Coordination by Timers for Channel-Based Anonymous Communications\textsuperscript{1}

Gabriel Ciobanu and Cristian Prisacariu\textsuperscript{2}

\textit{Institute of Computer Science, Romanian Academy} \\
Bld. Carol 8, 700505 Iași, Romania

Abstract

In considering systems of distributed processes, we must deal with access to resources, locations and communication among processes. In this paper we present a new model for timed coordination of communicating distributed processes using an extension of the $\pi$-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. Coordination is given by these timers, together with additional coordination rules. We define the syntax of the model and its operational semantics and then provide some results regarding the typing system and the timers. A timed barbed bisimulation relation is defined to compare the processes in different settings. The timed coordination aspects are given through a coordinator pair. It consists of a timers assigning function which can be dynamically changed, and a set of coordination rules. Some illustrative examples are given. Finally we compare the new model with other existing coordination models.

\textit{Key words:} timers, typing system, locations, $\pi$-calculus

1 Introduction

During the last two decades many coordination languages and models were developed. These are divided into two major categories: data-driven (Linda-like) and control-driven or channel-based (Manifold-like) models. A comprehensive survey on coordination models and languages has been given in [22]. More recently in [21,7] authors survey the state of the art in coordination models for systems of agents (which has become of great interest in the last years).

One disadvantage with 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.

\textsuperscript{1} Research partially supported by CEEX Project 47/2005  
\textsuperscript{2} Email: gabriel@iit.tuiasi.ro and cprisacariu@iit.tuiasi.ro

©2006 Published by Elsevier Science B. V.
We consider coordination to be mainly a problem of message communications and time scheduling. This means that time is important, both for expressing communication availability and resource access (resources being the communication channels). In [15], Hennessy and Riely introduce and study a formalism called distributed \( \pi \)-calculus (\( D\pi \)) as an extension with types and locations of the \( \pi \)-calculus [18]. The central operation in \( \pi \)-calculus is the communication of names through channels among processes. \( D\pi \) provides a 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. Whenever a channel timer reaches the value 1, the channel is discarded. Similarly, a channel type is lost whenever its attached timer expires. The new calculus is called timed distributed \( \pi \)-calculus (\( tD\pi \)). Over this model we define time coordination by assigning values to timers and defining specific coordination rules. The model gives us a strong formal ground for designing of coordinated systems, where a coordination language comes as a linguistic embodiment of the model.

The time aspects of process algebras is an intense studied topic. An extension of the \( \pi \)-calculus with a timer construct is introduced and studied in [6]. Process algebras based on a discrete time domain are studied in [5] and a timed bisimulation has been given in [25].

In coordination languages, most related to our work are Manifold [4] and its timed extension in [17], and Reo [2]. MoCha-\( \pi \) [14] and \( \sigma \pi \) [3] are models based on \( \pi \)-calculus for channel-based coordination languages. Manifold is based on the IWIM model [1] and has basically two kinds of processes; manager and worker. The manager coordinates of 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 determin the manager to change its state.

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 in Linda, and present several results regarding observational equivalences. \( \mu \)Klaim [12] has been given recently as the model of KLAIM (data-driven) coordination language.

Time has also been investigated 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 (\( \text{in, out or rd} \)) take no time to execute, i.e., one time unit means the execution of such an action. A new action is introduced which makes possible to wait for a tuple only
for a predetermined number of time units. After the time passes the process changes state. In [16] several extensions of Linda with different notions of time are introduced. And most recently in [23] a extension with time of the tuple-centre based model ReSpecT was presented. 

The model introduced with $tD\pi$ is included in the large class of control-driven models. The triggering events in $tD\pi$ are the communications on channels or the migration with go or the expiring of a timer. A $tD\pi$ process is eager in the sense that it needs to make as much communications as possible. With timers $tD\pi$ goes beyond the coordination only through channels of certain types or messages transmitted through the channels from the manager processes to worker processes. 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 for certain channels (as types) and also timer values for that channel (defining the time constraints). Thus, 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 agents 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, besides communication a process may move from one location to another (by go actions). 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 $\pi$-calculus. A detailed presentation can be found in [10]. In Section 2.3 we give some technical results stating 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 models.

2 Timed Distributed $\pi$-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. The timer $\Delta t$ of each channel makes the channel available for communication only for the period of time determined by $t$. As we have synchronous communication we consider timers for both input and output channels. The rational behind the choice of adding timers to outputs comes from the fact that in distributed systems we have both multiple clients and multiple servers. This means that clients can switch from one server to another depending on the waiting time. To simplify the presentation we omit the syntax for matching or summation of the classical $\pi$-calculus. 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 $a^{\Delta t}?(X : T).(P, Q)$ reduces to $P$ whenever a communication is established during the interval of time given by $\Delta t$; otherwise it reduces 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.

| $u ::= x$ | Variable Name | $P, Q ::= stop$ | Termination |
| $| a^{\Delta t}$ | Timed Channel | $| P | Q$ | Composition |
| $l ::= x$ | Variable Name | $(\nu u : A)P$ | Channel Restriction |
| $| k$ | Location Name | $| go_1P$ | Movement |
| $v ::= bv$ | Base Value | $| u!(v).(P, Q)$ | Output |
| $| u | l$ | Name | $| u?(X:T).(P, Q)$ | Input |
| $| u@l$ | Located Name | $| *P$ | Replication |
| $(v_1,\ldots,v_n)$ | Tuple of Values | $M, N ::= M | N$ | Composition |
| $X ::= x$ | Variable | $(\nu u@l : T)N$ | Located Restriction |
| $| X@l$ | Located Variable | $| l[[P]]r$ | Located Process |
| $(X_1,\ldots,X_n)$ | Tuple of Variables |

Two channels are equal $a^{\Delta t_1}_1 = a^{\Delta t_2}_2$ if and only if $a_1 = a_2$ and $t_1 = t_2$. Waiting indefinitely on a channel $a$ is allowed by considering $\Delta t$ as $\infty$. An output process expression $a^{\infty}!(v).(P, Q)$ awaits forever to send the value $v$, simulating the behaviour of an output process in untimed $\pi$-calculus. Two processes running in parallel can interact along a common channel $a$:

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

Each process is tagged with a type environment $\Gamma$ which is a set of location types. A location type 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 $res\{r(T), w(T'), ro(T''\})$ 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. The set of types increases when a name is received along a $r(\cdot)$ channel. With $ro(\cdot)$ capability we describe processes which may use a received channel only if their type environment has a corresponding channel type.

In $D\pi$ the resources are accumulated, but they can never be lost (discarded). We extend the channel types of $D\pi$ with timers of form $\Delta t$. Communication is now permitted on channels only in the interval of time given by the timer value $t$ (until its type timer 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>
<thead>
<tr>
<th>Table 2: Type system and subtyping relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types:</td>
</tr>
<tr>
<td>Subtyping:</td>
</tr>
<tr>
<td>K ::= \text{loc}{\kappa} \quad \kappa &lt; (: \kappa</td>
</tr>
<tr>
<td>A ::= \text{res}{\alpha}\Delta t \quad a : A &lt; (: A' if A &lt;: A'</td>
</tr>
<tr>
<td>E ::= A</td>
</tr>
<tr>
<td>T ::= E</td>
</tr>
<tr>
<td>\quad</td>
</tr>
<tr>
<td>Capabilities:</td>
</tr>
<tr>
<td>r(T) &lt; (: r(T') if T &lt; (: T'</td>
</tr>
<tr>
<td>w(S) &lt; (: w(S') if S &lt; (: S</td>
</tr>
<tr>
<td>\alpha ::= r(T)</td>
</tr>
</tbody>
</table>

\(B\) represents a set of base types. The subtyping relation < (: is similar to the subtyping relation of \(D\pi\), excepting the new type \(ro\). 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). A process moves to a location, and waits for a period of time to establish a communication on a particular channel. We offer capabilities as \(r/w/ro\) for the fixed resources, telling 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 \(r(T)\) can receive (without generating errors) only messages of type \(T\); the error system is presented in Table 4. When the channel type \(\text{res}(r(T))\) is extended with \(r(T')\), it follows naturally that the process is now able to receive messages of higher types: \(T\) and \(T'\).

We define a function \(\psi\) 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 generates a runtime error. Thus we define \(\psi\) to remove also the empty locations.

**Definition 2.1** (Cleanup function) \(\psi : \mathcal{P}_\Delta \rightarrow \mathcal{P}_\Delta\) is defined over the set of tagged located processes such that \(\psi(l[[P]]_r) = l[[P]]_r\), where \(l\) can be any location in the distributed system and \(\Gamma'\) is obtained from \(\Gamma\) where every type \(c : \text{res}()\Delta t, t > 1, t \neq \infty\) is changed to \(c : \text{res}()\Delta(t - 1)\), and every
$c: \text{res}(\cdot) \Delta 1$ disappears. Location types of form $\text{loc} : \{\text{go}\}$ are removed.

2.2 Semantics

The passage of time is described by a time-stepping function $\phi_\Delta$ 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. $\phi_\Delta$ decreases 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 $\phi_\Delta$ to check the existence of the needed types and change the process accordingly. As $\phi_\Delta$ decreases the channel timers we also extend it to take care of the type environments by applying the cleanup function $\psi$. In the definition of $\phi_\Delta$ we drop the channel type and the transmitted message in the input and output processes for brevity. In the $\text{go}k$ syntax if the location type contains the capability $\text{go}$, then $R$ is executed; if $k$ is not defined in $\Gamma$, then $Q$ is executed. If $\text{go}$ is not present, the process is considered to do something against its permissions and an error is generated.

**Definition 2.2** (Tagged time-stepping function) We define $\phi_\Delta : \mathcal{P}_\Delta \rightarrow \mathcal{P}_\Delta$, where $\Gamma'$ is obtained by application of the cleanup function $\psi$, by:

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

We consider the type environment as a mapping from names to types. A type environment is associated with each located process to restrict the range of resources it can access. Well-typedness of processes is defined by a set of static rules which are included in Appendix. 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.

We write $\Gamma \vdash P$ and say the process $P$ is well-typed with respect to type environment $\Gamma$; we also write $\Gamma \vdash_k P$ and say that $P$ is well-typed to run at location $k$. In order to say that $a^{\Delta t}(v).\langle R, Q \rangle$ is well-typed to run at location $k$, with respect to type environment $\Gamma$, 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) $\Gamma \vdash_k a : \text{res}(w(T)) \Delta t$, 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 \(\psi\) changes the capability set \(\Gamma\) by removing channel and location types, we are interested if the process is still well-typed under the new \(\Gamma'\). The following lemma relates the type 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 \(\Gamma \models l[[P]]_{\Delta}\) then \(\Gamma \models \psi(l[[P]]_{\Delta})\).

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 \((R_{\Gamma}\text{-COM1})\) and \((R_{\Gamma}\text{-COM2})\) cannot be applied. Using these notations, we give the following reduction rules providing a dynamic semantics for \(tD\pi\).

\[\Gamma(l, a) \lessdot: \text{res}\{r(T)\} \quad (R_{\Gamma}\text{-IDLE})\]

\[
\frac{l[[P]]_{\Gamma} \not\rightarrow}{l[[P]]_{\Gamma} \rightarrow \phi_{\Delta}(l[[P]]_{\Gamma})}
\]

\[\Gamma(l, a) \lessdot: \text{res}\{ro(T)\} \quad (R_{\Gamma}\text{-COM1})\]

\[
\frac{[\alpha^{\Delta_1}(v), (P, Q)]_{\Delta} \mid l[[\alpha^{\Delta'}?(X : T), (P', Q')]]_{\Gamma} \rightarrow \psi(l[[P]]_{\Delta}) \mid \psi(l[[P']\{v/X\}]_{\Gamma})_{\Gamma \equiv \{v \equiv \alpha : T\}}}{\Gamma(l, a) \lessdot: \text{res}\{ro(T)\}}
\]

\[\frac{N \rightarrow N' \quad M \rightarrow M' \quad (R_{\Gamma}\text{-RES})}{N \mid M \rightarrow N' \mid M'}\]

\[\frac{N \equiv N' \quad N \rightarrow M \quad M \equiv M'}{N' \rightarrow M'}\]

\[\frac{(\nu a \equiv l : T)N \rightarrow (\nu a \equiv l : T)N'}{(R_{\Gamma}\text{-CONG})}\]

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

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.

We describe the error system of \(tD\pi\) in the Appendix and we denote by...
the generation of an error. 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 \( \phi_\Delta \) 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 because of lack of space; they can be found in [10].

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

**Proof:** This lemma states that the passage of time does not interfere with the typing system. \( \phi_\Delta \) does not change the property of a process of being well-typed under some \( \Gamma \). To prove this we consider the form of \( P \) from the definition of \( \phi_\Delta \). We use induction on the depth of the inference tree. \( \square \)

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

(a) If \( N \equiv N' \) then \( \Gamma \models N \) if and only if \( \Gamma \models N' \).

(b) If \( N \rightarrow N' \) then \( \Gamma \models N \) if and only if \( \Gamma \models N' \).

**Proof:** For the first part of the theorem, which relates the typing system with the structural equivalence (which is similar to the one in \( D\pi \)) proof proceeds by considering all the equivalence equations. The part (b) asserts the consistency between the static and dynamic semantics. We proceed by induction on the depth of inference tree given by \( N \rightarrow N' \). We also use Lemma 2.3 which relates time and type environments, and Lemma 2.4 which relates time and channel names. \( \square \)

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 property states that if a system is well-typed, then it cannot give rise to runtime errors, and this is denoted by \( P \not\rightarrow^{err} \).

**Theorem 2.6** For all tagged located processes \( N \) and all type environments \( \Gamma \) such that \( \Gamma \models N \) we have \( N \not\rightarrow^{err} \).

**Proof:** The proof considers the contrapositive of the theorem which states that if \( N \) gives rise to a runtime error \( (N \not\rightarrow^{err}) \), then \( N \) cannot be well-typed under any type environment \( \Gamma \) \( (\Gamma \not\models N, \forall \Gamma) \). We use induction on the structure of \( N \) and consider a proof cases for each rule in the definition of the runtime errors (in Table 4 of the Appendix). \( \square \)
**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. 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* communication. We consider as unobservable the *movement with go*, the application of the *time-stepping function* $\phi_\Delta$, and the *internal interaction* of processes. Intuitively an observer of $P$ is a process $Q$ 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 (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. A *timed observer* monitors the timers of the channel types inside the type environment, or and the timers on 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 $\Delta$; denoting the observers distinguishing power over types. The barb $\downarrow^{t,t^\Delta}_{\mu a@k}$ identifies processes which have enough permissions for the channel $\mu$ with respect to the type environment $\Delta$.

**Definition 2.7** A timed global typed barb predicate, $\downarrow^{t,t^\Delta}_{\mu a@k}$ where $\mu \in \{a?, a!\}$ with $a$ being any channel name, is defined inductively by the following system of rules. We denote by $\mu$ the names of the input or output channels. If $\mu = a?$ then $\mu = a$.

\[
\begin{align*}
\Gamma(k,a) < : \Delta(k,a) & \quad \Gamma(k,a) = res\{\ldots\}\Delta' & \quad \Gamma(k,a) < : \Delta(k,a) \\
\Gamma(k,a) = res\{\ldots\}\Delta' & \quad k[[a^\Delta @ \nu.(P,R)] ]r \downarrow^{t,t^\Delta}_{\mu a@k} & \quad k[[a^\Delta ? ((X:T).(P,R)) ]r \downarrow^{t,t^\Delta}_{a@0k} \\
N \downarrow^{t,t^\Delta}_{\mu a@k} & \quad N \downarrow^{t,t^\Delta}_{\mu a@k} & \quad \mu \neq a \\
N | M \downarrow^{t,t^\Delta}_{\mu a@k} & \quad (\nu a @ l : A) N \downarrow^{t,t^\Delta}_{\mu a@k} & \quad k[[P]] \downarrow^{t,t^\Delta}_{\mu a@k} \\
N | M \downarrow^{t,t^\Delta}_{\mu a@k} & \quad \downarrow^{t,t^\Delta}_{\mu a@k} & \quad k[[P]] \downarrow^{t,t^\Delta}_{\mu a@k} \\
\end{align*}
\]

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

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

Two processes are timed global typed barbed bisimilar, denoted $P \sim_{\text{GTB}} Q$,
if and only if \((P, Q) \in S\) for some timed global typed barbed bisimulation \(S\).

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 [24]. Equivalently, we can close the barbed bisimulation under all observers well-typed with respect to the type environment \(\Delta\). Thus \(N\) and \(M\) are timed global typed barbed equivalent \((N \sim_{\text{tgTB}} M)\) if and only if for all \(\Delta \models_{\text{T}} O\) we have \(N \mid O \sim_{\text{tgTB}} M \mid O\).

3 Coordination Part

Because \(tD\pi\) uses distributed resources and code migration we find 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 also defined in \(tD\pi\).

- **Coordination entities** are the distributed mobile communicating processes;
- **Coordination media** is composed by the timers assigning function \(T,A\) 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 \(CP\).

\(tD\pi\) 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 \((T,A)\) is a function assigning values to the timers, and the second component \((CP)\) 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 \(\Delta t\).
**Definition 3.1 (Assigning values to timers)**
We denote by $\Delta T$ the set of timers on channels together with the timers on types. An assigning of natural values to the timers in $\Delta T$ is done through a function $TA : \Delta T \rightarrow \mathbb{N} \cup \{\infty\}$. We denote by $\Delta T|_P$ the set of timers specific to process $P$; considering a process $P$, $TA$ is restricted naturally to $\langle TA|_P \rangle$.

In an arbitrary process $P = a^{\Delta t}(v),(R,Q)$, the timer can take two special values: $\infty$ and 0. If the value is $\infty$, the process can wait on channel $a$ forever. If the value is 0, 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 $CP$ 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. In $tD\pi$, the coordination part is expressed as a pair $\langle TA,CP \rangle$ called coordinator, denoted by $C$.

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

An equivalence relation can be defined over the timers assigning functions as follows. First we need a mapping $(\cdot,\cdot) : P_\Delta \times TA \rightarrow P_\Delta$ which changes the times of a process $P$ accordingly to a function $TA_1$. Given a protocol $CP$, we say that two functions $TA_1$ and $TA_2$ are equivalent if they do not change the related behaviours of any two systems; i.e., $\forall P,Q \in P_\Delta, (P,TA_1) \sim^C_{iGTB} (Q,TA_1) \leftrightarrow (P,TA_2) \sim^C_{iGTB} (Q,TA_2)$. An example of two equivalent functions $TA_1$ and $TA_2$ is given by a simple constant translation; i.e., $TA_2(\Delta t) = TA_1(\Delta t) + c$, where $c$ is a constant. This equivalence relation over timers assigning functions can be extended to coordinators: two coordinators $C_1,C_2$ are equivalent, denoted $C_1 \sim C_2$, if they have the same set of rules $CP$, and the corresponding timers assigning functions are equivalent. The timed bisimilar relation with respect to a coordinator $(\sim^C_{iGTB})$ and the equivalence relation $\sim$ over coordinators are related by the following result:

**Theorem 3.2** For every processes $P,Q$ and coordinators $C_1,C_2$,
if $P \sim^C_{iGTB} Q$ and $C_1 \sim C_2$, then $P \sim^C_{iGTB} Q$.

The new defined bisimulation emphasises the role of timers assigning functions in coordination. The set $CP$ of rules become more important in coordination languages, when the algorithmic aspects are predominant in the strategies of controlling the computation. The aspects described by $CP$ are visible at the implementation of a coordination language according to its coordination model. Interesting insides about $CP$ can be drawn from work on Law Governed Interactions in [20].
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.

![Diagram of connectors in Reo](image)

Figure 1. Examples of connectors in Reo.

The simple channel in Fig 1.a can be expressed in $tD\pi$ as the process $k[[a^{\infty}?(X : T).b^{\infty}!(X).\text{stop}]]$ where the two ends are located at the same location $k$. 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 $l$ and $k$ respectively is modelled by the $tD\pi$ process $l[[a^{\infty}?(X : T).go.k.b^{\infty}!(X)]]$. 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 we transform the Reo channels into timed channels. Messages can be transmitted through the channel only for a restricted period of time.

We give now a time version of the replicator connector of Fig. 1.d. The Reo behaviour is that it replicates as many messages as possible. Consider the $tD\pi$ process $R_T \overset{df}{=} a^{\Delta5}?(X : T).((b^{\Delta20}!(X) | c^{\Delta6}!(X)), R_T)$. By the structural congruence we have $^*R_T \equiv R_T \upharpoonright ^*R_T$. When receiving a message $v_1$ through $a$, the process reduces to $b^{\Delta20}!(v_1) | c^{\Delta6}!(v_1) | ^*R_T$. If after 4 units of time another message $v_2$ is received, it is replicated and the system reduces to $b^{\Delta10}!(v_1) | c^{\Delta21}(v_1) | b^{\Delta20}!(v_2) | c^{\Delta6}!(v_2) | ^*R_T$.

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

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.

The Take-cue Replicator in Fig. 1.e is modelled by the $tD\pi$ process $a?((X:T)e!((C),c!(X))$. After receiving a message through $a$ the process must first send that message through the source $e$ of the Reo channel $ef$ and only then it can send the message along the channel $cd$.

4 Conclusion

We overview three coordination models related to our approach. We consider first MoCha-$\pi$ [14], a coordination calculus inspired from $\pi$-calculus [18] and based on mobile channels. MoCha-$\pi$ is introduced as the coordination model of the MoCha middleware for distributed systems. Modelling channels by processes described in $\pi$-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-$\pi$ 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-$\pi$ 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 $\pi$-calculus, MoCha-$\pi$ does not have yet many 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 and form a lattice. 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. 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. Time constraints in Linda models are useful, if we look at the time extensions of Linda languages given in [16].

$\mu$Klaim [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 $\mu$Klaim contains the notions of location and located components similar to $tD\pi$. Failures are an important feature of the distributed networks implemented by $\mu$Klaim. 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 µKlam. The typing system and the timers on channel types make tDπ a model suited for modelling a wider range of distributed systems with various time and resource access constraints, in contrast to µKlam. An extension of tDπ to incorporate failures should be taken into consideration for the future work.

By now we have laid down the timed distributed π-calculus as a coordination model based on timed message communications through channels of certain types. We have added a time coordinator as a pair of a timers assigning function TA and a coordination protocol CP. We have emphasised the time coordination and the expressiveness of tDπ by modelling simple notions from the channel-based model Reo. The new model is sound with respect to the type system. We have also given a timed barbed bisimulation in order to be able to compare the behaviours of two processes. The bisimulation is extended to compare the processes with regard to specific coordinators.

References


Appendix

The soundness proof in Section 2.3 depends upon a set of typing rules. These rules express the necessary conditions which must be satisfied for each syntactic construction of a process to be well-typed. The typing rules below describe the static semantics of $tD\pi$. Considering the rules (T-R$_{new}$) and (T-W$_{new}$), we accept tagged located processes with missing channel types (the types are removed with the passage of time). These processes do not generate errors.

<table>
<thead>
<tr>
<th>Table 3: The Typing System (Typing rules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes</td>
</tr>
<tr>
<td>(T-R)</td>
</tr>
<tr>
<td>$\Gamma \vdash_1 \alpha : \text{res}{\tau}\Delta t$</td>
</tr>
<tr>
<td>$fv(X) \cap fv(\Gamma) = \emptyset$</td>
</tr>
<tr>
<td>$\Gamma {X @ l : T} \vdash P$</td>
</tr>
<tr>
<td>$\Gamma \vdash_1 Q$</td>
</tr>
<tr>
<td>$\Gamma \vdash_1 a^{\Delta!}(X : T).(P, Q)$</td>
</tr>
<tr>
<td>(T-NEWCH)</td>
</tr>
<tr>
<td>$\Gamma(l) &lt; \text{loc}{\text{newch}}$</td>
</tr>
<tr>
<td>$\alpha \not\in fn(\Gamma)$</td>
</tr>
<tr>
<td>$\Gamma {a @ l : A} \vdash P$</td>
</tr>
<tr>
<td>$\Gamma \vdash_1 (\nu a : A)P$</td>
</tr>
<tr>
<td>(T-R$_{new}$)</td>
</tr>
<tr>
<td>$a : \alpha \not\in \Gamma(l)$</td>
</tr>
<tr>
<td>$\Gamma \vdash_1 a^{\Delta!}(X : T).(P, Q)$</td>
</tr>
<tr>
<td>Located Processes</td>
</tr>
<tr>
<td>$\Delta \vdash_1 P$</td>
</tr>
<tr>
<td>$\Delta \vdash_1 a^{\Delta!}(\nu a : A) \vdash_1 N$</td>
</tr>
<tr>
<td>$\Gamma \vdash_1 l[[P]]_{\Delta}$</td>
</tr>
</tbody>
</table>

We describe the error system of $tD\pi$: $\frac{\text{err}}{\text{err}}$ means that an error is generated. $\text{roobj}(\cdot)$, $\text{roobj}(\cdot)$, $\text{wobj}(\cdot)$ are partial functions defined over the set of channel types, and return the transmitted type. For example, considering in type environment $\Gamma$ at location $l$ a channel type $\alpha : \text{res}\{w(T), w(T')\}$, the
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. A runtime error appears 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 resort process is chosen by \( \phi_\Delta \) function. We 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).

<table>
<thead>
<tr>
<th>Table 4: Runtime errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E-GO) ( \Gamma(k) ) is defined and ( \Gamma(k) \not &lt;: \text{loc}{go} )</td>
</tr>
<tr>
<td>( l[\text{go } k.(P,Q)]_\Gamma \xrightarrow{err} )</td>
</tr>
<tr>
<td>(E-SND) ( \Gamma(l, a) ) is defined and ( \Gamma(l, v) \not &lt;: wobj(\Gamma(l, a)) )</td>
</tr>
<tr>
<td>( l[\Delta t\langle v \rangle.(P,Q)]_\Gamma \xrightarrow{err} )</td>
</tr>
<tr>
<td>(E-RCV) ( \Gamma(l, a) ) is defined and ( \text{roobj}(\Gamma(l, a)) \not &lt;: T ) or ( \text{roobj}(\Gamma(l, a)) \not &lt;: T )</td>
</tr>
<tr>
<td>( l[\Delta t\langle X : T \rangle.(P,Q)]_\Gamma \xrightarrow{err} )</td>
</tr>
<tr>
<td>( \Gamma(l, a) ) and ( \Delta(l, a) ) are defined and</td>
</tr>
<tr>
<td>(E-COM) ( wobj(\Gamma(l, a)) \not &lt;: \text{roobj}(\Delta(l, a)) ) or ( wobj(\Gamma(l, a)) \not &lt;: \text{roobj}(\Delta(l, a)) )</td>
</tr>
<tr>
<td>( l[\Delta t\langle v \rangle.(P,Q)]<em>\Gamma \rightarrow l[\Delta t\langle X : T \rangle.(P,Q)]</em>\Delta \xrightarrow{err} )</td>
</tr>
<tr>
<td>(E-NEW) ( \nu a(k : T)N \xrightarrow{err} )</td>
</tr>
<tr>
<td>(E-NEW) ( \nu a(k : T)N \xrightarrow{err} )</td>
</tr>
<tr>
<td>(E-NEW) ( \nu a(k : T)N \xrightarrow{err} )</td>
</tr>
</tbody>
</table>