Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 175 (2007) 3–17
www.elsevier.com/locate/entcs

Coordination by Timers for Channel-Based Anonymous Communications

Gabriel Ciobanu[1ab] and Cristian Prisacariu[b]

[a] "A.I.Cuza" University, Faculty of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania.
[b] Romanian Academy, Institute of Computer Science
Blvd. Carol I no.8, 700505 Iaşi, Romania

Abstract

In this paper we present a new model for timed coordination of communicating distributed processes. The proposed model is an extension of the π-calculus with locations, types, and timers. Types are used to express restricted access to distributed resources. Timers define timeouts for both communication channels and resources. We define the syntax of the model and its operational semantics and provide a few results regarding the typing system and the timers. A timed barbed bisimulation relation is defined to compare the processes. Coordination is given in two stages: by strategically assigning values to timers, and then by employing a set of additional coordination rules. The timed coordination aspects are given through a *coordinator* pair. It consists of a *timers assigning function* which can be changed dynamically, and a set of coordination rules. As an illustrating example, we relate our model with the channels of the Reo coordination model.

Keywords: timers, typing system, locations, coordinator

1 Introduction

During the last two decades many coordination languages and models were developed. A comprehensive survey on coordination models and languages has been given in [21]. More recently, in [20,7], the authors survey the state of the art in coordination models for systems of agents (which has become of great interest in the last years). The coordination models are divided into two major categories: *data-driven* (Linda-like) and *control-driven* or *channel-based* (Manifold-like) models. One disadvantage of the data-driven models is that they typically lack the flexibility and control required by complex multicomponent applications, which are instead typical for control-driven models.

We consider coordination of distributed agents to be mainly a problem of message communications and time scheduling. Time is important, both for restricting

[1] The author was partially supported by CEEX Project 47/2005
[2] Email: {gabriel, cprisacariu}@iit.tuiasi.ro

doi:10.1016/j.entcs.2007.03.002

communication availability and for enforcing limited resource access (from the point of view of channel-based coordination, the communication channels represent the resources).

In [15], Hennessy and Riely introduce and study a formalism called distributed π-calculus ($D\pi$) as an extension with types and locations of the π-calculus [18]. $D\pi$ provides a theoretical framework for describing communications between distributed processes with *restricted resource access*.

We adopt $D\pi$ (thus we are able to model communications restricted by types) and introduce *timers* over channel names in order to define timeouts for communications. We also attach timers to channel types in order to restrict their existence inside the type environment of the process. All these timers are decreasing. The channel is discarded whenever a channel timer reaches value 1 (no communication is possible anymore on the channel, and the system advances to another state). Similarly, a channel type is lost whenever its attached timer expires. The new calculus is called *timed distributed π-calculus* ($tD\pi$), and it is introduced in [10]. Over this formalism we define a timed coordination by assigning specific values to timers and defining a constant set of coordination rules. The model gives us a strong formal ground for designing coordinated systems; the coordination language would come as a linguistic embodiment of the model.

The time aspects of process algebras is an intense studied topic. An extension of the π-calculus with a timer construct is introduced and studied in [6]. Process algebras based on a discrete time domain are presented in [5], and a variant of timed bisimulation is given in [24].

The coordination languages most related to our work are Manifold [4] together with its timed extension in [17], and Reo [2]. Manifold is based on the Ideal Worker Ideal Manager (IWIM) model [1] and has basically two kinds of processes; manager and worker. The manager coordinates the workers and the communications among them. The workers are computational processes which are not aware of who needs the results of their "work" or to whom they communicate to. Manifold is *event-driven*: managers wait for some specific event to trigger some actions; these actions determine the manager to change its state. Other models for channel-based coordination languages, also based on the π-calculus, are MoCha-π [14] and $\sigma\pi$ [3] which we describe more in details in conclusion.

Linda [9] is a popular data-driven coordination language which uses a data-space of tuples. It makes a clear separation between the computational part and the coordination part of an application. In [8] the authors use process algebra to express the coordination primitives of Linda, and present several results regarding observational equivalences. μKlaim [12] has been recently proposed as the model of KLAIM (data-driven) coordination language.

Time has been investigated also for data-driven coordination models. A form of timeouts can be expressed in JavaSpaces or TSpaces. In timed Linda [11] a global clock is considered and the basic actions (*in*, *out* or *rd*) take no time to execute, i.e., one time unit means the execution of such an (atomic) action. A new action is introduced which makes possible to wait for a tuple only for a predetermined

number of time units; the process changes its state after this number of time units. Several extensions of Linda with different notions of time are introduced in [16]. Most recently, an extension with time of the tuple-centre based model ReSpecT is presented in [22].

The coordination model introduced over $tD\pi$ is included in the large class of control-driven models. The triggering events in $tD\pi$ are either the communications on channels, or the migration with *go*, or the expiration of a timer. A $tD\pi$ process is *eager* in the sense that it needs to make as much communications as possible. Using timers in the process of coordination, $tD\pi$ goes beyond the coordination only by messages transmitted among processes, or by restricting the actions permitted on channels using channel types. Timers and the time constraints they impose provide temporal synchronisation.

In $tD\pi$ we have exogenous coordination. A process (the manager) sends to the other processes messages containing permissions (as types) for certain channels and also timer values for those channels (defining the time constraints). With this behaviour, an outside process dictates to other processes what they are allowed to do. Note that in our model any process may become a manager. We abstract away from the many details of real agent systems, and focus only on the basic features relevant for a coordination model. We consider as main action the communication on located channels. In a distributed environment a process may also move (by *go* actions) from one location to another. The timers and types are used to restrict and coordinate these actions.

In Section 2 we briefly introduce the syntax and semantics of the timed distributed π-calculus. A detailed presentation can be found in [10]. In Section 2.3 we give some technical results expressing that the typing system and typing rules are sound with respect to the dynamic semantics given by reduction rules and equivalence relation. A timed barbed bisimulation relation is given. The coordination part is treated in Section 3. Simple examples related to Reo model are given in Section 3.1. We conclude the paper by comparing our coordination model with three other recently proposed models.

2 Timed Distributed π-calculus

2.1 Syntax

In $tD\pi$, waiting for a communication on a channel is no longer indefinite (like in $D\pi$); if no communication happens in a predefined interval of time, the waiting process goes to another state. This approach leads to a method of sharing the channels in time. The timer Δt of each channel makes the channel available for communication only for the period of time determined by the discrete value t. We consider timers for both input and output channels. The reason for adding timers to outputs comes from the fact that in distributed systems we have both multiple clients and multiple servers. This means that clients may switch from one server to another depending on the waiting time. To simplify our presentation we choose a simpler π-calculus and omit the syntax for *matching* or *summation*. A communication channel is considered

a fixed resource at a location.

The syntax of *Input* and *Output* communication uses a pair of processes. For instance, an *Input* expression $a^{\Delta t}?(X:T).(P,Q)$ evolves to P whenever a communication is established during the interval of time given by Δt; otherwise it evolves to Q. The variable X is considered bound only in P and we should provide its type T; the types are presented in Table 2.

Table 1: *Syntax of tDπ*

$u ::= x$	Variable Name	$P, Q ::= stop$	Termination
$\mid a^{\Delta t}$	Timed Channel	$\mid P \mid Q$	Composition
$l ::= x$	Variable Name	$\mid (\nu\, u\colon A)P$	Channel Restriction
$\mid k$	Location Name	$\mid go\, l.(P,Q)$	Movement
$v ::= bv$	Base Value	$\mid u!\langle v\rangle.(P,Q)$	Output
$\mid u \mid l$	Name	$\mid u?(X:T).(P,Q)$	Input
$\mid u@l$	Located Name	$\mid {}^*P$	Replication
$\mid (v_1,..,v_n)$	Tuple of Values	$M, N ::= M \mid N$	Composition
$X ::= x$	Variable	$\mid (\nu\, u@l : T)N$	Located Restriction
$\mid X@l$	Located Variable	$\mid l[[P]]_\Gamma$	Located Process
$\mid (X_1,..,X_n)$	Tuple of Variables		

Two channels are equal $a_1^{\Delta t_1} = a_2^{\Delta t_2}$ if and only if $a_1 = a_2$ and $t_1 = t_2$. Waiting indefinitely on a channel a is allowed by considering Δt as ∞. For example, an output process defined by the expression $a^{\infty}!\langle v\rangle.(P,Q)$ awaits forever to send the value v, simulating the behaviour of an output process in untimed π-calculus. In the expression below, two processes are running in parallel and can interact along the common channel a:

$$a^{\Delta t}!\langle v\rangle.(P,Q) \mid a^{\Delta t'}?(X:T).(P',Q') \longrightarrow P \mid P'\{{}^{v}/_{X}\}$$

We label each located process with a type environment Γ which is a set of *location types*. The purpose of the type environment associated with a specific process is to restrict the range of accessible resources the process can access. Formally, $\Gamma \subseteq \mathcal{L} \times K$ is a relation associating to a location name a location type. A location type is a set of *location capabilities* which may contain *channel types*, *move* capability (i.e., permission to migrate to that location), or *channel creation* capability (i.e., permission to create channels). A channel type may contain the channel capabilities *read* (r), *write* (w), and *read only* (ro). A process which has a channel type $res\{r\langle T\rangle, w\langle T'\rangle, ro\langle T''\rangle\}$ can receive messages of type T and send messages of type T'. The ro capability behaves as r with the difference that the types of the received messages are not added to the type environment of the process. A type environment increases (new types are added) when a name is received along a $r\langle\,\rangle$ channel. With $ro\langle\,\rangle$ capability we describe processes which may use a received channel only if their type environment has a corresponding channel type.

In Dπ the resources are accumulated, but they can never be lost (discarded). We extend the channel types of $D\pi$ with timers of the form Δt. Communication is now permitted on channels only in the interval of time given by the timer value t (i.e. until the timer of the channel type expires). These timers define the existence of the channel types inside the type environment. Timers decrease with each "tick" of an universal clock (we assume that we have an universal clock). Upon expiration, the

channel types are discarded. Timers are created once with the channel types, and are activated when the types are added to the type environment.

Table 2: *Type system and subtyping relation*

Types:	Subtyping:	
K ::= loc$\{\tilde{\kappa}\}$	$\kappa <: \kappa$	
A ::= res$\{\tilde{\alpha}\}\Delta t$	$a : A <: a : B$	if $A <: B$
E ::= A | K | $\mathcal{B}$	$K <: L$	if $\forall \lambda \in L$: $\exists \kappa \in K$: $\kappa <: \lambda$
T ::= E | $(T_1, .., T_n)$	$A <: B$	if $\forall \beta \in B$: $\exists \alpha \in A$: $\alpha <: \beta$
| $A_1, .., A_n@K$	$\tilde{A}@K <: \tilde{B}@L$	if K<:L and $\tilde{A}<:\tilde{B}$
Capabilities:	$r\langle T\rangle <: r\langle T'\rangle$	if $T <: T'$
$\kappa ::= a : A \mid go \mid new\, c$	$w\langle S\rangle <: w\langle S'\rangle$	if $S' <: S$
$\alpha ::= r\langle T\rangle \mid w\langle T\rangle \mid ro\langle T\rangle$	$ro\langle T\rangle <: ro\langle T'\rangle$	if $T <: T'$

$\mathcal{B}$ represents a set of *base types* (Integer, Boolean, etc.). The subtyping relation $<:$ is similar to the subtyping relation of $D\pi$, excepting the new type $ro\langle\,\rangle$. Note that the intuitive behaviour of the subtyping relation is the inverse of the inclusion of sets ($A <: B$ for types means $A \supset B$ for sets). Usually, a process moves to a location (by performing a *go* action), and waits for a period of time to establish a communication on a particular channel. The capabilities $r/w/ro$ for the fixed resources tell a process what is it allowed to do when it reaches a location.

When the processes receive new channel names, types for the new channels become available. It means that the processes can communicate on the new channels according to the new types. For example, if a process receives through an input channel a located name $a@k$, then it gains the capability to move to location k, and to communicate on channel a. A process which has a channel type with the capability $r\langle T\rangle$ can receive (without generating errors) only messages of type T; the error system is presented in Table 3. When the channel type $res\{r\langle T\rangle\}$ is extended with $r\langle T'\rangle$, it follows naturally that the process is able now to receive messages of a richer type: T and T'.

Contrary to the case for channel names, the equality between channel types does not depend on their associated timers. The equality must be tested only with the names and the capabilities, and it keeps the old timers.

We define a function ψ which affects only the set of capabilities. It decreases the timers of the channel types and removes the types with an expired timer. By removing channel types, it is possible to get location types with only *go* capability (we call them *empty locations*). A process can move to an empty location, but there it does not have the capability to perform any action, and consequently produces a *runtime error*. Thus ψ removes also the empty locations.

Definition 2.1 (Cleanup function)
$\psi : \mathcal{P}_\Delta \to \mathcal{P}_\Delta$ is defined over the set $\mathcal{P}_\Delta$ of tagged located processes such that

$$\psi(l[[P]]_\Gamma) = l[[P]]_{\Gamma'}$$

where l can be any location in the distributed system and Γ' is obtained from Γ where every type $c : res\,\langle\,\rangle\,\Delta t,\ t > 1, t \neq \infty$ is changed to $c : res\,\langle\,\rangle\,\Delta(t-1)$, and every $c : res\,\langle\,\rangle\,\Delta 1$ disappears. Location types of form $loc : \{go\}$ are removed.

2.2 Semantics

The passage of time is formalised by a *time-stepping function* ϕ_Δ defined over the set $\mathcal{P}_\Delta$ of tagged located processes. The possible communications are performed at every tick of the universal clock. Active channels are those that could be involved in these communications. ϕ_Δ affects the active channels which do not communicate at the tick of the universal clock (the channels involved in communication disappear together with their timers). Due to timers, the capabilities can be lost, which leads to "errors". We define ϕ_Δ to check the existence of the needed types and change the process accordingly. As ϕ_Δ decreases the channel timers we also extend it to take care of the type environments by applying the cleanup function ψ. In the definition of ϕ_Δ we omit the channel type and the transmitted message in the input and output processes for brevity. For the $go\,k$ syntax if the location type contains the capability go, then R is executed; if k is not defined in Γ, then Q is executed. If go is not present, the process is considered to do something against its permissions and an error is generated.

Well-typedness of processes is defined by a set of static rules (a detailed presentation of the static typing rules is given in [10]). These rules express the behaviour of a process with regard to its types. If a process wants to communicate on a channel for which it has no capability, it can still be well-typed if the alternative process Q is well-typed. Q is called the *safety process*.

Definition 2.2 (Tagged time-stepping function)
We define $\phi_\Delta : \mathcal{P}_\Delta \to \mathcal{P}_\Delta$, where Γ' is obtained by application of the cleanup function ψ. Note that we use a concise notation $a^{\Delta t}.(R,Q)$ to stand for both $a^{\Delta t}!\langle v\rangle.(R,Q)$ and $a^{\Delta t}?(X:T).(R,Q)$.

$$\phi_\Delta(l[[P]]_\Gamma) = \begin{cases} k[[R]]_{\Gamma'} & \text{if } P = go\,k.(R,Q) \text{ and } \Gamma(k) <: loc\{go\} \\ l[[Q]]_{\Gamma'} & \text{if } P = go\,k.(R,Q) \text{ and } k \notin dom(\Gamma) \\ l[[a^{\Delta(t-1)}.(R,Q)]]_{\Gamma'} & \text{if } P = a^{\Delta t}.(R,Q),\ \mathrm{t}{>}1 \text{ and } \mathrm{t} \neq \infty \\ l[[Q]]_{\Gamma'} & \text{if } P = a^{\Delta t}.(R,Q),\ t \leq 1 \\ l[[Q]]_{\Gamma'} & \text{if } P = a^{\Delta t}.(R,Q),\ \mathrm{t}{>}1 \text{ and } \Gamma \not<: \Gamma(l,a) \\ \phi_\Delta(l[[R]]_\Gamma) \mid \phi_\Delta(l[[Q]]_\Gamma) & \text{if } P = R \mid Q \\ (\nu a@l{:}A)\phi_\Delta(l[[R]]_{\Gamma\{a@l:A\}}) & \text{if } P = (\nu a : A)R \\ l[[P]]_{\Gamma'} & \text{otherwise} \end{cases}$$

We write $\Gamma \vdash P$ and say that *process P is well-typed with respect to type environment* Γ; we also write $\Gamma \vdash_k P$ and say that *P is well-typed to run at location k*. To say that $P = a^{\Delta t}!\langle v\rangle.(R,Q)$ is well-typed to run at location k, with respect to type environment Γ, the following statements should hold: *(i)* $\Gamma \vdash_k v : T$ which means that v is a well-formed value at location k of type T; *(ii)* $\Gamma \vdash_k a : res\{w\langle T\rangle\}\Delta t$ which means that channel a exists at location k and may communicate values of type T for another t units of time; *(iii)* $\Gamma \vdash_k R$; $\Gamma \vdash_k Q$ which means that R and Q are well-typed at location k.

Since the function ψ changes the capability set Γ by removing channel and location types, we are interested if the process is still well-typed under the new Γ'. The following lemma relates the typing environment of the processes with the passage of time. For complete proofs see [10].

Lemma 2.3 *(Well-typedness is preserved by the cleanup function)*
If $\Gamma \Vdash l[[P]]_\Delta$ *then* $\Gamma \Vdash \psi(l[[P]]_\Delta)$.

We consider the tagged located processes ranged over by N and M (e.g., N represents $l[[P]]_\Gamma$). We denote by $\not\rightarrow$ the fact that rules (R_Γ-COM1) and (R_Γ-COM2) cannot be applied. Using these notations, we give the following reduction rules providing a dynamic semantics for $tD\pi$.

$$(\mathrm{R}_\Gamma\text{-IDLE})\ \frac{l[[P]]_\Gamma \not\rightarrow}{l[[P]]_\Gamma \rightarrow \phi_\Delta(l[[P]]_\Gamma)}$$

$$(\mathrm{R}_\Gamma\text{-COM1})\ \frac{\Gamma(l,a) <: res\{r\langle T\rangle\}}{\begin{array}{c} l[[a^{\Delta t}!\langle v\rangle.(P,Q)]]_\Delta \mid l[[a^{\Delta t'}?(X:T).(P',Q')]]_\Gamma \rightarrow \\ \psi(l[[P]]_\Delta) \mid \psi(l[[P'\{^v/_X\}]]_{\Gamma \sqcap \{v@l:T\}}) \end{array}}$$

$$(\mathrm{R}_\Gamma\text{-COM2})\ \frac{\Gamma(l,a) <: res\{ro\langle T\rangle\}}{\begin{array}{c} l[[a^{\Delta t}!\langle v\rangle.(P,Q)]]_\Delta \mid l[[a^{\Delta t'}?(X:T).(P',Q')]]_\Gamma \rightarrow \\ \psi(l[[P]]_\Delta) \mid \psi(l[[P'\{^v/_X\}]]_\Gamma) \end{array}}$$

$$(\mathrm{R}_\Gamma\text{-PAR})\ \frac{N \rightarrow N' \quad M \rightarrow M'}{N \mid M \rightarrow N' \mid M'} \qquad (\mathrm{R}_\Gamma\text{-RES})\ \frac{N \rightarrow N'}{(\nu a@l:T)N \rightarrow (\nu a@l:T)N'}$$

$$(\mathrm{R}_\Gamma\text{-CONG})\ \frac{N \equiv N' \quad N \rightarrow M \quad M \equiv M'}{N' \rightarrow M'}$$

We have two communication rules which depend on the type of the channel. In (R_Γ-COM2) we consider $ro\langle\rangle$ channels, and the process may use the received information without adding the new type to its type environment Γ, as the case in rule (R_Γ-COM1). In these cases the type environments are affected by the cleanup function ψ. In (R_Γ-IDLE) the function ϕ_Δ decreases the timers on channels, and for the expired timers the function discards the channels. Because the movement syntax enters under the application of function ϕ_Δ, we have no (R_Γ-GO) rule. At each tick of the universal clock (R_Γ-IDLE) is applied to *go* processes and to processes which do not enter any communication. In rule (R_Γ-PAR) a process M reduces to M' by (R_Γ-IDLE) rule if it has no internal communication reductions.

In the following table we give the error system of $tD\pi$ where by $\xrightarrow{err}$ we denote the generation of an error.

Table 3: *Runtime errors*

$$(\text{E-GO})\ \frac{\Gamma(k) \text{ is defined and } \Gamma(k) \not<: loc\{go\}}{l[[go\,k.(P,Q)]]_\Gamma \xrightarrow{err}} \qquad (\text{E-SUBC})\ \frac{\Gamma(l) \not<: loc\{new\,c\}}{l[[(\nu a:A)P]]_\Gamma \xrightarrow{err}}$$

$$(\text{E-SND})\ \frac{\Gamma(l,a) \text{ is defined and } \Gamma(l,v) \not<: wobj(\Gamma(l,a))}{l[[a^{\Delta t}!\langle v\rangle.(P,Q)]]_\Gamma \xrightarrow{err}}$$

$$(\text{E-RCV})\ \frac{\Gamma(l,a) \text{ is defined and } robj(\Gamma(l,a)) \not<: T \text{ or } roobj(\Gamma(l,a)) \not<: T}{l[[a^{\Delta t}?(X:T).(P,Q)]]_\Gamma \xrightarrow{err}}$$

$$(\text{E-COM})\ \frac{\begin{array}{c}\Gamma(l,a) \text{ and } \Delta(l,a) \text{ are defined and} \\ wobj(\Gamma(l,a)) \not<: robj(\Delta(l,a)) \text{ or } wobj(\Gamma(l,a)) \not<: roobj(\Delta(l,a))\end{array}}{l[[a^{\Delta t}!\langle v\rangle.(P,Q)]]_\Gamma \mid l[[a^{\Delta t'}?(X:T).(P,Q)]]_\Delta \xrightarrow{err}}$$

$$(\text{E-NEW})\ \frac{N \xrightarrow{err}}{(\nu a@k:T)N \xrightarrow{err}} \qquad (\text{E-PAR})\ \frac{N \xrightarrow{err}}{N \mid M \xrightarrow{err}} \qquad (\text{E-STR})\ \frac{M \equiv N \quad N \xrightarrow{err}}{M \xrightarrow{err}}$$

robj(), roobj(), wobj() are partial functions defined over the set of channel types, and return the transmitted type. For example, considering in type environment Γ

at location l a channel type $a : res\{r\langle T\rangle, w\langle T'\rangle\}$, the application of $wobj(\Gamma(l,a))$ returns T'. A runtime error is possible only when the channel or location type **is** in the type environment (when a process tries to do something against the types accumulated in its type environment). When a type is not in the type environment of the process, the safety process is chosen by ϕ_Δ function.

2.3 Some Technical Results

Soundness: We follow a method introduced in [13]. This is a syntactic approach in contrast to other approaches based on denotational or structural operational semantics developed by authors like Abadi, Cardelli or Milner. We omit the proofs which can be found in [10].

Lemma 2.4 *If* $\Gamma \Vdash l[[P]]_\Delta$ *then* $\Gamma \Vdash \phi_\Delta(l[[P]]_\Delta)$.

Proof: This lemma claims that the passage of time does not interfere with the typing system. ϕ_Δ does not change the property of a process of being well-typed under some Γ. To prove this we consider the form of P from the definition of ϕ_Δ. We use induction on the depth of the inference tree. □

Theorem 2.5 *(Subject reduction)*
For all tagged located processes

(a) If $N \equiv N'$ *then* $\Gamma \Vdash N$ *if and only if* $\Gamma \Vdash N'$.

(b) If $N \to N'$ *then* $\Gamma \Vdash N$ *if and only if* $\Gamma \Vdash N'$.

Proof: For the first part of the theorem, the proof proceeds by considering all the equivalence equations (the equivalences equations [10] are in the spirit of the equations given by Milner for the π-calculus). The part *(b)* of the theorem asserts the consistency between the static semantics (the typing rules) and the dynamic semantics (the reduction rules). We proceed by induction on the depth of inference tree given by $N \to N'$. We also use Lemma 2.3 which relates time and type environments, and Lemma 2.4 which relates time and channel names. □

Subject reduction assures us that once well-typed, a process remains well-typed. Note that contrary to the general approach in functional programming, in $tD\pi$ well-typedness must be preserved also by the structural equivalence relation. In the following we give a result of *type safety* which is needed to get a complete proof of the soundness property of $tD\pi$. This result states that if a system is well-typed, then it cannot give rise to runtime errors, and this is denoted by $P \overset{err}{\not\longrightarrow}$.

Theorem 2.6 *For all tagged located processes* N *and all type environments* Γ *such that* $\Gamma \Vdash N$ *we have* $N \overset{err}{\not\longrightarrow}$.

Proof: The proof considers the contrapositive of the theorem which states that *if* N *gives rise to a runtime error* ($N \overset{err}{\longrightarrow}$), *then* N *cannot be well-typed under any type environment* Γ ($\Gamma \not\Vdash N$, $\forall\Gamma$). We use induction on the structure of N and

consider a proof cases for each rule in the definition of the runtime errors of Table 3. □

Timed barbed bisimulation: When the operational semantics is defined by a reduction relation (i.e., no labels over transitions), *barbed bisimulation* helps to compare the evolution of two systems. Two systems are equivalent if an observer cannot distinguish differences in their behaviour. Following the presentation of barbed bisimulation in [19], we specify first what is observable, and what is unobservable. To simplify the presentation we choose as observable only the communication along the located channel names, without considering the transmitted messages. In $tD\pi$ we have synchronous communication on fixed located channels. In consequence, the observables can be both *input* and *output* communications. We consider as unobservable the *movement* with *go*, the *application of the time-stepping function* ϕ_Δ, and the *internal interaction* of processes. Intuitively an observer of process P is a process O which runs in parallel with P.

There are mainly four observation coordinates in $tD\pi$: one involves the name of the communication channel (Milner and Sangiorgi's barbed bisimulation), another is given by the locations, third is given by the type environment, and forth is given by time. Abstract observables (or barbs) are unary predicates over processes. Barbs are sometimes called commitment predicates and define the possibility of a process to immediately communicate on a specific channel. A natural question arises: how powerful an observer should be? We consider that an *untimed observer* cannot distinguish the values of the timers. On the other hand, a *timed observer* monitors the timers of the channel types inside the type environment, or/and the timers of the channel names.

We define a class of barbs which observe both timers on channel names and on channel types. Furthermore, they also observe the location of the communication channel. The barbs are restricted to a type environment Δ; denoting the observers distinguishing power over types. The barb $\downarrow^{t,t'\Delta}_{\mu@k}$ identifies processes which have enough permissions for the channel $\underline{\mu}$ with respect to the type environment Δ.

Definition 2.7 A *timed global typed barb predicate*, $\downarrow^{t,t'\Delta}_{\mu@k}$ where $\mu \in \{a?, a!\}$ with a being any channel name, is defined inductively by the following system of rules. We denote by $\underline{\mu}$ the names of the input or output channels. If $\mu = a?$ then $\underline{\mu} = a$.

$$\frac{\Gamma(k,a) <: \Delta(k,a) \qquad \Gamma(k,a) = res\{\ldots\}\Delta t'}{k[[a^{\Delta t}!\langle v\rangle.(P,R)]]_\Gamma \downarrow^{t,t'\Delta}_{a!@k}} \qquad \frac{\Gamma(k,a) <: \Delta(k,a) \qquad \Gamma(k,a) = res\{\ldots\}\Delta t'}{k[[a^{\Delta t}?(X:T).(P,R)]]_\Gamma \downarrow^{t,t'\Delta}_{a?@k}}$$

$$\frac{N \downarrow^{t,t'\Delta}_{\mu@k}}{N \mid M \downarrow^{t,t'\Delta}_{\mu@k}} \qquad \frac{N \downarrow^{t,t'\Delta}_{\mu@k} \quad \text{and} \quad a \neq \underline{\mu}}{(\nu\, a@l : A)N \downarrow^{t,t'\Delta}_{\mu@k}} \qquad \frac{k[[P]] \downarrow^{t,t'\Delta}_{\mu@k}}{k[[{}^*P]] \downarrow^{t,t'\Delta}_{\mu@k}}$$

Definition 2.8 A *timed global typed barbed bisimulation* $\mathcal{S}$ is a symmetric binary relation over processes which for each $(P,Q) \in \mathcal{S}$ implies

(1) if $P \downarrow^{t,t'\Delta}_{\mu@k}$ then $Q \downarrow^{t,t'\Delta}_{\mu@k}$ for any barb $\downarrow^{t,t'\Delta}_{\mu@k}$;
(2) if $P \to P'$ then $Q \to Q'$ and $(P', Q') \in \mathcal{S}$.

Two processes are timed global typed barbed bisimilar, denoted $P \dot{\sim}_{tGTB} Q$, if and only if $(P, Q) \in \mathcal{S}$ for some timed global typed barbed bisimulation $\mathcal{S}$.

The barbed bisimulation by itself does not offer satisfactory properties. In order to obtain a barbed equivalence (barbed congruence) the bisimulation is closed under all static (respectively normal) contexts [23]. Equivalently, we can close the barbed bisimulation under all observers well-typed with respect to the type environment Δ. Thus N and M are *timed global typed barbed equivalent* ($N \sim_{tGTB} M$) if and only if for all $\Delta \Vdash O$ we have $N \,|\, O \dot{\sim}_{tGTB} M \,|\, O$.

3 Coordination part

Since $tD\pi$ uses distributed resources and code migration we find easily similarities with the *channel-based* coordination of Reo. The distributed resources are the fixed located channels attached at a single location. In order to be able to communicate messages on a particular channel a process must migrate to the location of the channel. At each moment in time there can be only one process at each end of a channel. The communication is anonymous as each process does not care to whom it sends or from whom it receives the message. It only cares for sending the required message immediately as another complementary process tries to communicate at the other end of the located channel. The time to wait to achieve the communication is not indefinite as in other approaches. $tD\pi$ offers the possibility to define a deadline timer (or time-window) which defines how long a process is allowed to wait on a channel.

A classical coordination model should clearly define the set of *entities to be coordinated*, the *media* used to coordinate the entities (the coordination architecture), and the rules of the *coordination protocol*. The same separation is defined in $tD\pi$.

- *Coordination entities* are the distributed mobile communicating processes;
- *Coordination media* is composed by a *timers assigning function* $\mathcal{TA}$ (see the definition below) together with the located communication channels and the types;
- *Coordination laws* are given by the static and dynamic semantics of $tD\pi$ (i.e., reduction and typing rules, and the time-stepping and cleanup functions), together with a set of coordination rules $\mathcal{CP}$.

$tD\pi$ is designed to model a distributed architecture (it has features as mobility and locations). However, $tD\pi$ can also model a single platform architecture by restricting the system to only one location. Other requirements of a coordination model refer to the separation of coordination part from the computation part. In our model $tD\pi$ represents the computation part. The coordination part is given by a coordinating pair. The first component ($\mathcal{TA}$) is a function assigning values to the timers, and the second component ($\mathcal{CP}$) is a coordination protocol given by a set of rules (known generally).

Among all the possible reductions (traces) of a $tD\pi$ process expression we can select a subset by imposing certain values for the timers. We define a function which assigns values from $\mathbb{N}$ to the timers Δt.

Definition 3.1 (Assigning values to timers)
We denote by ΔT the set of timers on channels together with the timers on types. An assigning of natural values to the timers in ΔT is done through a function $\mathcal{TA} : \Delta T \to \mathbb{N} \cup \{\infty\}$. We denote by $\Delta T|_P$ the set of timers specific to process P; considering a process P, $\mathcal{TA}$ is restricted naturally to $\mathcal{TA}|_P$.

In an arbitrary process $P = a^{\Delta t}!\langle v\rangle.(R, Q)$, the timer can take two special values: ∞ and 1. If the value is ∞, the process can wait on channel a forever. If the value is 1, we have no communication on a and the process reduces to the second alternative process Q (in this case we call P a transitory process).

A coordination protocol $\mathcal{CP}$ is given by a set of rules providing some conditions when we have more than one communication choice in a reduction step. An example is the choice among the processes which can send or receive messages along a common channel named a. A coordination rule can be defined to select the sender or receiver which has the lowest channel timer value; such a rule could be extended to type timers too.

Definition 3.2 (Coordinator)
The coordination part is expressed in $tD\pi$ as a pair $\mathcal{C} = (\mathcal{TA}, \mathcal{CP})$ called *coordinator*.

We extend the timed bisimulation in Definition 2.8 to incorporate the coordination part. Two processes P and Q are *bisimilar with respect to a coordinator* $\mathcal{C}$, denote by $P \sim^{\mathcal{C}}_{tGTB} Q$, if $P \sim_{tGTB} Q$ and both the timers of P and Q are controlled by the same coordinator $\mathcal{C}$.

An equivalence relation can be defined over the timers assigning functions as follows. First we need a mapping $(\cdot,\cdot) : \mathcal{P}_\Delta \times \mathcal{TA} \to \mathcal{P}_\Delta$ which changes the timers of a tagged located process according to a timers assigning function. Given a protocol $\mathcal{CP}$, we say that two functions $\mathcal{TA}_1$ and $\mathcal{TA}_2$ are equivalent if they do not change the related behaviours of any two systems; i.e., $\forall N, M \in \mathcal{P}_\Delta$, $(N, \mathcal{TA}_1) \sim_{tGTB} (M, \mathcal{TA}_1) \Leftrightarrow (N, \mathcal{TA}_2) \sim_{tGTB} (M, \mathcal{TA}_2)$. An example of two equivalent functions $\mathcal{TA}_1$ and $\mathcal{TA}_2$ is given by a simple constant translation; i.e., $\mathcal{TA}_2(\Delta t) = \mathcal{TA}_1(\Delta t) + c$, where c is a constant. This equivalence relation over timers assigning functions can be extended to coordinators: *two coordinators $\mathcal{C}_1, \mathcal{C}_2$ are equivalent, denoted $\mathcal{C}_1 \sim \mathcal{C}_2$, if they have the same set $\mathcal{CP}$ of rules, and the corresponding timers assigning functions are equivalent.* The timed bisimilar relation with respect to a coordinator ($\sim^{\mathcal{C}}_{tGTB}$) and the equivalence relation $\sim$ over coordinators are related by the following result:

Theorem 3.3 *For every tagged located processes N, M and coordinators $\mathcal{C}_1, \mathcal{C}_2$, if $N \sim^{\mathcal{C}_1}_{tGTB} M$ and $\mathcal{C}_1 \sim \mathcal{C}_2$, then $N \sim^{\mathcal{C}_2}_{tGTB} M$.*

The new defined bisimulation emphasises the role of timers assigning functions

in coordination. The set $\mathcal{CP}$ of rules becomes more important in coordination languages, when the algorithmic aspects are predominant in the strategies of controlling the computation. The aspects described by $\mathcal{CP}$ are visible at the implementation of a coordination language according to its coordination model.

3.1 Exemplifying coordination and modelling power

In this section we give some simple examples of $tD\pi$ processes relating them to the *connectors* of Reo model [2]. Reo is a channel-based exogenous coordination model based on complex connectors made up of simple channels. Connectors impose specific coordination patterns based on anonymous communication between entities through these connector mechanisms.

A channel is made up of two ends (source a and sink b) and denoted ab. At each end there can be at most one entity connected at any time. Channels and entities are considered mobile. On the source end it can be written messages and on the sink end the messages can be read. There are a set of operations on channels like creation, or movement of one end to another location, or read according to a read pattern. A take operation also removes the message from the channel, as opposed to a read operation.

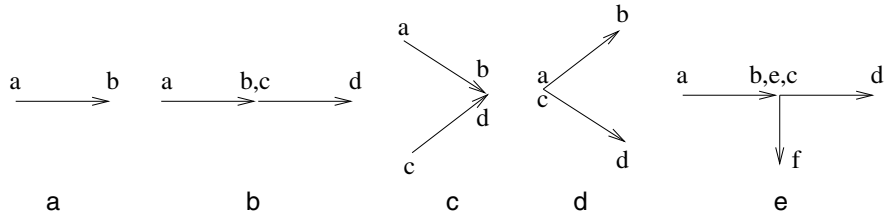

Figure 1. Examples of connectors in Reo.

The simple *Channel* in Fig 1.a can be expressed in $tD\pi$ as the process

$$P_{a@k,b@k} = k[[a^{\infty}?(X:T).(b^{\infty}!\langle X\rangle.(stop, Q_1), Q_2)]]$$

where the two ends are located at the same location k. The behaviour of the $tD\pi$ process above is of an *asynchronous* type of Reo channel. The process receives a value on the input channel a and replaces the variable X in the process that follows; after which sends the received value on the output channel b. Note that the notion of channel in $tD\pi$ differs from the notion in Reo. A Reo channel with the source a and the sink b located at different locations k and l respectively is modelled by the $tD\pi$ process

$$P_{a@k,b@l} = k[[a^{\infty}?(X:T).(go\, l.(b^{\infty}!\langle X\rangle, Q_1), Q_2)]].$$

The process uses the two channels a and b differently located by moving from one location to another. Note that if we use a discrete value for the timer, then we transform the simple Reo channels into timed channels. Messages can be transmitted through the channel only for a restricted period of time.

The connector in Fig. 1.b is called *Flow-through*, and it allows data to pass from source a of channel ab to sink d of channel cd. It is sufficient to put in parallel

two processes $P_{a,b}$ and $P_{c,d}$ which represent the two Reo channels ab and cd. The condition is that the sink b and the source c are the same; which in $tD\pi$ means that the channel names b and c represent the same channel. The corresponding process is:

$$P_F = k[[a^\infty?(X:T).(b^\infty!\langle X\rangle.(stop, Q_1), Q_2) \mid b^\infty?(X':T).(d^\infty!\langle X'\rangle.(stop, Q_1), Q_2)]]$$

Patterns of Reo can be modelled in $tD\pi$ with channel types. On a $tD\pi$-channel one can not receive values of a lesser type than the one specified in the type environment of the process.

For the connector in Fig. 1.c called *Merger*, a simple $tD\pi$ process can be given in the same way, only that we need the extended version of $tD\pi$ with the summation operator for nondeterministic choice (as required by the Reo behaviour).

We give now a timed version of the *Replicator* connector of Fig. 1.d.; its behaviour is that it replicates as many messages as possible. Consider the $tD\pi$ process

$$R_T \stackrel{def}{=} a^{\Delta 5}?(X:T).((b^{\Delta 20}!\langle X\rangle \mid c^{\Delta 6}!\langle X\rangle), R_T).$$

By the structural congruence we have ${}^*R_T \equiv R_T \mid {}^*R_T$. When receiving a message v_1 through a, the process reduces to

$$b^{\Delta 20}!\langle v_1\rangle \mid c^{\Delta 6}!\langle v_1\rangle \mid {}^*R_T.$$

If after 4 units of time another message v_2 is received, it is replicated and the system reduces to

$$b^{\Delta 16}!\langle v_1\rangle \mid c^{\Delta 2}!\langle v_1\rangle \mid b^{\Delta 20}!\langle v_2\rangle \mid c^{\Delta 6}!\langle v_2\rangle \mid {}^*R_T.$$

The *Take-cue Replicator* in Fig. 1.e is related to *Flow-through* of Fig. 1.b. This time the sink b and the sources e and c are the same (they represent the same channel name in $tD\pi$). After receiving a message from source a, the process must first send the message through the source e to sink f of channel ef, and only then it is allowed to send the message along the channel cd.

$$k[[a^\infty?(X:T).(b^\infty!\langle X\rangle.(b^\infty!\langle X\rangle.(stop, Q_1), Q_2) \mid b^\infty?(Y:T).(f^\infty!\langle Y\rangle.(stop, Q_1), Q_2) \mid b^\infty?(Z:T).(d^\infty!\langle Z\rangle.(stop, Q_1), Q_2), Q_2)]]$$

4 Related work

We briefly review three coordination models related to our approach. First we consider MoCha-π [14], a coordination calculus inspired from π-calculus [18] and based on mobile channels. MoCha-π is introduced as the coordination model of the MoCha middleware for distributed systems. Modelling channels by processes described in π-calculus offers the possibility of constantly creating new types of channels. The channels are similar to the ones in Reo, and have two possible different located communication ends. A process can connect or disconnect, read or write data to a channel-end. MoCha-π provides anonymous communication, and offers the possibility of replacing the processes at the ends of the channels or even the channels between processes. This is different from our model, first because MoCha-

π does not have an explicit notion of location which is quite important in distributed systems. There is also no explicit notion of time, and thus no time constraints over the communications. Furthermore, $tD\pi$ offers the possibility of defining resource access restrictions by means of a typing system. Although extending the π-calculus, MoCha-π does not have yet important technical results, and no notion of equivalence relation.

A recent formalisation of Linda coordination language is done by means of process algebra in [8]. Eight languages extending Linda with several primitives are compared. For each language an observational semantics is given by barbed bisimulations. The tuples in the tuple space are the only observables. The paper shows that process algebra is perfectly suited to model a coordination language based on data-space. However these models do not employ a typing system as in $tD\pi$ (no data access restriction scheme). There is no notion of time, thus no explicit time constraints. Some time extensions of Linda languages are presented in [16].

μKlaim [12] has been given recently as the model of the coordination language Klaim. The language Klaim was designed to program distributed systems composed by mobile components communicating through multiple tuple spaces. The syntax of μKlaim contains the notions of location and located components similar to $tD\pi$. Failures are an important feature of the distributed networks implemented by μKlaim. Observational semantics are studied by giving a may testing equivalence. A clear difference to our approach is given by timers which are not present in μKlaim. The typing system and the timers on channel types make $tD\pi$ a model suited for modelling a wider range of distributed systems with various time and resource access constraints, in contrast to μKlaim. Inspired by μKlaim, an extension of $tD\pi$ to incorporate failures should be taken into consideration in the future.

References

[1] Arbab, F., *Iwim model for coordination of concurrent activities, the*, in: P. Ciancarini and C. Hankin, editors, *1st International Conference on Coordination Languages and Models (COORDINATION '96)*, Lecture Notes in Computer Science **1061** (1996), pp. 34–56.

[2] Arbab, F., *Reo: a channel-based coordination model for component composition*, Mathematical Structures in Computer Science **14:3** (2004), pp. 329–366.

[3] Arbab, F., M. M. Bonsangue and F. S. De Boer, *A coordination language for mobile components*, , **1** (2000), pp. 166–173.

[4] Arbab, F., I. Herman and P. Spilling, *An overview of manifold and its implementation*, Concurrency: Practice and Experience **5:1** (1993), pp. 23–70.

[5] Baeten, J. and J. A. Bergstra, *Discrete time process algebra: Absolute time, relative time and parametric time*, Fundamenta Informaticae **29:1** (1997), pp. 51–76.

[6] Berger, M., *Basic theory of reduction congruence fortwo timed asynchronous pi-calculi.*, in: P. Gardner and N. Yoshida, editors, *15th International Conference On Concurrency Theory (CONCUR'04)*, Lecture Notes in Computer Science **3170** (2004), pp. 115–130.

[7] Busi, N., P. Ciancarini, R. Gorrieri and G. Zavattaro, *Coordination models: A guided tour*, in: A. Omicini, F. Zambonelli, M. Klusch and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer, 2001 pp. 6–24.

[8] Busi, N., R. Gorrieri and G. Zavattaro, *A process algebraic view of linda coordination primitives.*, Theoretical Computer Science **192:2** (1998), pp. 167–199.

[9] Carriero, N. and D. Gelernter, *Linda in context.*, Communications of ACM **32:4** (1989), pp. 444–458.

[10] Ciobanu, G. and C. Prisacariu, *Timers for distributed systems*, in: A. Di Pierro and H. Wiklicky, editors, *4th International Workshop on Quantitative Aspects of Programming Languages (QAPL 2006)*, Electronic Notes In Theoretical Computer Science **164:3** (2006), pp. 81–99.

[11] De Boer, F. S., M. Gabbrielli and M. C. Meo, *A timed linda language*, in: A. Porto and G.-C. Roman, editors, *4th International Conference on Coordination Languages and Models (COORDINATION'00)*, Lecture Notes in Computer Science **1906** (2000), pp. 299–304.

[12] De Nicola, R., D. Gorla and R. Pugliese, *Pattern matching over a dynamic network of tuple spaces.*, in: *7th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'05)*, Lecture Notes in Computer Science **3535** (2005), pp. 1–14.

[13] Felleisen, M. and A. K. Wright, *A syntactic approach to type soundness*, Information and Computation **115:1** (1994), pp. 38–94.

[14] Guillen-Scholten, J., F. Arbab, F. S. De Boer and M. M. Bonsangue, *Mocha-pi, an exogenous coordination calculus based on mobile channels.*, in: *Symposium on Applied Computing (SAC'05)*, 2005, p. 436.

[15] Hennessy, M. and J. Riely, *Resource access control in systems of mobile agents*, Information and Computation **173:1** (2002), pp. 82–120.

[16] Jacquet, J.-M., K. De Bosschere and A. Brogi, *On timed coordination languages.*, in: A. Porto and G.-C. Roman, editors, *4th International Conference on Coordination Languages and Models (COORDINATION'00)*, Lecture Notes in Computer Science **1906** (2000), pp. 81–98.

[17] Limniotes, T. A. and G. A. Papadopoulos, *Real-time coordination in distributed multimedia systems*, in: J. D. P. Rolim, editor, *IPDPS Workshops on Parallel and Distributed Processing*, Lecture Notes in Computer Science **1800** (2000), pp. 685–691.

[18] Milner, R., "Communicating and Mobile Systems: the π-Calculus." Cambridge Univ. Press, 1999.

[19] Milner, R. and D. Sangiorgi, *Barbed bisimulation*, in: W. Kuich, editor, *19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, Lecture Notes in Computer Science **623** (1992), pp. 685–695.

[20] Papadopoulos, G. A., *Models and technologies for the coordination of internet agents: A survey*, in: A. Omicini, F. Zambonelli, M. Klusch and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer, 2001 pp. 25–56.

[21] Papadopoulos, G. A. and F. Arbab, *Coordination models and languages*, Advances in Computers **46** (1998), pp. 329–400.

[22] Ricci, A. and M. Viroli, *A timed extension of respect*, in: H. Haddad, L. M. Liebrock, A. Omicini and R. L. Wainwright, editors, *Symposium on Applied Computing (SAC'05)* (2005), pp. 420–427.

[23] Sangiorgi, D., "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms," Ph.D. thesis, University of Edinburgh (1992).

[24] Yi, W., *Real-time behaviour of asynchronous agents*, in: J. C. M. Baeten and J. W. Klop, editors, *Theories of Concurrency: Unification and Extension (CONCUR'90)*, Lecture Notes in Computer Science **458** (1990), pp. 502–520.