Timers for Distributed Systems [1]

Gabriel Ciobanu and Cristian Prisacariu [2]

Institute of Computer Science, Romanian Academy
Blvd. Carol 8, 700505 Iaşi, Romania

Abstract

We deal with temporal aspects of distributed systems, introducing and studying a new model called *timed distributed π-calculus*. This model extends distributed π-calculus with timers, transforming the communication channels into temporary resources. Distributed π-calculus describes located interactions between processes with restricted access to resources. We introduce time constraints by considering timeout timers for channels. Combining these timers with types and locations, we provide a formal framework able to describe complex systems with constraints on time and on resource access. Its typing system and operational semantics are presented. It is proved that the passage of time does not interfere with the typing system. The new model is proved to be sound by using a method based on subject reduction.

Key words: timers, typing system, locations, subject reduction

1 Introduction

In this paper we use timers and study their role in modelling complex systems of distributed and mobile processes. We select the π-calculus [10] as a ground platform; this formalism is well suited for modelling systems based on communicating processes. In order to emphasise the spatial aspects in distributed systems we use an explicit notion of *location*. The interaction between processes can be controlled by using various *sorts*. The sorts allow to restrict the use of distributed resources, namely located communication channels.

A combination of locations and sorts for the π-calculus is already presented in [6]; the resulting calculus is called *distributed π* ($D\pi$). In $D\pi$ the authors use the word "types" (instead of "sorts") to express certain capabilities for the interaction channels. Sorting is used in the π-calculus to define patterns of interactions; the sort of an interaction channel defines the type of the messages

[1] Research partially supported by CEEX Project 47/2005
[2] Email: gabriel@iit.tuiasi.ro and cprisacariu@iit.tuiasi.ro

sent or received along that channel. By "interaction between processes" we understand a "communication between processes". A communication channel is considered to be a fixed resource at a certain location. The communication is local and *code migration* is used to move processes to the same location, in order to communicate along a common local channel for which they have proper capabilities. The typing system offers the possibility to restrict the access to resources by tagging the processes with a *type environment*, and to restrict the messages that could be transmitted along the channels.

We take up $D\pi$, extending it with decreasing timers attached to communication channels and to channel types. The new formalism is called *timed distributed π-calculus* ($tD\pi$), and it is presented as a rigorous framework for describing distributed systems with time and resource constraints. The timers on channels define timeouts for communications, and timers on the channel types restrict the channels availability. Whenever the timer of either a channel or a channel type expires, the corresponding channel is discarded, and respectively the channel type is lost. $tD\pi$ combines the temporal constraints with types and locations in order to give the possibility of modelling *located* and *timed* interactions between distributed processes with *time restricted resource access*. Following the method introduced in [4], we prove that the typing system of $tD\pi$ is sound with respect to the equivalence and reduction relations of the π-calculus. Moreover, time does not interfere with the typing system.

2 Syntax and Semantics of $tD\pi$

By adding timers to communication channels, communication along a channel is no longer available for an indefinite time (like in $D\pi$). If no interaction happens in the predefined interval of time determined by the timer value, the process goes to another state. Each channel has two alternatives: one when the communication is achieved, and another when we have no communication. The channel timers are created once with the channel, but started only when the channel becomes active (available for communication).

2.1 $tD\pi$ Syntax

The syntax of a $D\pi$ channel a is extended by tagging it with a timer Δt; this means that the channel $a^{\Delta t}$ is waiting for communication only for the period of time determined by the timer value t (namely t units of time, as we use a discrete time domain).

The syntax of *Input* and *Output* communication uses a pair of processes (P, Q). For instance, the *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. In this expression, the variable X of type T is considered bounded only in P. 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 output processes (clients) can switch from one server to another depending on the waiting time. In general, an input process awaits for a resource for a certain period of time, and an output process offers a resource for a certain period of time.

Table 1: *Syntax of $tD\pi$*

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

The above table defines in order the *names*, *values*, *variables*, *processes* and *tagged located processes* of $tD\pi$. For a variable X of the *Input* expression $a^{\Delta t}?(X : T).(P, Q)$ we should also provide its type T, and for the channel name a in the *Channel Restriction* expression we have to provide its channel type A (the types are presented in Section 2.2).

Note that with the located restriction $(\nu\, a@k : T)N$ we specify a new private channel a and its location k. For example, in the process

$$(\nu\, a@k : T)(l[[P]] \mid k[[Q]]) \mid k[[Q']]$$

the channel a is private to P and Q and is located at the current location k of Q. Moreover, Q' does not have any knowledge about channel a even though it runs also at location k. This means that process P must move to location k before communicating on the private channel a. Also note that the syntax for channel restriction specifies only the name of the private channel, and not the associated timer; this is because a restriction refers only to the names of the channels.

The interaction between processes is given through the input and output process expressions which must have the same channel name; the channel timers are playing a secondary role in such an interaction.

Example 2.1 The following two processes running in parallel can interact along the common channel a.

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

Intuitively, the process on the left evolves to the process on the right after such an interaction. The output process (the process on the left of the parallel composition operator) sends the value v on the channel named a and then behaves as P. When receiving the value v, in the input process (the process on the right of the parallel composition operator) all the occurrences of the bound variable X are replaced by v in P'.

Waiting indefinitely on a channel a is allowed by considering Δt as ∞. An output process expression $a^{\infty}!\langle v\rangle.(P,Q)$ awaits forever to send the value v, simulating the behaviour of an output process in untimed synchronous π-calculus.

2.2 Typing System

Each located process is tagged with a type environment Γ which is a set of *location types* denoted by K in Table 2. Formally the type environment is a mapping from location names k to location types K. A location type K may contain location capabilities denoted by κ; these capabilities may express either capabilities of using *channel names* $\tilde{a}$ with their corresponding *channel types* $\tilde{A}$ ($\tilde{a}$:$\tilde{A}$), or *move* capabilities *go*, or *channel restriction* capabilities (i.e., permissions to create private channels) *newch*. A *channel type* A may contain the following channel capabilities generically denoted by α: capability $r\langle T\rangle$ of *reading* messages of type T, capability $w\langle T\rangle$ of *writing* messages of type T, and capability $ro\langle T\rangle$ of *reading only* messages of type T. A type T may contain tuples $(T_1,\ldots,T_n)$ of types corresponding to tuples of names, and channel types $A_1,\ldots,A_n@K$ corresponding to channel names $a_1,\ldots,a_n$ located at a location of type K. $\mathcal{B}$ represents the set of *base types*.

We may have only one instance of the capabilities *go* and *newch* in a location type K; they represent respectively the capability of a process to move to that location, and the capability to create private channel names at that location.

In order to exemplify, let us consider a process which has in its type environment Γ a channel name a with a channel type $res\{r\langle T\rangle, w\langle T'\rangle, ro\langle T''\rangle\}$. This means that along this channel a the process can receive messages of type T, and send messages of type T'. The ro capability is similar to an r capability, with the difference that the types of the received messages are not added to the type environment of the process. Types are accumulated when a name is received along an input channel with capability $r\langle\,\rangle$.

Having $ro\langle\,\rangle$ capabilities, we can describe processes which may use the data received in a message through an input channel with capability $ro\langle\,\rangle$ only if there exists a proper type for the new data within their type environments. More precisely, let us consider a process P at location k which receives a located channel name $b@k$ on the input channel a of type $res\{ro\langle T\rangle\}$. The located process $k[[P]]_{\Gamma}$ can use the new channel name b to communicate with-

Table 2: *Type system and subtyping relation*

Types:

$$\begin{aligned}
\mathrm{K} &::= \mathrm{loc}\{\tilde{\kappa}\} \\
\mathrm{A} &::= \mathrm{res}\{\tilde{\alpha}\}\Delta t \\
\mathrm{E} &::= \mathrm{A} \mid \mathrm{K} \mid \mathcal{B} \\
\mathrm{T} &::= \mathrm{A} \mid \mathcal{B} \mid (T_1, \ldots, T_n) \\
&\quad \mid A_1, \ldots, A_n@K
\end{aligned}$$

Capabilities:

$$\begin{aligned}
\kappa &::= a : A \\
&\quad \mid go \mid newch \\
\alpha &::= r\langle T\rangle \mid w\langle T\rangle \mid ro\langle T\rangle
\end{aligned}$$

Subtyping:

$$\begin{aligned}
\kappa &<: \kappa \\
a : A &<: a : B \quad \text{if } A <: B \\
K &<: L \quad \text{if } \forall \lambda \in L\colon \exists \kappa \in K\colon \kappa <: \lambda \\
A &<: B \quad \text{if } \forall \beta \in B\colon \exists \alpha \in A\colon \alpha <: \beta \\
\tilde{A}@K &<: \tilde{B}@L \quad \text{if K<:L and } \tilde{A}<:\tilde{B} \\
\tilde{S} &<: \tilde{T} \quad \text{if } \forall i : S_i <: T_i \\
r\langle T\rangle &<: r\langle T'\rangle \quad \text{if } T <: T' \\
w\langle S\rangle &<: w\langle S'\rangle \quad \text{if } S' <: S \\
ro\langle T\rangle &<: ro\langle T'\rangle \text{if } T <: T'
\end{aligned}$$

out generating errors only if its type environment Γ contains at location k the corresponding type of b, i.e., $\Gamma(k, b)$ should be defined. Runtime errors are presented at the end of Section 2.3, where Table 9 contains the rules of the error system.

In Dπ the resources are accumulated, but they can never be discarded. We extend the channel types of $D\pi$ with timers of form Δt. These timers define the existence of the channel types inside the type environment. The timers decrease with each "tick" of the universal clock (we assume that we have an universal clock). Communication actions can be performed along a channel until the timer on its type has expired. After expiration, the channel capabilities are discarded and any communication would generate a runtime error. Timers are created once with the channel types, and they are activated when capabilities are added to the type environment.

In our approach a process can move to a certain location, and wait for a period of time to establish a communication on a particular channel (a fixed local resource) with a complementary process. It is necessary to offer capabilities as $r\langle T\rangle$, $w\langle T\rangle$, and $ro\langle T\rangle$ for these fixed resources in order to restrict the actions performed by a process. The capabilities for the locations, and the capabilities for the channels from the type environment impose to a process what actions are allowed to be executed at each location. An example of a type environment is:

$$\Gamma = \{l : loc\{a : A, b : B\}, k : loc\{a : A'\}\}$$

where we denote by $\Gamma(k)$ the type $loc\{a : A'\}$ of location k, and by $\Gamma(l, b)$ the channel type B of the channel b located at l. The process of accumulating capabilities is made explicit by using *environment extensions*. We denote by $\Gamma\{k : K\}$ an environment Γ extended with a new location k of type K.

Moreover, considering Γ as above, we can extend the type environment with a new channel c located at k by

$$\Gamma\{c@k : B'\} = \{l : loc\{a : A, b : B\}, k : loc\{a : A', c : B'\}\}$$

When a process receives new channel names together with their associated types, capabilities for the new names become available (are added to the type environment of the process). As an example, let us suppose a process receiving a name of a located channel $c@k$ with channel type B' through an input channel with *reading* capability. The type of the new channel is added to the type environment at the corresponding location type of $k : loc\{\ldots\}$. It means that now the process knows about the new channel, and gains the capability to communicate through the accumulated channel c according to type B'.

A *subtyping relation* ($<:$) is introduced to compare type environments. If we consider two type environments $\Gamma = \{l : loc\{\tilde{a} : \tilde{A}, \tilde{b} : \tilde{B}\}\}$ and $\Gamma' = \{l : loc\{\tilde{a} : \tilde{A}\}\}$, then we have $\Gamma <: \Gamma'$ according to the definition in the second column of Table 2. Comparing type environments Γ and Γ', we see that an environment with more capabilities (Γ) is a subtype of an environment with less capabilities (Γ'). The reason for such an interpretation of the subtyping relation is that Γ' is more restrictive than Γ. The subtyping relation represents the inverse of the subset relation from the set theory; if we consider the type environments as sets of location types, the relation above becomes $\Gamma \supseteq \Gamma'$.

We extend both the *partial meet* $\sqcap$ and *partial join* $\sqcup$ operators of $D\pi$ with the new channel capability $ro\langle\,\rangle$. Intuitively the partial meet operator behaves as the union operator of the set theory, and the partial join operator behaves as the intersection operator. We denote by $a : - \notin K$ the fact that in the location type K there is no channel type A for channel a such that $a : A \in K$. We denote by γ any of the location capabilities *go* or *newch*.

Table 3: *Partial meet operator for locations:*

$$\begin{aligned} K \sqcap K' = \{&\gamma \mid \gamma \in K \text{ or } \gamma \in K'\} \\ \cup\, \{&a : A \mid a : A \in K \text{ and } a : - \notin K'\} \\ \cup\, \{&a : A' \mid a : - \notin K \text{ and } a : A' \in K'\} \\ \cup\, \{&a : A'' \mid a : A \in K \text{ and } a : A' \in K' \text{ and } A'' = A \sqcap A'\} \end{aligned}$$

The partial meet operator for location types $K \sqcap K'$ is undefined if and only if there exists a channel name a such that $a : A \in K$, $a : A' \in K'$ and $A \sqcap A'$ is undefined (see Table 4 for the definition of $\sqcap$ for channel types).

The method of removing capabilities is formalised by a binary *subtraction operator* $\setminus_\Delta$ defined by using a *join* operator $\sqcup$ (see Table 5), and a *symmetrical difference* operator denoted by $\setminus$ similar to the one defined in set theory (in our case it is applied to type environments). We write $\setminus_\Delta$ for the operation of removing from the first type environment all the types contained in the second type environment. We denote by $\mathcal{E}$ the set of type environments. The subtraction operator $\setminus_\Delta$ described above is defined as $\setminus_\Delta : \mathcal{E} \times \mathcal{E} \to \mathcal{E}$ where

$\Gamma \setminus_\Delta \Gamma' = \Gamma \sqcup (\Gamma \setminus \Gamma')$. If we consider two type environments

$$\Gamma = \{loc\{a : A, b : B\}\} \quad \text{and} \quad \Gamma' = \{loc\{b : B, c : C\}\},$$

each composed of one location type with two channel types, then by applying the subtraction operator $\setminus_\Delta$ we obtain

$$\Gamma \setminus_\Delta \Gamma' = loc\{a : A, b : B\} \sqcup (loc\{a : A, b : B\} \setminus loc\{b : B, c : C\}) = loc\{a : A\}$$

Table 4: *Partial meet operator for channel types:*

Partial meet operator for channel types ($A \sqcap A'$) is undefined iff:

$r\langle T\rangle \in A$ and $r\langle T\rangle \in A'$ and $T \sqcap T'$ undefined

$ro\langle T\rangle \in A$ and $ro\langle T\rangle \in A'$ and $T \sqcap T'$ undefined

$w\langle S\rangle \in A$ and $w\langle S'\rangle \in A'$ and $S \sqcup S'$ undefined

$r\langle T\rangle \in A$ and $w\langle S'\rangle \in A'$ and $S' \not<: T$

$w\langle S\rangle \in A$ and $r\langle T'\rangle \in A'$ and $S \not<: T'$

$ro\langle T\rangle \in A$ and $w\langle S'\rangle \in A'$ and $S' \not<: T$

$w\langle S\rangle \in A$ and $ro\langle T'\rangle \in A'$ and $S \not<: T'$

$ro\langle T\rangle \in A$ and $r\langle T'\rangle \in A'$ and $T' \setminus_\Delta T$ undefined

$r\langle T\rangle \in A$ and $ro\langle T'\rangle \in A'$ and $T \setminus_\Delta T'$ undefined

The definition

$$\begin{aligned}
A \sqcap A' = {} & \{ro\langle T\rangle \mid ro\langle T\rangle \in A \text{ and } ro\langle -\rangle \notin A'\} \\
\cup {} & \{ro\langle T''\rangle \mid ro\langle T\rangle \in A \text{ and } ro\langle T'\rangle \in A' \text{ and } T'' = T \sqcap T'\} \\
\cup {} & \{w\langle S\rangle \mid w\langle S\rangle \in A \text{ and } w\langle -\rangle \notin A'\} \\
\cup {} & \{w\langle S''\rangle \mid w\langle S\rangle \in A \text{ and } w\langle S'\rangle \in A' \text{ and } S'' = S \sqcup S'\} \\
\cup {} & \{ro\langle T'\rangle \mid r\langle T\rangle \in A \text{ and } ro\langle T'\rangle \in A' \} \\
\cup {} & \{r\langle T\rangle \mid r\langle T\rangle \in A \text{ and } ro\langle -\rangle \notin A' \text{ and } r\langle -\rangle \notin A' \text{ or} \\
& \qquad r\langle T\rangle \in A \text{ and } ro\langle T'\rangle \in A', r\langle -\rangle \notin A' \text{ and} \\
& \qquad\qquad T \sqcup T' = \emptyset \text{ or undefined}\} \\
\cup {} & \{r\langle T''\rangle \mid r\langle T\rangle \in A \text{ and } ro\langle -\rangle \notin A' \text{ and } r\langle T'\rangle \in A' \text{ and } T'' = T \sqcap T' \text{ or} \\
& \qquad r\langle T\rangle \in A \text{ and } ro\langle S\rangle \in A' \text{ and } r\langle T'\rangle \in A' \text{ and} \\
& \qquad\qquad T'' = T \sqcap T' \text{ and } T \sqcup S = \emptyset \text{ or undefined } \} \\
\cup {} & \{r\langle T''\rangle \mid r\langle T\rangle \in A \text{ and } ro\langle T'\rangle \in A' \text{ and } r\langle -\rangle \notin A' \text{ and } T'' = T \setminus_\Delta T' \text{ or} \\
& \qquad r\langle T\rangle \in A \text{ and } ro\langle T'\rangle \in A' \text{ and } r\langle S\rangle \in A' \text{ and} \\
& \qquad\qquad T'' = T \setminus_\Delta T' \text{ and } T \sqcup S = \emptyset \text{ or undefined } \}
\end{aligned}$$

plus all other natural cases resulted from swapping A with A'

A process which has a channel type with a capability $ro\langle T\rangle$ can receive only messages of type T (or any subtype of T) without generating errors. When the type of the channel is extended with the capability $ro\langle T'\rangle$, then the process is able to receive messages of a less restrictive type $T'' = T \sqcap T'$. We solve the possible conflict between $r\langle\,\rangle$ and $ro\langle\,\rangle$ by providing a higher priority to $ro\langle\,\rangle$ capability (because it is more restrictive than $r\langle\,\rangle$). In consequence, $ro\langle\,\rangle$ keeps its types and $r\langle\,\rangle$ loses them in favour of $ro\langle\,\rangle$ whenever $r\langle T\rangle$ and $ro\langle T'\rangle$ overlap (i.e., $T \sqcup T' \neq \emptyset$). When extending the writing capability $w\langle S\rangle$ with a new capability $w\langle S'\rangle$, the channel becomes more restricted, having the capability $w\langle S''\rangle$ where $S''{=}S \sqcup S'$.

We denote by $r\langle -\rangle \notin A$ the fact that there is no type T such that $r\langle T\rangle \in A$. The notations $w\langle -\rangle \notin A$ and $ro\langle -\rangle \notin A$ are defined similar.

Table 5: *The Join operator*

$$
\begin{array}{rcll}
K \sqcup K' & = & \{\gamma & \mid \gamma \in K \text{ and } \gamma \in K'\} \\
 & \cup & \{a : A'' & \mid a : A \in K \text{ and } a : A' \in K' \text{ and } A'' = A \sqcup A'\} \\
A \sqcup A' & = & \{r\langle T''\rangle & \mid r\langle T\rangle \in A \text{ and } r\langle T'\rangle \in A' \text{ and } T'' = T \sqcup T'\} \\
 & \cup & \{ro\langle T''\rangle & \mid ro\langle T\rangle \in A \text{ and } ro\langle T'\rangle \in A' \text{ and } T'' = T \sqcup T'\} \\
 & \cup & \{w\langle S''\rangle & \mid w\langle S\rangle \in A \text{ and } w\langle S'\rangle \in A' \text{ and } S'' = S \sqcap S'\}
\end{array}
$$

Proposition 2.2

i. $(\mathcal{E}, \sqcup)$ *is a commutative monoid;*

ii. $(\mathcal{E}, \setminus)$ *is a commutative group;*

iii. $\sqcup$ *is distributive over* $\setminus$*, and* $(\mathcal{E},\setminus,\sqcup)$ *is a ring.*

Proof: It is easy to observe that $\sqcup$ and $\setminus$ are commutative, and the empty environment is the identity element. The distributivity of $\sqcup$ over $\setminus$ can be simply verified by translating the set operators into boolean operators, and using the truth tables. □

We define a cleanup function ψ which changes the type environments according to the passage of time. It decreases the timers of the channel types, and removes the types with an expired timer. It also removes location types with only *go* capability.

Definition 2.3 (Cleanup function)
$\psi : LP_\Gamma \to LP_\Gamma$ is defined over the set of tagged located processes LP_Γ by:

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

where l can be any location of a distributed system, Γ' is obtained from Γ such that every channel type $res\{\tilde{\alpha}\}\Delta t$ with $t > 1$ and $t \neq \infty$ is changed to $res\{\tilde{\alpha}\}\Delta(t-1)$, and every $res\{\tilde{\alpha}\}\Delta 1$ is removed. Moreover, location types $loc\{go\}$ are removed.

By removing channel types from Γ, we get Γ' where it is possible to have location types having only *go* capabilities. We consider these location types as *empty* because the only allowed action is a movement. Even if we have $k : loc\{go\}$ in Γ', and a sequence of movements for a process $go\,k.go\,l.P$, this process can be reduced to $go\,l.P$ because we can avoid the intermediary code migration to location k without losing any useful effect. Therefore ψ removes $k : loc\{go\}$ from Γ'. A process moving to a location l having the type $loc\{go\}$ has no other capability, thus when performing any action (communication or channel creation) it gives rise to *runtime errors*.

For simulating the passage of time we use a *time-stepping function* ϕ defined over the set $\mathcal{P}_l$ of processes running at an arbitrary location l. The possible communications are performed at each tick of the universal clock; active channels are those which could be involved in these communications. The time-stepping function affects the active channels which do not communicate at that tick; the timers of the affected channels are decreased by one unit of time. The channels involved in communication disappear together with their timers. In the definition of the time-stepping function ϕ, we omit the channel type and the transmitted message in the input and output processes in order to simplify the presentation.

Definition 2.4 (Time-stepping function $\phi : \mathcal{P}_l \to \mathcal{P}_l$)

$$\phi(P) = \begin{cases} a^{\Delta(t-1)}.(R,Q) & \text{if } P = a^{\Delta t}.(R,Q),\ t > 1 \text{ and } t \neq \infty \\ \mathrm{Q} & \text{if } P = a^{\Delta t}.(R,Q),\ t \leq 1 \\ \phi(R) \,|\, \phi(Q) & \text{if } P = R \,|\, Q \\ (\nu\, a : A)\phi(R) & \text{if } P = (\nu\, a : A)R \\ \mathrm{P} & \text{otherwise} \end{cases}$$

We also define a *tagged time-stepping function* ϕ_Δ taking care of the missing types. ϕ_Δ is a *global* function defined by using the *local* function ϕ.

Definition 2.5
Tagged time-stepping function $\phi_\Delta : LP_\Gamma \to LP_\Gamma$ is defined by using ϕ:

$$\phi_\Delta(l[[P]]_\Gamma) = \begin{cases} l[[\phi(P)]]_{\Gamma'} & \text{if } P = a^{\Delta t}.(R,Q),\ \mathrm{t}{>}1 \text{ and } \mathrm{t}{\neq}\infty \\ & \text{or if } P = a^{\Delta t}.(R,Q),\ t \leq 1 \\ l[[Q]]_{\Gamma'} & \text{if } P = a^{\Delta t}.(R,Q),\ \mathrm{t}{>}1 \\ & \quad \text{and } \Gamma \not<: \Gamma(l,a) \\ \phi_\Delta(l[[R]]_\Gamma) \,|\, \phi_\Delta(l[[Q]]_\Gamma) & \text{if } P = R \,|\, Q \\ (\nu\, a@l : A)\phi_\Delta(l[[R]]_{\Gamma\{a@l:A\}}) & \text{if } P = (\nu\, a : A)R \\ l[[\phi(P)]]_{\Gamma'} & \text{otherwise} \end{cases}$$

where Γ' is obtained by applying the cleanup function ψ.

Tagged time-stepping function ϕ_Δ is applied to tagged located processes $(l[[P]]_\Gamma)$; it also changes the type environment of the located process by applying the cleanup function ψ.

The static semantics of $tD\pi$ is defined as a set of inference rules which describe the relationship between expressions and their corresponding types. In this paper we consider the type environment as a mapping from free names to types. A type environment is associated with each located process to restrict the range of resources it may access. The typing rules describe the behaviour of a process with respect to its types. A typing system is used to decide the *well-typedness* of the processes. Syntactically we write $\Gamma \vdash P$, and say that *a process P is well-typed with respect to a type environment* Γ. We also write $\Gamma \vdash_k P$ and say that P *is well-typed to run at location* k.

Table 6: *The Typing System (Typing rules)*

Processes

(T-R)

$$\frac{\Gamma \vdash_l a : res\{r\langle T\rangle\}\Delta t \quad fv(X) \cap fv(\Gamma) = \emptyset \quad \Gamma\{X@l : T\} \vdash_l P \quad \Gamma \vdash_l Q}{\Gamma \vdash_l a^{\Delta t}?(X : T).(P, Q)}$$

(T-RO)

$$\frac{\Gamma \vdash_l a : res\{ro\langle T\rangle\}\Delta t \quad fv(X) \cap fv(\Gamma) = \emptyset \quad \Gamma \vdash_l P \quad \Gamma \vdash_l Q}{\Gamma \vdash_l a^{\Delta t}?(X : T).(P, Q)}$$

(T-W)

$$\frac{\Gamma \vdash_l a : res\{w\langle T\rangle\}\Delta t \quad \Gamma \vdash_l v : T \quad \Gamma \vdash_l P \quad \Gamma \vdash_l Q}{\Gamma \vdash_l a^{\Delta t}!\langle v\rangle.(P, Q)}$$

(T-NEWCH)

$$\frac{\Gamma(l) <: loc\{newch\} \quad a \notin fn(\Gamma) \quad \Gamma\{a@l : A\} \vdash_l P}{\Gamma \vdash_l (\nu\, a : A)P}$$

(T-STR)

$$\frac{\Gamma \vdash_l P \quad \Gamma \vdash_l Q}{\Gamma \vdash_l stop, P \,|\, Q, {}^*P}$$

(T-GO)

$$\frac{\Gamma(k) <: loc\{go\} \quad \Gamma \vdash_k P}{\Gamma \vdash_l go\, k.P}$$

(T-R$_{new}$)

$$\frac{a : - \notin \Gamma(l) \quad \Gamma \vdash_l Q}{\Gamma \vdash_l a^{\Delta t}?(X : T).(P, Q)}$$

(T-W$_{new}$)

$$\frac{a : - \notin \Gamma(l) \quad \Gamma \vdash_l Q}{\Gamma \vdash_l a^{\Delta t}!\langle v\rangle.(P, Q)}$$

Located Processes

(N-RUN)

$$\frac{\Delta \vdash_l P \quad \Gamma <: \Delta}{\Gamma \Vdash l[[P]]_\Delta}$$

(N-SRT)

$$\frac{\Gamma \Vdash M \quad \Gamma \Vdash N}{\Gamma \Vdash 0,\ M \,|\, N}$$

(N-NEWCH)

$$\frac{\Gamma(l) <: loc\{newch\} \quad a \notin fn(\Gamma) \quad \Gamma\{a@l : A\} \Vdash N}{\Gamma \Vdash (\nu a@l : A)N}$$

In Table 6 we give the rules for the typing system of $tD\pi$. Considering the rules (T-R$_{new}$) and (T-W$_{new}$), we observe that the intuitive notion of *well-typedness* from $D\pi$ is no longer valid in $tD\pi$. In our calculus we accept tagged located processes with missing channel types (the types are removed with the passage of time), and these processes do not generate errors.

In order to say that $a^{\Delta t}!\langle v\rangle.(R, Q)$ is well-typed to run at location k with respect to type environment Γ, the following statements should hold:

- $\Gamma \vdash_k v : T$ which means that v is a value of type T at location k;
- $\Gamma \vdash_k a : res\{w\langle T\rangle\}\Delta t'$ which means that channel a exists at location k, and may send values of type T for t' units of time;
- $\Gamma \vdash_k R$; $\Gamma \vdash_k Q$ which means that both R and Q are well-typed to run at location k.

For a tagged located process $k[[P]]_\Delta$, the well-typedness relation is denoted by $\Vdash$ and is defined by using the well-typedness relation $\vdash_k$ for a process P running at location k (see rule (N-RUN) in Table 6).

If a process communicates on a channel for which it has no capability, it can still be well-typed if the alternative process Q is well-typed. We call this second process the *safety process*. This behaviour is reflected in one of the cases in the definition of ϕ_Δ.

We can imagine the process action flow as a binary decision tree because of the decision-like syntax of the channels. At each time step one of the following alternatives must be chosen for an action: communication action, timer expiration or move action (see Section 2.3 for the extension of the *go* operator with a choice syntax). An alternate definition for well-typedness of processes is: A process is well-typed if in the action flow tree there exists a path from the root to a leaf which does not generate a runtime error.

Proposition 2.6 *[6] (Weakening property)*

(a) *If* $\Gamma \Vdash N$ *and* $\Delta <: \Gamma$ *then* $\Delta \Vdash N$. *(for tagged located processes)*

(b) *If* $\Gamma \vdash_k P$ *and* $\Delta <: \Gamma$ *then* $\Delta \vdash_k P$. *(for processes)*

(c) *If* $\Gamma \vdash_k a : A$ *and* $\Delta <: \Gamma$ *then* $\Delta \vdash_k a : A$. *(for channels)*

The weakening property extends the well-typedness property of the processes from a given type environment Γ to a less restrictive environment Δ (which has more capabilities). The second statement can be read as: if P is well-typed to run at location k with respect to a type environment Γ and Δ is a subtype of Γ, then P is also well-typed to run at location k with respect to the type environment Δ.

Since the cleanup function ψ changes the type environment Δ by removing channel and location types, we are interested in whether the process is still well-typed under the new type environment Δ'.

Lemma 2.7 *(Well-typedness is preserved by the cleanup function)*
If $\Gamma \Vdash l[[P]]_\Delta$, *then* $\Gamma \Vdash \psi(l[[P]]_\Delta)$. *In other words, if* $\Gamma \Vdash l[[P]]_\Delta$, *then*

$\Gamma \Vdash l[[P]]_{\Delta'}$ *where* Δ' *is obtained by removing channel and location types from the type environment* Δ.

Proof: The proof proceeds by induction on the structure of P, having a case for each process expression. We give here only the most interesting and significant cases. For a complete proof see the online technical report [12].

Case inferred from (*Composition*: R | Q). By the equivalence rule (S_Γ-SPLIT) we have $\Gamma \Vdash l[[R]]_\Delta \,|\, l[[Q]]_\Delta$ which, by rule (N-STR) for located processes, is transformed into $\Gamma \Vdash l[[R]]_\Delta$ and $\Gamma \Vdash l[[Q]]_\Delta$. Applying the induction hypothesis, we obtain $\Gamma \Vdash \psi(l[[R]]_\Delta)$ and $\Gamma \Vdash \psi(l[[Q]]_\Delta)$ which, by applying ψ, become $\Gamma \Vdash l[[R]]_{\Delta'}$ and $\Gamma \Vdash l[[Q]]_{\Delta'}$. For both processes we have the same Δ' because the application of ψ to the tagged located processes takes into account only the type environment, and in our case the type environment is the same Δ. By applying the relation (S_Γ-SPLIT) we get the result $\Gamma \Vdash l[[R \mid Q]]_{\Delta'}$ which means $\Gamma \Vdash \psi(l[[R \mid Q]]_\Delta)$.

Case inferred from (*Restriction*: (ν a:A)Q). From (N-RUN) we have $\Delta \vdash_l (\nu a : A)Q$ and $\Gamma <: \Delta$. By (T-NEWCH) we infer $\Delta\{a@l : A\} \vdash_l Q$ and we also have $\Gamma\{a@l : A\} <: \Delta\{a@l : A\}$. By applying the weakening property 2.6 we infer that $\Gamma\{a@l : A\} \Vdash l[[Q]]_{\Delta\{a@l:A\}}$. Applying the induction hypothesis, we get $\Gamma\{a@l : A\} \Vdash \psi(l[[Q]]_{\Delta\{a@l:A\}})$ which is equivalent to $\Gamma\{a@l : A\} \Vdash l[[Q]]_{\Delta'\{a@l:A\}}$ because the application of the function ψ does not affect the new name a. We apply again the (T-NEWCH) rule obtaining $\Gamma \Vdash (\nu a : A)l[[Q]]_{\Delta'\{a@l:A\}}$ which is structurally equivalent to $\Gamma \Vdash l[[(\nu a : A)Q]]_{\Delta'}$.

Case inferred from (*Movement*: $go\, k.Q$). In the same way of reasoning as before we have $\Delta \vdash_l go\ k.Q$ and $\Gamma <: \Delta$. By (T-GO) we get $\Delta \vdash_k Q$ and $\Delta(k) <: loc\{go\}$. Using (N-RUN) we have $\Gamma \Vdash k[[Q]]_\Delta$ which by induction implies $\Gamma \Vdash k[[Q]]_{\Delta'}$. We now infer that $\Delta' \vdash_k Q$ and $\Gamma <: \Delta'$ are true. By application of the ψ function, the capability of the process to move to location k cannot be lost. This means that $\Delta'(k) <: loc\{go\}$ holds and, together with what we obtained above and by using the rule (T-GO), we have $\Delta' \vdash_l go\, k.Q$ and again $\Gamma \Vdash l[[go\, k.Q]]_{\Delta'}$. This is another syntactic form of what we were looking for, namely $\Gamma \Vdash \psi(l[[Q]]_\Delta)$.
The proof proceeds in the same manner if instead of (T-GO) we use the new rules (T-GO1) and (T-GO2) defined in Section 2.3.

Case inferred from (*Input*: $a^{\Delta t}?(X : T).(R, Q)$). If we consider that channel a has the type $ro\langle\,\rangle$, then from $\Delta \vdash_l a^{\Delta t}?(X : T).(R, Q)$ and by using (T-RO) we have the following statements: $\Delta \vdash_l a : res\{ro\langle T\rangle\}$, $fv(X) \cap fv(\Delta) = \emptyset$, $\Delta \vdash_l R$ and $\Delta \vdash_l Q$. Applying the induction hypothesis, the last two statements are transformed into $\Gamma \Vdash l[[R]]_\Delta$ and $\Gamma \Vdash l[[Q]]_\Delta$ which provide the following two true statements: $\Gamma \Vdash \psi(l[[R]]_\Delta)$ and $\Gamma \Vdash \psi(l[[Q]]_\Delta)$. This means that three ($fv(X) \cap fv(\Delta) = \emptyset$, $\Delta' \vdash_l R$ and $\Delta' \vdash_l Q$) of the four statements needed by (T-RO) are true. If the cleanup function does not remove the type of the input channel from the capability set, then it is valid in the new environment

Δ'. Thus we can apply (T-RO), and obtain $\Gamma \Vdash l[[a^{\Delta t}?(X:T).(R,Q)]]_{\Delta'}$. On the other hand, if the type of the active channel is missing, we can use the rule (T-R$_{new}$) and obtain the same result as before which is equivalent to the desired result, namely $\Gamma \vdash \psi(l[[a^{\Delta t}?(X:T).(R,Q)]]_{\Delta})$.

The cases for *Output*, *Replication* and *Termination* are natural, and they follow the proof steps of the cases presented above. □

The following lemma shows that the passage of time does not interfere with the typing system. The lemma states that if a tagged located process is well-typed with respect to a type environment Γ, then the application of the tagged time-stepping function ϕ_Δ preserves its well-typedness property.

Lemma 2.8 *(Tagged time passage)*

$$\textit{If } \Gamma \Vdash l[[P]]_\Delta, \textit{ then } \Gamma \Vdash \phi_\Delta(l[[P]]_\Delta).$$

Proof: We use induction on the inference depth of $\Gamma \Vdash l[[P]]_\Delta$. From the hypothesis we derive that $\Delta \vdash_l P$ by (N-RUN), and $\Gamma <: \Delta$. We get $\Gamma \vdash_l P$ by using the weakening property. The proof continues by considering a case for each line in the definition of ϕ_Δ.

Case inferred from $(P = R \,|\, Q)$. Using (T-STR) we have $\Delta \vdash_l R$ and $\Delta \vdash_l Q$ which is equivalent to $\Delta \vdash l[[R]]$; by Lemma 2.6 we get $\Gamma \Vdash l[[R]]_\Delta$. The same result is obtained for process Q. By applying the induction hypothesis, we get $\Gamma \Vdash \phi_\Delta(l[[R]]_\Delta)$ and $\Gamma \Vdash \phi_\Delta(l[[Q]]_\Delta)$. These lead to the desired result, by the application of ϕ_Δ to $R \,|\, Q$, *i.e.* $\Gamma \Vdash \phi_\Delta(l[[P]]_\Delta)$.

Case inferred from $(P = a^{\Delta t}.(R,Q),\ t \leq 1)$. We have two subcases, one when a is an input channel, and another when a is an output channel. The result of the application of ϕ_Δ to P is $l[[Q]]_{\Delta'}$ (with Δ' obtained by applying the cleanup function ψ) because $t \leq 1$. Let us consider that a is an *output* channel, and thus $\Delta \vdash_l a^{\Delta t}!\langle v\rangle.(R,Q)$ and $\Gamma <: \Delta$. Using (T-W), we get $\Delta \vdash_l Q$, and by Lemma 2.7 we get $\Delta' \vdash_l Q$. Since $\Gamma <: \Delta <: \Delta'$, we infer $\Gamma \Vdash l[[Q]]_{\Delta'}$. A similar proof is obtained when we consider an *input* channel, by using the rules corresponding to the type of the input channel.

Case inferred from $(P = a^{\Delta t}.(R,Q),\ \mathrm{t}{>}1$ and $\Gamma \not<: \Gamma(l,a))$. This case is similar to the previous one, but instead of using the normal typing rules we use (T-R$_{new}$) and (T-W$_{new}$) just because the capabilities of a are not included in the type environment.

Case inferred from $(P = a^{\Delta t}.(R,Q),\ \mathrm{t}{>}1$ and $\mathrm{t} \neq \infty)$. For this case we consider the *input* expression, namely $\Delta \vdash_l a^{\Delta t}?(X:T).(R,Q)$. In this case ϕ_Δ decreases the channel timer from $a^{\Delta t}$ to $a^{\Delta t-1}$. From the point of view of the typing system, the processes $a^{\Delta t}?(X:T).(R,Q)$ and $a^{\Delta t-1}?(X:T).(R,Q)$ are the same, and we can apply Lemma 2.7 and get $\Delta' \vdash_l a^{\Delta t}?(X:T).(R,Q)$. Since $\Gamma <: \Delta <: \Delta'$, we get the conclusion $\Gamma \Vdash l[[a^{\Delta t}?(X:T).(R,Q)]]_{\Delta'}$.

The case for the channel restriction is similar, and uses the typing rule

(T-NEWCH). □

Definition 2.9 We define a syntactic equivalence $\equiv$ over timed channels by

$$a_1^{\Delta t_1} \equiv a_2^{\Delta t_2} \text{ if and only if } a_1 = a_2 \text{ and } t_1 = t_2.$$

If the timers of the same channel name have different values, the corresponding processes have different behaviour. This aspect must be considered when defining timed bisimulations [1].

We define the *tagged structural equivalence* relation.

Table 7: *Tagged structural equivalence*

(S_Γ-GARBAGE)	$l[[stop]]_\Gamma$	$\equiv stop$
(S_Γ-SPLIT)	$l[[P \mid Q]]_\Gamma$	$\equiv l[[P]]_\Gamma \mid l[[Q]]_\Gamma$
(S_Γ-COPY)	$l[[*P]]_\Gamma$	$\equiv l[[P]]_\Gamma \mid l[[*P]]_\Gamma$
(S_Γ-NEW)	$l[[(\nu\, a : A)P]]_\Gamma$	$\equiv (\nu\, a@l : A)l[[P]]_\Delta$ if $a \notin fn(\Gamma) \cup \{l\}$ where $\Delta = \Gamma\{a@l : A\}$
(S_Γ-EXTR)	$M \mid (\nu a@k : A)N$	$\equiv (\nu a@k : A)(M, \mid N)$ if $a \notin fn(M)$
(S_Γ-ASSOC)	$l[[P]]_\Gamma \mid l[[Q \mid R]]_\Gamma$	$\equiv l[[P \mid Q]]_\Gamma \mid l[[R]]_\Gamma$
(S_Γ-COMMU)	$l[[P]]_\Gamma \mid l[[Q]]_\Gamma$	$\equiv l[[Q]]_\Gamma \mid l[[P]]_\Gamma$
(S_Γ-NEUTR)	$l[[P]]_\Gamma \mid stop$	$\equiv l[[P]]_\Gamma$

The subject reduction property states that well-typedness is preserved by reduction relation. This is a general approach in functional programming frameworks [4,11]. We are also interested to prove that the well-typedness property is preserved by structural equivalence relation. We present now such a result related to the structural equivalence relation. A more general subject reduction theorem is presented in Section 3.

If we have two tagged located processes which are structurally equivalent, and one of them is well-typed with respect to a type environment Γ, then the other process is also well-typed with respect to type environment Γ.

Theorem 2.10 *(Subject reduction for tagged equivalence relation)*
For all tagged located processes N, N' such that $N \equiv N'$,

$$\Gamma \Vdash N \text{ if and only if } \Gamma \Vdash N'.$$

Proof: We must consider all the equivalences given in Table 7.

Case inferred from (S_Γ-NEW). From hypothesis we have $\Gamma \Vdash l[[(\nu a : A)P]]_\Delta$, which means that $\Gamma <: \Delta$ and $\Delta \vdash_l (\nu a : A)P$. By using (T-NEWCH) we get $\Delta\{a@l : A\} \vdash_l P$. By applying (N-RUN) we get $\Delta\{a@l : A\} \vdash l[[P]]$, and together with $\Gamma\{a@l : A\} <: \Delta\{a@l : A\}$ we have $\Gamma\{a@l : A\} \Vdash l[[P]]_{\Delta\{a@l:A\}}$. We apply again (T-NEWCH) for tagged processes and get the result, namely $\Gamma \Vdash (\nu a@l : A)l[[P]]_{\Delta\{a@l:A\}}$.

Case inferred from (S_Γ-SPLIT). We start from $\Gamma <: \Delta$ and $\Delta \vdash_l P \,|\, Q$, and by using (T-STR) we get $\Delta \vdash_l P$ and $\Delta \vdash_l Q$. From $\Gamma <: \Delta$ and (N-RUN) we get $\Gamma \Vdash l[[P]]_\Delta$ and $\Gamma \Vdash l[[Q]]_\Delta$. We apply again (N-STR) and obtain the conclusion $\Gamma \Vdash l[[P]]_\Delta \,|\, l[[Q]]_\Delta$.

Case inferred from (S_Γ-COPY). This case follows the steps of the previous one, and we leave it as an exercise to the reader.

Case inferred from (S_Γ-EXTR). For this case we use the rules for located processes. Starting from $M \,|\, (\nu a@k : A)N$ and by using (N-STR), we get $\Gamma \Vdash M$ and $\Gamma \Vdash (\nu a@k : A)N$. $\Gamma \Vdash (\nu a@k : A)N$ together with (N-NEWCH) infer $\Gamma\{a@k : A\} \Vdash N$. By weakening, and because $a \notin fn(\Gamma)$, we get $\Gamma\{a@k : A\} \Vdash M$. We apply again (N-STR) and then (N-NEWCH), and we get the desired result $\Gamma \Vdash (\nu a@k : A)(M \,|\, N)$.

The cases inferred from (S_Γ-GARBAGE) and other rules are similar to the monoid laws of the π-calculus. □

2.3 Operational Semantics

We consider the tagged located processes ranged over by N and M, namely N and M can be thought as process expressions of form $l[[P]]_\Gamma$. We denote by $\not\to$ the fact that rules (R_Γ-COM1) and (R_Γ-COM2) cannot be applied. Using these notations, we give the following reduction rules providing an operational semantics for $tD\pi$.

Table 8: *Reduction relation of $tD\pi$*

$$(R_\Gamma\text{-GO})\ \frac{}{l[[go\,k.P]]_\Gamma \to \psi(k[[P]]_\Gamma)} \qquad (R_\Gamma\text{-IDLE})\ \frac{l[[P]]_\Gamma \not\to}{l[[P]]_\Gamma \to \phi_\Delta(l[[P]]_\Gamma)}$$

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

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

$$(R_\Gamma\text{-PAR})\ \frac{N \to N' \qquad M \to M'}{N \mid M \to N' \mid M'} \qquad (R_\Gamma\text{-RES})\ \frac{N \to N'}{(\nu\, a@l : A)N \to (\nu\, a@l : A)N'}$$

$$(R_\Gamma\text{-CONG})\ \frac{N \equiv N' \qquad N \to M \qquad M \equiv M'}{N' \to M'}$$

We have two communication rules which depend on the type of the communication 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, contrary to the behaviour of rule (R_Γ-COM1). The communication rules and (R_Γ-GO) do not enter under the scope of ϕ_Δ. In this case 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 and changes the state of the process. At each tick of the universal clock, the rule (R_Γ-IDLE) is applied to processes which do not enter any communication. When applying the rule (R_Γ-PAR), if process M does not have an internal communication reduction, then it is transformed into M' by rule (R_Γ-IDLE). The same argument is valid for N as well.

Removing location types from the type environment can lead to errors generated by *go* actions. We solve this problem by extending the syntax of *go* with a choice syntax similar to the one given for channels; therefore $go\, l.P$ becomes $go\, l.(P, Q)$. If $\Gamma(l)$ is not defined, then Q is executed. If the location type of l contains a capability *go*, then P is executed; otherwise, if the location type of l does not contain a capability *go*, an error is generated. We should change the corresponding typing rules where the operator *go* appears. Thus (T-GO) is translated into (T-GO1) and (T-GO2).

$$\text{(T-GO1)}\ \frac{k \notin dom(\Gamma) \qquad \Gamma \vdash_l Q}{\Gamma \vdash_l go\, k.(P, Q)} \qquad \text{(T-GO2)}\ \frac{\Gamma(k) : loc\{go\} \qquad \Gamma \vdash_k P}{\Gamma \vdash_l go\, k.(P, Q)}$$

A process P generating an error is denoted by $P \xrightarrow{err}$. The cases when a process generates a runtime error are defined by a set of rules in Table 9.

robj(), roobj(), wobj() are partial functions defined over the set of channel types, and returning the type of the corresponding channel capabilities. For example, considering a channel type $a : res\{w\langle T\rangle\}$ in the type environment Γ at location l, the application of $wobj(\Gamma(l, a))$ returns T. In order to derive a runtime error, the channel type or location type must be 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 process is chosen by ϕ_Δ.

The reduction rule (R_Γ-GO) cannot check if the type of the location is in the type environment, and consequently we change the time-stepping function ϕ_Δ by adding two more lines to its definition:

$$\begin{cases} k[[R]]_{\Gamma'} & \text{if } P = go\, k.(R, Q) \text{ and } \Gamma(k) <: loc\{go\} \\ l[[Q]]_{\Gamma'} & \text{if } P = go\, k.(R, Q) \text{ and } k \notin dom(\Gamma) \end{cases}$$

The rule (R_Γ-GO) is changed into $l[[go\, k.(P, Q)]]_\Gamma \rightarrow \phi_\Delta(l[[go\, k.(P, Q)]]_\Gamma)$ which is captured by the (R_Γ-IDLE) rule. A process of the form $go\, k.(P, Q)$ is beyond the scope of any of the reduction rules R_Γ, excepting (R_Γ-IDLE), and so ϕ_Δ is applied. This function applies one of its new lines, and changes the

Table 9: *Runtime errors*

$$\text{(E-GO)}\quad \frac{\Gamma(k) \text{ is defined and } \Gamma(k) \not<: loc\{go\}}{l[[go\,k.(P,Q)]]_\Gamma \xrightarrow{err}}$$

$$\text{(E-SUBC)}\quad \frac{\Gamma(l) \not<: loc\{newch\}}{l[[(\nu\, a : A)P]]_\Gamma \xrightarrow{err}}$$

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

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

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

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

process either by allowing the movement to the new location, or by choosing the safety process.

Regarding the behaviour of the $tD\pi$ system, we can say that a nondeterministic method is applied to select two interacting processes for each communication channel at each location of a distributed system. Afterwords the reduction rules are applied, and the communications are performed. ϕ_Δ is applied to the processes which do not enter in any communication. The type environments of the communicating processes are affected the application of ψ function. We can say that a system described with $tD\pi$ satisfies the following properties [5]:

- **Time Determinism**: at each time only one reduction rule can be applied. A possible problem could appear only if we apply R_Γ-IDLE when we can apply a communication rule. However this is not possible because R_Γ-IDLE is applied only if the process does not enter in any communication ($\not\rightarrow$).
- **Maximal Progress**: a process cannot delay if it can enter a communication.
- **Time Continuity**: to go from a process P at time t, to a process P'' at time $t + \Delta t$, we must go through all the intermediate time steps of the interval $[t, t + \Delta t]$.

Some papers which discuss the time problem in distributed systems consider a global clock synchronising all the timers. Recent work [9] on Network

Time Synchronisation Protocol (NTP) shows that it is possible to achieve time synchronisation in real applications. Having this technology we can suppose that the theoretical assumption about a universal clock is practical rather than speculation. Our global timing function ϕ_Δ has to apply the local time-stepping function ϕ for the locations of the distributed system. If we adopt the NTP synchronisation model, we can get a guaranteed frequency and local oscillator phase precision of no more than a few milliseconds, which in many cases is acceptable.

3 Soundness of $tD\pi$

Regarding the soundness of $tD\pi$, we follow a method based on *subject reduction* and *type safety* [4] used also in proving the soundness of $D\pi$. This is a syntactic approach, in contrast to other approaches based on denotational semantics or structural operational semantics.

Theorem 3.1 *(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: Part *(a)* is in fact Theorem 2.10; its proof is in Section 2.2.
Part *(b)* is similar to the result presented in [11] which asserts the consistency between the static and the dynamic semantics. We use the same technique, and proceed by induction on the depth of inference for $N \to N'$. We also use Lemma 2.7 which relates time and type environments, and Lemma 2.8 which relates time and communication channels. More details can be found in [12].

Case inferred from (R_Γ-IDLE). This is covered by Lemma 2.8.

Case inferred from (R_Γ-RES). From the hypothesis we know that $\Gamma \Vdash (\nu a@k : A)N$. This means that $\Gamma\{a@k : A\} \Vdash N$, and according to the induction hypothesis we have $\Gamma \Vdash N'$. Since $\Gamma\{a@k : A\} <: \Gamma$, then by applying the weakening property of Proposition 2.6 we get $\Gamma\{a@k : A\} \Vdash N'$. Simply applying again (N-NEWCH) we get $\Gamma \Vdash (\nu a@k : A)N'$.

Case inferred from (R_Γ-COM1 or R_Γ-COM2). These two related rules can be treated in the same way. Let us consider the first one. Starting from $\Gamma \Vdash l[[a^{\Delta t}!\langle v\rangle.(P, Q)]]_\Delta \mid l[[a^{\Delta t'}?(X : T).(P', Q')]]_{\Delta'}$ and applying (N-STR) we get $\Gamma \Vdash l[[a^{\Delta t}!\langle v\rangle.(P, Q)]]_\Delta$ *(*)* and $\Gamma \Vdash l[[a^{\Delta t'}?(X : T).(P', Q')]]_{\Delta'}$ *(**)*. By (N-RUN) and *(*)* we have $\Delta \vdash_l a^{\Delta t}!\langle v\rangle.(P, Q)$ and with *(**)* we have $\Delta' \vdash_l a^{\Delta t'}?(X : T).(P', Q')$. By applying (T-W) we get $\Delta \vdash_l P$ which together with (N-RUN) give the statement $\Gamma \Vdash l[[P]]_\Delta$. We also have $\Delta \vdash_l v : T$ and the subtyping reactions $\Gamma <: \Delta$, $\Gamma <: \Delta'$ which means that $\Delta(l, u)$ and $\Delta'(l, u)$ must agree on the type they use. So by weakening we get $\Delta'\{v@l : T\} \vdash_l v : T$.

Now it is the moment to consider the difference between (R_Γ-COM1) and (R_Γ-COM2), difference given by the typing rule used for the type of the in-

put channel. By applying (T-R) we get $\Delta'\{X@l : T\} \vdash_l P'$. We denote by Δ'' the type environment $\Delta'\{v@l : T\}$. Thus, by weakening we get $\Delta''\{X@l : T\} \vdash_l P'$, and we can use the substitution lemma of [6] to obtain $\Delta'' \vdash_l P'\{^v/_X\}$. However $\Gamma <: \Delta''$ and so, by applying (N-RUN), we get $\Gamma \Vdash l[[P'\{^v/_X\}]]_{\Delta'\{v@l:T\}}$. We apply Lemma 2.7 two times, and also (N-STR) to get the result $\Gamma \Vdash \psi(l[[P]]_\Gamma) \mid \psi(l[[P'\{^v/_X\}]]_{\Delta'\{v@l:T\}})$.
It is easy to prove the second inference for (R_Γ-COM2), but we have to pay attention to the rules we use, because the type of the channel is now different.

Case inferred from (R_Γ-PAR). We have $\Gamma \Vdash N \mid M$ which by applying (N-STR) gives us $\Gamma \Vdash N$ and $\Gamma \Vdash M$. We can also infer by induction that $\Gamma \vdash N'$. By Lemma 2.8 we have that $\Gamma \Vdash \phi(M)$, and we can apply again (N-STR) obtaining the result $\Gamma \Vdash N' \mid \phi(M)$. For the case when M reduces to M' by other rule than (R_Γ-IDLE) (i.e., it is not affected by the passage of time), the proof steps are easy to find (and left to the reader).

Thus we have concluded the subject reduction proof for the typing system with temporary resources. □

Subject reduction assures us that once well-typed, a process remains well-typed during its evolution. Note that well-typedness must be preserved by both equivalence rules and reduction rules. In the following we give a result of *type safety* which is necessary to have a complete proof of the soundness property of $tD\pi$. The *type safety* property states that if a system is well-typed, then it cannot generate runtime errors, and this is denoted by $P \overset{err}{\not\longmapsto}$.

Theorem 3.2 *(Type safety)*
We have $N \overset{err}{\not\longmapsto}$ *for all tagged located processes* N*, and all type environments* Γ *such that* $\Gamma \Vdash N$.

Proof: The outline of the proof follows a method which proves the contrapositive, namely *if* N *gives rise to a runtime error* ($N \overset{err}{\longmapsto}$) *then* N *cannot be well-typed under any type environment* Γ ($\Gamma \not\Vdash N$ for all Γ). In [4] the authors use the same statement as a lemma to prove that the *faulty expressions are untypable*. We use induction on the definition of the runtime errors, and have a proof case for each rule of Table 9.

Case inferred from (E-SND). The rule says that $l[[a^{\Delta t}!\langle v\rangle.(P,Q)]]_\Gamma \overset{err}{\longmapsto}$ *if* $\Gamma(l,a)$ *is defined and* $\Gamma(l,v) \not<: wobj(\Gamma(l,a))$. Let us consider that there is a type environment Δ such that the process generating a runtime error is well-typed under this environment, i.e., $\Delta \Vdash l[[a^{\Delta t}!\langle v\rangle.(P,Q)]]_\Gamma$. This means that $\Delta <: \Gamma$ and $\Gamma \vdash_l a^{\Delta t}!\langle v\rangle.(P,Q)$. Therefore there are two typing rules which can be applied, depending on the type of the output channel. If $a : - \notin \Gamma(l)$, then we have a contradiction with the fact that $\Gamma(l,a)$ *must be defined* from the definition of the rule. Otherwise we have to use rule (T-W), obtaining $\Gamma \vdash_l a : res\{w\langle T\rangle\}\Delta t$ and $\Gamma \vdash_l v : T$. Statement $\Gamma \vdash_l v : T$ implies that $\Gamma(l,v) <: T$. From $\Gamma \vdash_l a : res\{w\langle T\rangle\}\Delta t$ we get $\Gamma(l,a) = res\{w\langle T\rangle\}$ (by definition), which

by application of the function $wobj$ leads to $wobj(\Gamma(l,a)) = T$. Together with $\Gamma(l,v) <: T$, this leads us to the contradiction $\Gamma(l,v) <: wobj(\Gamma(l,a))$.

Case inferred from (E-GO). We have $l[[gok.(P,Q)]]_\Gamma \stackrel{err}{\longmapsto}$ if $\Gamma(k)$ is defined and $\Gamma(k) \not<: loc\{go\}$. We consider that there exists a type environment Δ such that $\Delta \Vdash l[[gok.(P,Q)]]_\Gamma$, and try to see if we can conclude a contradiction. If the location k is not defined in the type environment Γ, then we can use (T-GO1); however this would result in a contradiction. By using (T-GO2), we have $\Gamma(k) : loc\{go\}$ which means that $\Gamma(k) <: loc\{go\}$; we get again a contradiction. Therefore we have the following statement: *there is no type environment* Δ *such that* $\Delta \Vdash l[[gok.(P,Q)]]_\Gamma$ *and* $l[[gok.(P,Q)]]_\Gamma \stackrel{err}{\longmapsto}$.

Case inferred from (E-RCV). We consider that there exists a type environment Δ such that $\Delta \Vdash l[[a^{\Delta t}?(X:T).(P,Q)]]_\Gamma$. From this we have that $\Delta <: \Gamma$ and $\Gamma \vdash_l a^{\Delta t}?(X:T).(P,Q)$. If we consider our input channel to be *reading only*, then we apply the rule (T-RO) and we get $\Gamma \vdash_l a : res\{ro\langle T\rangle\}\Delta t$. We immediately have $\Gamma(l,a) = res\{ro\langle T\rangle\}$, and by applying the function $roobj$ we get $roobj(\Gamma(l,a)) = T$, and thus $roobj(\Gamma(l,a)) <: T$, contradicting the definition.

Case inferred from (E-COMM). We use the same method as before, and consider that there is a type environment Δ' such that $\Delta' \Vdash l[[a^{\Delta t}!\langle v\rangle.(P,Q)]]_\Gamma \mid l[[a^{\Delta t'}?(X:T).(P',Q')]]_\Delta$. By applying the rule (N-STR), and then (N-RUN), we get $\Delta' <: \Gamma$, $\Delta' <: \Delta$, and $\Gamma \vdash_l a^{\Delta t}!\langle v\rangle.(P,Q)$, $\Delta \vdash_l a^{\Delta t'}?(X:T).(P',Q')$. Using the rule (T-W), we get $\Gamma \vdash_l a : res\{w\langle T\rangle\}$ which means that $\Gamma(l,a) = res\{w\langle T\rangle\}$. We apply the function $wobj$ and get $wobj(\Gamma(l,a)) = T$ *(1)*. We suppose that the channel a under type environment Δ is an $r\langle\,\rangle$ channel, and infer from $\Delta \vdash_l a^{\Delta t'}?(X:T).(P',Q')$ that $\Delta \vdash_l a : res\{r\langle T\rangle\}$. As before, we can apply the function $robj$ and get $robj(\Delta(l,a)) = T$ *(2)*. From *(1)* and *(2)* we have the contradiction $wobj(\Gamma(l,a)) <: robj(\Delta(l,a))$.

Case inferred from (E-SUBC), (E-NEW), (E-PAR) and (E-STR). These rules are the same as in $D\pi$, and the proofs are natural. □

4 Conclusion

Timed systems represent an active field, and there are many papers devoted to this topic. In the following we compare our approach with a recent paper [7] having some common features. The authors introduce $web\pi$, a calculus for distributed systems with locations, and treat failures and time. They also use a time-stepping function to decrease the time stamps. Each location has a private clock, but the clocks are not synchronised by a universal clock. In $web\pi$ the time stamp is attached to a transaction expression as a timeout for an entire process (a series of actions). In our calculus, each channel has a private timer which measures the timeout for a communication, and not for a

series of communications. An important difference between our calculus and $web\pi$ is the possibility of $tD\pi$ to express resource access constraints by using a typing system.

Another timed extension of the π-calculus which shares common features with our calculus is presented in [8]. The main contribution of πRT-calculus is the introduction of a timeout operator. The behaviour of the timeout operator is the same as the behaviour of a timed channel in our calculus. The authors also adopt a discrete time domain and synchronisation with a global clock. Our calculus respects three of the time properties treated in πRT: time determinism, time continuity and maximal progress. The other time properties treated by the authors are specific to the design choices adopted in πRT.

The actions in our approach are atomic as in the both models above. The communication of a name, and the moving with the *go* operator are supposed to take no time. Instantaneous actions are also found in [2], where an extension of the π-calculus with time is studied. An extension with locations to this timed π-calculus (π_t) is introduced. However types are not taken into consideration.

Our calculus adds timers on output channels, and timers on channel types; these features appear to be new. The combination between the quantitative constraints imposed by timers and the resource access constraints imposed by the typing system provides modelling power to the new formalism $tD\pi$. We are interested in modelling molecular networks and biological system [3]. In molecular networks there are strict rules which determine the next reaction a molecule can take part in. These are based on reaction times, quantitative coefficients, putative times and other external stimuli. Time represents an important quantitative measure in molecular networks, able to impose strong constraints on the interactions between molecules or complexes.

References

[1] Bengtsson, J. and W. Yi, *Timed automata: Semantics, algorithms and tools*, in: J. Desel, W. Reisig and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, Lecture Notes in Computer Science **3098** (2004), pp. 87–124.

[2] Berger, M., "Towards Abstractions for Distributed Systems", Ph.D. Thesis, Imperial College, Department of Computing (2002).

[3] Ciobanu, G. and G. Rozenberg, editors, "Modelling in Molecular Biology", Natural Computing Series, Springer, (2004).

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

[5] Hennessy, M. and T. Regan, *A process algebra for timed systems*, Information and Computation **117** (1995), pp. 221–239.

[6] Hennessy, M. and J. Riely, *Resource Access Control in Systems of Mobile Agents*, Information and Computation **173(1)** (2002), pp. 82–120.

[7] Laneve, C. and G. Zavattaro, *Foundations of web transactions.*, in: V. Sassone, editor, *Foundations of Software Science and Computational Structures*, Lecture Notes in Computer Science **3441** (2005), pp. 282–298.

[8] Lee, J. and J. Zic, *On modeling real-time mobile processes*, in: *Proceedings 25th Australasian Conference on Computer Science* (2002), pp. 139–147.

[9] Mills, D., *A brief history of NTP time: memoirs of an internet timekeeper*, Computer Communication Review **33** (2003), pp. 9–21.

[10] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes (i-ii)*, Information and Computation **100** (1992), pp. 1–77.

[11] Milner, R. and M. Tofte, *Co-induction in relational semantics*, Theoretical Computer Science **87** (1991), pp. 209–220.

[12] Prisacariu, C. and G. Ciobanu, *Timed distributed π-calculus*, Technical Report FML-05-01, Formal Methods Laboratory, Institute of Computer Science, Romanian Academy (2005).