrow[r, rightsquigarrow] & \langle A'_1 \mid \alpha'_1 \rangle \arrow[r, "*"] \arrow[d, "\mathcal{C}"'] & \ldots \arrow[r, "*"] & \langle A_n \mid \alpha_n \rangle \not\rightarrow \\
\langle B \mid \emptyset \rangle \arrow[r, "*"] & \langle B_1 \mid \beta_1 \rangle \arrow[r, rightsquigarrow] \arrow[u, "\mathcal{D}^e"'] & \langle B'_1 \mid \beta'_1 \rangle \arrow[r, "*"] & \ldots \arrow[r, "*"] & \langle B_n \mid \beta_n \rangle \not\rightarrow \arrow[u, "\mathcal{D}^e"']
\end{tikzcd}$$

Fig. 3. Phased embedding.

Obs:

$$\forall X \in Obs: \ \mathcal{D}^e(X) = \{\mathcal{D}^e{}_{el}(x) \mid x \in X\} \qquad (P_1)$$

for some appropriate mapping $\mathcal{D}^e{}_{el}$. Second, the coder $\mathcal{C}$ should be defined in a compositional way w.r.t. the sequential, parallel and choice operators: [4]

$$\begin{aligned} \mathcal{C}(A \ ; \ B) &= \mathcal{C}(A) \ ; \ \mathcal{C}(B) \\ \mathcal{C}(A \mid\mid B) &= \mathcal{C}(A) \mid\mid \mathcal{C}(B) \\ \mathcal{C}(A \ + \ B) &= \mathcal{C}(A) \ + \ \mathcal{C}(B) \end{aligned} \qquad (P_2)$$

Finally, the embedding should preserve the behavior of the original processes w.r.t. deadlock, failure and success (*termination invariance*):

$$\forall X \in Obs, \forall x \in X: \ tm'(\mathcal{D}^e{}_{el}(x)) = tm(x) \qquad (P_3)$$

where tm and tm' extract the information on termination from the observables of L and L', respectively. An embedding satisfying these properties (P_1, P_2, P_3) is said to be *modular.*

3.2 Phased embedding

In our time context, we introduce an additional requirement associated with time. Intuitively, we require that statements and their codings obey the commuting equation $\mathcal{D}^e(\mathcal{S}(\mathcal{C}(A))) = \mathcal{S}'(A)$ after each phase, thus giving rise to the situation depicted in figure 3. A modular embedding satisfying this constraint is called modular phased embedding. The formal definition is as follows. It is phrased directly in our time coordination setting.

Definition 3.1 Define the semantics $\mathcal{O}^*$ as a generalisation of the semantics $\mathcal{O}_r$ or $\mathcal{O}_m$ to arbitrary starting store but restricted to one phase: for any agent A and any store α,

$$\begin{aligned} \mathcal{O}^*(A)(\alpha) = \{&(\sigma, \delta^+) : \langle A \mid \alpha \rangle \rightarrow^* \langle E \mid \sigma \rangle\} \\ &\cup \{(\sigma, \delta^-) : \langle A \mid \alpha \rangle \rightarrow^* \langle A' \mid \sigma \rangle \not\rightarrow, A' \neq E\} \end{aligned}$$

Definition 3.2 For any agents A and B and any stores α and β, $\langle B \mid \beta \rangle$ is decodable in $\langle A \mid \alpha \rangle$ iff

[4] Actually, this is only required for the parallel and choice operators in [8].

(i) $\mathcal{C}(A) = B$ where the coder $\mathcal{C}$ is extended with $\mathcal{C}(E) = E$.

(ii) $\mathcal{D}^e{}_{el}((\beta, \delta)) = (\alpha, \delta)$ where $\delta = \begin{cases} \delta^+ \text{ if } A = E = B \\ \delta^- \text{ otherwise} \end{cases}$

Definition 3.3 Assume a coder $\mathcal{C}$ and a decoder $\mathcal{D}^e$. For any agents A, B, any store α, β, (A, α) is *phase-simulable* in (B, β) iff the following properties hold:

(i) $\langle B \mid \beta \rangle$ is decodable in $\langle A \mid \alpha \rangle$

(ii) for any agent A_1 and any store α_1 such that $\langle A \mid \alpha \rangle \to^* \langle A_1 \mid \alpha_1 \rangle \not\to$, there exist B_1 and β_1 such that $\langle B \mid \beta \rangle \to^* \langle B_1 \mid \beta_1 \rangle \not\to$ and $\langle B_1 \mid \beta_1 \rangle$ is decodable in $\langle A_1 \mid \alpha_1 \rangle$; moreover $\langle A_1 \mid \alpha_1 \rangle \rightsquigarrow \langle A'_1 \mid \alpha'_1 \rangle$ iff $\langle B_1 \mid \beta_1 \rangle \rightsquigarrow \langle B'_1 \mid \beta'_1 \rangle$ and (A'_1, α'_1) is phase-simulable in (B'_1, β'_1).

(iii) for any agent B_1 and any store β_1 such that $\langle B \mid \beta \rangle \to^* \langle B_1 \mid \beta_1 \rangle \not\to$, there exist A_1 and α_1 such that $\langle A \mid \alpha \rangle \to^* \langle A_1 \mid \alpha_1 \rangle \not\to$ and $\langle B_1 \mid \beta_1 \rangle$ is decodable in $\langle A_1 \mid \alpha_1 \rangle$; moreover $\langle A_1 \mid \alpha_1 \rangle \rightsquigarrow \langle A'_1 \mid \alpha'_1 \rangle$ iff $\langle B_1 \mid \beta_1 \rangle \rightsquigarrow \langle B'_1 \mid \beta'_1 \rangle$ and (A'_1, α'_1) is phase-simulable in (B'_1, β'_1).

Definition 3.4 [Modular phased embedding] Let L and L' be two languages of the families $\mathcal{R}$, $\mathcal{M}$, and let $\mathcal{O}_x$ and $\mathcal{O}'_x$ denote their corresponding operational semantics. The language L can embed L' in a modular and phased manner iff there exists a coder $\mathcal{C}$ (*coder*) from the statements of L' to the statements of L, and a decoder $\mathcal{D}^e$ (*decoder*) from $\mathcal{O}_x$ to $\mathcal{O}'_x$ such that properties (P_1), (P_2), (P_3) hold and such that for any agent A of L', $(A, \emptyset)$ is phase-simulable in $(\mathcal{C}(A), \emptyset)$.

The existence of a modular phased embedding from L' into L is subsequently denoted by $L' \le L$. It is easy to see that $\le$ is a pre-order relation. Moreover if $L' \subseteq L$ then $L' \le L$, that is, any language embeds all its sublanguages. This property descends immediately from the definition of modular phased embedding, by setting $\mathcal{C}$ and $\mathcal{D}^e$ equal to the identity function.

4 Intra family comparisons

4.1 The hierarchy of the languages with relative duration

The complete study of the relation between the 16 languages of the $\mathcal{R}$ family without *delay* has been operated in [14]. For the purpose of this present paper, let us just recall the following result.

Proposition 4.1
$\mathcal{R}(tell) < \mathcal{R}(ask, tell) < \mathcal{R}(ask, get, tell) < \mathcal{R}(ask, nask, get, tell)$

Let us now consider the introduction of the *delay* primitive. Obviously, the extended language embeds its sublanguage. However, this embedding is strict.

Proposition 4.2 $\mathcal{R}(ask, nask, get, tell) < \mathcal{R}(ask, nask, get, tell, delay)$

Proof.

By contradiction, assume that there is a coder $\mathcal{C}$ and decoder $\mathcal{D}^e$ provided by the definition 3.4. By the phased embedding property, the coding of the agent $delay(0)$ succeeds at time 1.

Let us now consider the agent $delay(1)$. By the phased embedding property, the coding of $delay(1)$ succeeds at time 2. It is easy to observe that the first step of any such computation corresponds to the execution of a $tell_d(t)$ or $nask_d(t)$ primitive on the empty set and thus is not a temporal step. Any computation can then be represented as follow.

$$\langle \mathcal{C}(delay(1)) \mid \emptyset \rangle_1 \rightarrow \langle C' \mid \sigma \rangle_1 \rightarrow^* \langle C'' \mid \tau \rangle_1 \rightsquigarrow \langle \mathcal{C}(delay(0)) \mid \tau^- \rangle_2 \rightarrow^* \langle E \mid \mu \rangle_2$$

where $\mathcal{D}^e((\tau, \delta^+)) = (\emptyset, \delta^+)$ and $\mathcal{D}^e((\mu, \delta^+)) = (\emptyset, \delta^+)$.

As the first step is not a temporal transition, this gives, by definition of $+$, a valid prefix for a computation of the coding of the agent $delay(0) \ + \ delay(1)$ that finishes at time 2. That contradicts the fact that, by the phased embedding property, any computation of this agent succeeds at time 1. □

4.2 The hierarchy of the languages with multiset rewriting

The first results in the expressiveness study of the $\mathcal{M}$ family come from language inclusion.

Proposition 4.3

$$\mathcal{M}(tell) \leq \mathcal{M}(ask, tell) \leq \mathcal{M}(ask, get, tell)$$
$$\leq \mathcal{M}(ask, nask, get, tell) \leq \mathcal{M}(ask, nask, get, tell, delay)$$

Proof. It is sufficient to consider the identity as coder and decoder. □

Let us now establish that these embeddings are strict.

Proposition 4.4

(i) $\mathcal{M}(ask, tell) \not\leq \mathcal{M}(ask)$

(ii) $\mathcal{M}(ask, get, tell) \not\leq \mathcal{M}(ask, tell)$

(iii) $\mathcal{M}(ask, nask, get, tell) \not\leq \mathcal{M}(ask, get, tell)$

(iv) $\mathcal{M}(ask, nask, get, tell, delay) \not\leq \mathcal{M}(ask, nask, get, tell)$

Proof.

(i). Consider the agent $(\{+a\}_1, \{\}_1)$. Its operational semantics is given by $\mathcal{O}_m((\{+a\}_1, \{\}_1)) = \{(\emptyset, \delta^-)\}$. As any agent in $\mathcal{M}(tell)$ has only successful computations, it is impossible to provide a coder and a decoder satisfying property $P3$.

(ii). Assume that $\mathcal{M}(ask, get, tell) \leq \mathcal{M}(ask, tell)$ and that there is a coder $\mathcal{C}$ and decoder $\mathcal{D}^e$ provided by the definition 3.4. Consider the

agent $(\{\}_1, \{+a\}_1)$; $(\{+a\}_1, \{-a\}_1)$. Since $\mathcal{C}$ is compositional and since $\mathcal{O}_m((\{\}_1, \{+a\}_1)$; $(\{+a\}_1, \{-a\}_1)) = \{(\emptyset, \delta^+)\}$, the termination mark of any element of $\mathcal{O}_m(\mathcal{C}((\{\}_1, \{+a\}_1))$; $\mathcal{C}((\{+a\}_1, \{-a\}_1)))$ is successful. As $\mathcal{C}((\{+a\}_1, \{-a\}_1))$ is composed of rewriting with empty negative postconditions, it does not destroy any element of the store. As all the preconditions have empty negative part, it follows that any element of $\mathcal{O}_m(\mathcal{C}((\{\}_1, \{+a\}_1))$; $\mathcal{C}((\{+a\}_1, \{-a\}_1)))$; $\mathcal{C}((\{+a\}_1, \{-a\}_1)))$ has a successful termination mark. However, $\mathcal{O}_m((\{\}_1, \{+a\}_1)$; $(\{+a\}_1, \{-a\}_1)$; $(\{+a\}_1, \{-a\}_1)) = \{(\emptyset, \delta^-)\}$, which contradicts property $P3$.

(iii). The third relation is also established by contradiction. Assume that $\mathcal{M}(ask, nask, get, tell) \leq \mathcal{M}(ask, get, tell)$ and that there is a coder $\mathcal{C}$ and decoder $\mathcal{D}^e$ provided by the definition 3.4. Let us first consider the coding of the agent $(\{\}_1, \{+a\}_1)$. As $(\{\}_1, \{+a\}_1)$ succeeds without temporal transition on the empty store, the phased embedding property ensures that any computation of its coding is of the following type

$$\langle \mathcal{C}((\{\}_1, \{+a\}_1)) \mid \emptyset \rangle_1 \rightarrow^* \langle E \mid \sigma \rangle_1$$

with $\mathcal{D}^e((\sigma, \delta^+)) = (\{a_1\}, \delta^+)$.

Now consider $\mathcal{C}((\{-a\}_1, \{\}_1))$. As it is in $\mathcal{M}(ask, get, tell)$, its normal form can be written as

$$\begin{aligned}\mathcal{C}((\{-a\}_1, \{\}_1)) = (&\{+a_1^{(1)}, \ldots, +a_{n_1}^{(1)}, +b_1^{(1)}, \ldots, +b_{m_1}^{(1)}\}_{d_1}, \\ &\quad \{+c_1^{(1)}, \ldots, +c_{l_1}^{(1)}, -b_1^{(1)}, \ldots, -b_{m_1}^{(1)}\}_{e_1}) \; ; \; A_1 \\ &+ \ldots + \\ &(\{+a_1^{(i)}, \ldots, +a_{n_i}^{(i)}, +b_1^{(i)}, \ldots, +b_{m_i}^{(i)}\}_{d_i}, \\ &\quad \{+c_1^{(i)}, \ldots, +c_{l_i}^{(i)}, -b_1^{(i)}, \ldots, -b_{m_i}^{(i)}\}_{e_i}) \; ; \; A_i\end{aligned}$$

Our first observation is that the coding can not contain any choice starting with an empty precondition. Indeed, if there is one choice starting with an empty precondition, i.e. there is some j such that $n_j = m_j = 0$, then the coding of the agent $A = (\{\}_1, \{+a\}_1)$; $((\{\}_1, \{+b\}_1) + (\{-a\}_1, \{\}_1))$ accepts the following derivation

$$\begin{aligned}\langle \mathcal{C}(A) \mid \emptyset \rangle_1 &\rightarrow^* \langle \mathcal{C}((\{\}_1, \{+b\}_1) + (\{-a\}_1, \{\}_1)) \mid \sigma \rangle_1 \\ &\rightarrow \langle A_j \mid \sigma \cup \tau \rangle_1\end{aligned}$$

where τ is the set of the tokens $c_k^{(j)} (1 \leq k \leq l_j)$ with duration e_j as subscript.

As $(\{-a\}_1, \{\}_1)$ fails on the store containing a, the agent A_j has to fail. This derivation provides then a valid prefix for a failing derivation of the agent. This contradicts, by property $P3$, the fact that A has only successful computations.

A second observation about $\mathcal{C}((\{-a\}_1, \{\}_1))$ is that, following its normal form, its computation on the empty store fails or starts with a temporal transition.

This contradicts the phased embedding property. Indeed, following $(\{-a\}_1, \{\}_1)$, $\mathcal{C}((\{-a\}_1, \{\}_1))$ has to succeed at time 1 without any temporal transition.

(iv). The proof proceeds as for proposition 4.2 but with the first observation on the agent $delay(1)$ modified as follows.

By the phased embedding property, the coding of $delay(1)$ succeeds at time 2. As it succeeds on the empty store, the first step of any such computation corresponds to the execution of a rewriting $(pre_d, post_e)$ with $d > 0$ and pre^+ and $post^-$ empty. Consequently, this first transition is not a temporal step.

□

5 Inter families comparisons

Proposition 5.1 *For any set of communication primitives $\mathcal{X}$, $\mathcal{R}(\mathcal{X}) \leq \mathcal{M}(\mathcal{X})$*

Proof. Immediate by defining the decoder as the identity and the coder as follows:

$$\begin{aligned} \mathcal{C}(tell_d(t)) &= (\{\}_1, \{+t\}_d) \\ \mathcal{C}(ask_d(t)) &= (\{+t\}_d, \{\}_1) \\ \mathcal{C}(delay(d)) &= delay(d) \end{aligned} \qquad \begin{aligned} \mathcal{C}(get_d(t)) &= (\{+t\}_d, \{-t\}_1) \\ \mathcal{C}(nask_d(t)) &= (\{-t\}_d, \{\}_1) \end{aligned}$$

□

Proposition 5.2 $\mathcal{R}(tell) \equiv \mathcal{M}(tell)$

Proof. The relation $\mathcal{R}(tell) \leq \mathcal{M}(tell)$ has already been observed. Let us now consider the converse relation. As any agent of the two families has only successful computations at time 1, it is sufficient to consider the following coding

$$\mathcal{C}((\{\}_d, \{t_1, \ldots, t_n\}_{d'})) = \ ||\ _{i=1}^{n} tell_{d'}(t_i)$$

and to use the identity as decoder. □

Proposition 5.3

(i) $\mathcal{R}(ask, tell) < \mathcal{M}(ask, tell)$

(ii) $\mathcal{R}(tell, ask, get) < \mathcal{M}(tell, ask, get)$

(iii) $\mathcal{R}(ask, nask, get, tell) < \mathcal{M}(ask, nask, get, tell)$

(iv) $\mathcal{R}(ask, nask, get, tell, delay) < \mathcal{M}(ask, nask, get, tell, delay)$

Proof. The four inclusions follow from proposition 5.1. The strict relations follow from the transitivity of $\leq$ and from the following result. □

Proposition 5.4 $\mathcal{M}(ask, tell) \not\leq \mathcal{R}(ask, nask, get, tell, delay)$

Proof. By contradiction, assume that there is a coder $\mathcal{C}$ and decoder $\mathcal{D}^e$ provided by definition 3.4.

Let us first observe that, for any token t, there are tokens $x_1, \cdots, x_m$, $y_1, \cdots, y_m$, stores $\sigma, \tau, \alpha, \alpha', \beta, \beta'$, and computations C_1, C_2, C_3, C_4 of

$$\begin{aligned}
&C = \mathcal{C}((\{\}_1, \{+x_1\}_1) \;;\; \dots \;;\; (\{\}_1, \{+x_m\}_1))\\
&C \;;\; \mathcal{C}((\{\}_1, \{+t\}_1))\\
&C \;;\; \mathcal{C}((\{\}_1, \{+y_1\}_1) \;;\; \dots \;;\; (\{\}_1, \{+y_n\}_1))\\
&C \;;\; \mathcal{C}((\{\}_1, \{+y_1\}_1) \;;\; \dots \;;\; (\{\}_1, \{+y_n\}_1) \;;\; (\{\}_1, \{+t\}_1))
\end{aligned}$$

respectively, such that

- C_2 is the continuation of C_1 by one computation C^* of $\mathcal{C}((\{\}_1, \{+t\}_1))$
- C_4 is the continuation of C_3 with C^*
- C_3 is the continuation of C_1 with one computation of the coding agent $\mathcal{C}((\{\}_1, \{+y_1\}_1) \;;\; \dots \;;\; (\{\}_1, \{+y_n\}_1))$
- if σ and τ are the store resulting from the computations C_1 and C_3, then, the computations C_2 and C_4 respectively end in the stores $\sigma \cup \alpha \setminus \beta$ and $\tau \cup \alpha' \setminus \beta'$
- $\alpha'^* = \alpha^*$ and $\beta'^* = \beta^*$.

Indeed, any computation of $(\{\}_1, \{+t\}_1))$ can be viewed as a sequence of *ask*, *nask*, *get*, *tell* and *delay* operations. Since the agent $\mathcal{C}((\{\}_1, \{+t\}_1)))$ is finite and, by property $P3$ and the phased embedding property, has to succeed at time 1, there is only a finite set of such sequences. Moreover, for any set of distincts tokens $z_1, \cdots, z_p$, any computation of

$$\mathcal{C}((\{\}_1, \{+z_1\}_1) \;;\; \dots \;;\; (\{\}_1, \{+z_p\}_1) \;;\; (\{\}_1, \{+t\}_1)))$$

which is necessarily successful by property $P3$, necessarily terminates by such a sequence, thanks to property $P2$. In these conditions, progressively increasing the set of tokens z_i neccessarily results in repeating a sequence, which establishes the claim.

To conclude, let us consider the normal form of $\mathcal{C}(X)$ for the agent $X = (\{+x_1, \dots, +x_m, +y_1, \dots, +y_n, +t\}_1, \{\}_1)$. In its most general form, this normal form is written as

$$\begin{aligned}
&tell_{d_1}(t_1) \;;\; A_1 + \cdots + tell_{d_p}(t_p) \;;\; A_p\\
&+ ask_{e_1}(u_1) \;;\; B_1 + \cdots + ask_{e_q}(u_q) \;;\; B_q\\
&+ get_{f_1}(v_1) \;;\; C_1 + \cdots + get_{f_r}(v_r) \;;\; C_r\\
&+ nask_{g_1}(w_1) \;;\; D_1 + \cdots + nask_{g_s}(w_s) \;;\; D_s\\
&+ delay(h_1) \;;\; E_1 + \cdots + delay(h_k) \;;\; E_k
\end{aligned}$$

Let us first observe that there is actually no alternative guarded by a $tell_{d_i}(t_i)$ primitive. Indeed, otherwise, the transition $\langle \mathcal{C}(X) \mid \emptyset \rangle_1 \to \langle A_j \mid \{(t_i)_{d_i}\}\rangle_1$ would be valid, which, as X has only one failing computation, can only be continued by failing computations. However, this transition then induces a failing computation for $\mathcal{C}(X) + \mathcal{C}((\{\}_1, \{+t\}_1))$, which is absurd by properties $P2$ and $P3$. Similarly, one can prove that there is no alternative guarded by a $nask_{g_i}(w_i)$ with $g_i > 0$, or by a $delay(0)$.

A second observation is that the tokens u_i's and v_j's do not appear in σ, τ and α. Formally, if $U = \{u_i : 1 \le i \le q \text{ and } e_i > 0\}$ and $V = \{v_i : 1 \le i \le r \text{ and } f_i > 0\}$, then $(U \cup V) \cap (\sigma^* \cup \tau^* \cup \alpha^*) = \emptyset$. Indeed, it is sufficient to use the same argument as that just employ for the *tell* primitives by observing the three agents

$$(\{\}_1, \{+x_1\}_1) \;;\; \ldots \;;\; (\{\}_1, \{+x_m\}_1) \;;\; X,$$
$$(\{\}_1, \{+x_1\}_1) \;;\; \ldots \;;\; (\{\}_1, \{+x_m\}_1) \;;\; (\{\}_1, \{+t\}_1) \;;\; X$$
$$(\{\}_1, \{+x_1\}_1) \;;\; \ldots \;;\; (\{\}_1, \{+x_m\}_1) \;;\; (\{\}_1, \{+y_1\}_1) \;;\; \ldots \;;\; (\{\}_1, \{+y_n\}_1) \;;\; X$$

which only one failing computation.

Let us finally consider the agent

$$\begin{aligned} Y = &(\{\}_1, \{+x_1\}_1) \;;\; \ldots \;;\; (\{\}_1, \{+x_m\}_1) \;;\\ &(\{\}_1, \{+y_1\}_1) \;;\; \ldots \;;\; (\{\}_1, \{+y_n\}_1) \;;\\ &(\{\}_1, \{+t\}_1) \;;\; X. \end{aligned}$$

The following computation prefix of its coder results from the above observations:

$$\begin{aligned} \langle \mathcal{C}(Y) \mid \emptyset \rangle_1 &\to^* \langle \mathcal{C}((\{\}_1, \{+t\}_1) \;;\; X) \mid \tau \rangle_1 \\ &\to \langle \mathcal{C}(X) \mid \sigma \cup \alpha' \setminus \beta' \rangle_1 \not\to \end{aligned}$$

To conclude, we need to analyze two cases: either $\mathcal{C}(X)^- = \mathcal{C}(X)$ or not. Both cases will lead to absurdity. On the one hand, if $\mathcal{C}(X)^- = \mathcal{C}(X)$, no temporal transition is possible and the previous computation prefix yields a failing computation for Y which is absurd by $P3$. On the other hand, if $\mathcal{C}(X)^- \ne \mathcal{C}(X)$, a temporal transition occurs which, by the phased property, contradicts the fact that any computation of Y finishes at time 1. $\square$

We finally observe the following results.

Proposition 5.5

(i) $\mathcal{R}(ask, get, tell) \not\le \mathcal{M}(ask, tell)$

(ii) $\mathcal{R}(ask, nask, get, tell) \not\le \mathcal{M}(ask, get, tell)$

(iii) $\mathcal{R}(ask, nask, get, tell, delay) \not\le \mathcal{M}(ask, nask, get, tell)$

Proof.

(i). The proof proceeds as for part (ii) of proposition 4.4 by considering the agents $tell_1(a) \;;\; get_1(a)$ and $tell_1(a) \;;\; get_1(a) \;;\; get_1(a)$.

$$\begin{tikzcd}
\begin{array}{c}\mathcal{R}(tell)\\ \mathcal{M}(tell)\end{array} \arrow[d] & \\
\mathcal{R}(ask, tell) \arrow[d] \arrow[dr] & \\
\mathcal{R}(ask, get, tell) \arrow[d] \arrow[dr] & \mathcal{M}(ask, tell) \arrow[d] \\
\mathcal{R}(ask, nask, get, tell) \arrow[d] \arrow[dr] & \mathcal{M}(ask, get, tell) \arrow[d] \\
\mathcal{R}(ask, nask, get, tell, delay) \arrow[dr] & \mathcal{M}(ask, nask, get, tell) \arrow[d] \\
& \mathcal{M}(ask, nask, get, tell, delay)
\end{tikzcd}$$

Fig. 4. families comparison

(ii). The proof proceeds as for part (iii) of proposition 4.4 by considering successively the agents $tell_1(a)$, $nask_1(a)$ and $tell_1(a)\ ;\ (tell_1(b)\ +\ nask_1(a))$.

(iii). The proof proceeds as for part (iv) of proposition 4.4.

□

6 Conclusion

This paper has studied the expressiveness of two timed extensions of coordination languages. Both are based on the two-phase functioning approach to real-time systems and incorporate relative time. The first family is based on a set of Linda like primitives while the second relies on multi-set rewriting, already employed in Gamma.

We have used the notion of modular phased embedding, introduced in our previous work, to study the expressiveness increase provided by the primitives and to compare the two families of languages. The graph on figure 4 summerizes the results we have obtained. It is worth noting that only the main results of the figure have been established in the paper. Other results are direct consequence of the transitivity of the embeddings.

On the point of graphical representation, languages on the same node of the graphs embed each other in a strong modular phased way. An arrow from a language $\mathcal{L}_1$ to a language $\mathcal{L}_2$ means that $\mathcal{L}_2$ strictly embeds $\mathcal{L}_1$ in a modular phased manner, that is $\mathcal{L}_1 \leq \mathcal{L}_2$ but that the converse modular phased embedding relation does not hold, $\mathcal{L}_2 \not\leq \mathcal{L}_1$. Excepted for the relations directly induced by the transitivity of the embeddings, the absence of arrow between two languages means that there is no modular phased embedding between these languages.

References

[1] J. Banatre and D. LeMetayer. Programming by Multiset Transformation. *Communications of the ACM*, 36(1):98–111, 1991.

[2] G. Berry and G. Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19, 1992.

[3] F.S. De Boer, M. Gabbrielli, and M.C. Meo. A Timed Concurrent Constraint Language. *Information and Computation*, 161(1):45–83, 2000.

[4] A. Brogi and J.-M. Jacquet. On the Expressiveness of Coordination via Shared Dataspaces. *Science of Computer Programming*, 46(1-2):71–98, 2003.

[5] N. Busi, R. Gorrieri, and G. Zavattaro. Process Calculi for Coordination: from Linda to JavaSpaces. In *Proc. AMAST*, Lecture Notes in Computer Science. Springer Verlag, 2000.

[6] N. Busi and G. Zavattaro. Expired Data Collection in Shared Dataspaces. *Theoretical Computer Science*, 298:529–556, 2003.

[7] P. Caspi, N. Halbwachs, P. Pilaud, and J. Plaice. Lustre: a Declarative Language for Programming Synchronous Systems. In *Proc. POPL'87*. ACM Press, 1987.

[8] F.S. de Boer and C. Palamidessi. Embedding as a Tool for Language Comparison. *Information and Computation*, 108(1):128–157, 1994.

[9] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces: Principles, Patterns, and Practice*. Addison-Wesley, 1999.

[10] D. Gelernter and N. Carriero. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2):97–107, 1992.

[11] D. Harel. Statecharts: a Visual Formalism for Complex Systems. *Science of Computer Programming*, 8, 1987.

[12] J.-M. Jacquet and L. Monteiro. Towards Resource Handling in Logic Programming: the PPL Framework and its Semantics. *Computer Language*, 22(2/3):51–77, 1996.

[13] Isabelle Linden and Jean-Marie Jacquet. On the Expressiveness of Absolute-Time Coordination Languages. In Rocco De Nicola, Gianluigi Ferrari, and Greg Meredith, editors, *Coordination Languages and Models, 6th International Conference, COORDINATION 2004, Pisa, Italy*, volume 2949 of *Lecture Notes in Computer Science*, pages 232–247. Springer, February 2004.

[14] Isabelle Linden, Jean-Marie Jacquet, Koen De Bosschere, and Antonio Brogi. On the Expressiveness of Relative-Timed Coordination Models. In *Proceedings of the International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA'03), Marseille, France, 2 september, 2003*, Electronic Notes in Computer Science, 2003.

[15] F. Maraninchi. Operational and Compositional Semantics of Synchronous Automaton Composition. In *Proc. Concur'92*, volume 630 of *Lecture Notes in Computer Science*. Springer, 1992.

[16] D. Gelernter N. Carriero and L. Zuck. Bauhaus Linda. In In P. Ciancarini, O. Nierstrasz, and A. Yonezawa, editors, *Object based models and languages for concurrent systems*, volume 924 of *Lecture Notes in Computer Science*, pages 66–76. Springer-Verlag, 1994.

[17] M. Nielsen, C. Palamidessi, and F.D. Valencia. On the Expressive Power of Temporal Concurrent Constraint Programming Languages. In *Proceedings of the 4th international ACM SIGPLAN conference on Principles and practice of declarative programming*, pages 156–167. ACM, 2002.

[18] V. Saraswat, R. Jagadeesan, and V. Gupta. Programming in Timed Concurrent Constraint Languages. In B. Mayoh, E. Tougu, and J. Penjam, editors, *Computer and System Sciences*, volume ASI-131 of *NATO*. Springer Verlag, 1994.

[19] V. Saraswat, R. Jagadeesan, and V. Gupta. Timed Default Concurrent Constraint Programming. *Journal of Symbolic Computation*, 11, 1996.

[20] V.A. Saraswat. *Concurrent Constraint Programming Languages.* The MIT Press, 1993.

[21] E.Y. Shapiro. Embeddings among Concurrent Programming Languages. In W.R. Cleaveland, editor, *Proceedings of CONCUR'92*, pages 486–503. Springer-Verlag, 1992.

[22] G. Smolka. The Oz Programming Model. In J. Van Leuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 324–343. Springer Verlag, 1995.

[23] S. Tini. On the Expressiveness of Timed Concurrent Constraint Programming. *Electronics Notes in Theoretical Computer Science*, 1999.

[24] P. Wyckoff, S.W. McLaughry, T.J. Lehman, and D.A. Ford. TSpaces. *IBM Systems Journal*, 37(3), 1998.