Mobile ambients with time constraints

B. Aman and G. Ciobanu

Approved for public release; further dissemination unlimited.

Prepared by
Formal Methods Laboratory.

Series of
Technical Reports (Institute of Computer Science Iași)

ISSN 1842 - 1490
FML is the Formal Methods Laboratory from the Institute of Computer Science of the Romanian Academy and is located in Iași. It is a research institution aimed at developing new formalisms for challenging open problems in computer science, systems biology and other emerging research fields. FML is headed by Dr. Gabriel Ciobanu.

IIT is the Institute of Computer Science of the Romanian Academy and is concerned with basic research in computer science.

Romanian Academy of Science is the supreme forum of science from Romania. Calea Victoriei 125, Sector 1, 010071, Bucharest, ROMANIA.

**Telephone:** +40 21 2128640  
**Telefax:** +40 21 2116608  
**Web:** [http://www.academiaromana.ro/](http://www.academiaromana.ro/)

Printed in Romania and reproduced directly from the authors original copy.
Mobile ambients with time constraints

by

Bogdan Aman 1
Institute of Computer Science,
Romanian Academy, Iaşi
baman@iit.tuiasi.ro

Gabriel Ciobanu 2,3
Institute of Computer Science,
Romanian Academy, Iaşi
gabriel@iit.tuiasi.ro

ABSTRACT

Mobile ambients calculus is a calculus for mobile computing able to express local communications inside ambients, as well as movement and dissolution of ambients by consuming capabilities. We add timers to communication channels, capabilities and ambients. We provide a static semantic given by a typing system, and an operational semantics of the new calculus given by a reduction relation. The passage of time is given by a (discrete) time stepping function. We prove that structural congruence and passage of time do not interfere with the typing system. A subject reduction result assures that once well-typed, an ambient remains well-typed. We provide some bisimulation relations with regard to the passage of time. A timed extension of the cab protocol illustrates how the new formalism is working.

1Member of Formal Methods Laboratory (FML)
2Head of Formal Methods Laboratory (FML)
3Contact person
# Contents

1 Introduction ........................................... 2

2 Mobile Ambients ...................................... 2

3 Mobile ambients with time constraints .............. 4
  3.1 Semantics .......................................... 5
  3.2 Well-typed relation and results .................. 8
  3.3 Properties related to $\phi_\Delta$ .................. 12

4 Example ................................................. 15

5 Conclusion ............................................. 18
1 Introduction

Ambient calculus [4] is a formalism for describing distributed and mobile computation in terms of ambients, named collections of running processes and nested subambients. In contrast with previous formalisms for mobile processes such as the π-calculus [15], whose computational model is based on the notion of communication, the MA computational model is based on the notion of movement. An ambient, which is a named location, is the unit of movement and processes within the same ambient may exchange messages; mobility is controlled by the capabilities in, out, open. Several variants of the ambient calculus have been proposed so far [3, 11, 13] by adding and/or retracting features from the original calculus.

In [4] is expressed the fact that is convenient to model long-range communication as the movement of “messenger” ambients that cross administrative boundaries. In TCP/IP because UDP (User Data Protocol) is unreliable, an SNMP (Simple Network Manage Protocol) client must implement its own strategy for timeout and retransmission. Servers do not apply a single fixed timeout for all entries, but allow the authority for an entry to configure its timeout. Whenever an authority responds to a request, it includes a Time to Live (TTL) value in the response that specifies how long it guarantees the binding to remain. Thus, authorities can reduce network overhead by specifying long timeouts for entries that they expect to remain unchanged, while improving correctness by specifying short timeouts for entries that they expect to change frequently. Mobile ambients can model communication protocols. Timeout and retransmission in TCP/IP provide a good motivation to add timers to ambients.

In this paper we associate timers not only to the ambients, but also to the capabilities and communication channels. The resulting formalism is called timed mobile ambients, and represent a conservative extension of the ambient calculus. Timed mobile ambients use also types inspired from [5].

The structure of the paper is as follows. Section 2 presents the mobile ambients. Section 3 gives the description of the calculus we introduce. First provide a static semantic given by a typing system, and an operational semantics of the new calculus given by a reduction relation. The passage of time is given by a (discrete) time-stepping function. We prove that structural congruence and passage of time do not interfere with the typing system. A subject reduction results assures that once well-typed, an ambient remains well-typed. We provide some bisimulation relations with regard to the passage of time. In Section 4 a timed extension of the cab protocol illustrates how the new formalism is working. Conclusion and references end the paper.

2 Mobile Ambients

In this section we provide a short description of mobile ambients; more information can be found in [4]. The following table describes the syntax of mobile ambients.
Table 1: Syntax of MA

<table>
<thead>
<tr>
<th>n, m, p</th>
<th>names</th>
<th>P, Q ::=</th>
<th>processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y, z</td>
<td>variables</td>
<td>0</td>
<td>inactivity</td>
</tr>
<tr>
<td>M ::= capabilities</td>
<td>M.P</td>
<td>capability action</td>
<td></td>
</tr>
<tr>
<td>in n</td>
<td>can enter n</td>
<td>n[P]</td>
<td>ambient</td>
</tr>
<tr>
<td>out n</td>
<td>can exit n</td>
<td>P</td>
<td>composition</td>
</tr>
<tr>
<td>open n</td>
<td>can open n</td>
<td>(\nu n) P</td>
<td>restriction</td>
</tr>
<tr>
<td>!\langle m \rangle . P</td>
<td>output action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?(x). P</td>
<td>input action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*P</td>
<td>replication</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The process 0 is an inactive process (it does nothing). The processes M.P are called actions, and the processes n[P] are called ambients. P | Q is a parallel composition of processes P and Q, and (\nu n)P creates a new unique name n within the scope of P. An output action !\langle m \rangle . P releases a name m into the surrounding ambient, and then behaves as P. An input action ?(x). P captures a capability from the surrounding ambient, and binds it to a variable x within the scope of P. *P denotes the unbounded replication of the process P, producing as many parallel replicas of P as needed.

The semantics of the ambient calculus is given by two relations: structural congruence relation and reduction relation. The structural congruence P \equiv Q relates different syntactic representation of the same process; it is used to define the reduction relation. The reduction relation P \rightarrow Q describes the processes evolution. We write \rightarrow^* for the reflexive and transitive closure of \rightarrow.

The structural congruence is defined as the least relation over processes satisfying the axioms from the table below: commuting the positions of parallel components, stretching the scope of a restriction, renaming of bounded names and unfolding recursion.

Table 2: Structural congruence

| P \equiv Q | P \equiv Q | (P | Q) \equiv R \equiv P | (Q | R) |
|----------|----------|------------------|------------------|
| P \equiv Q, Q \equiv R implies P \equiv R | *P \equiv P | *P |
| P \equiv Q implies (\nu m) P \equiv (\nu m) Q | (\nu m)(\nu m)P \equiv (\nu m)(\nu m)P if n \neq m |
| P \equiv Q implies P | R \equiv Q | R | (\nu m)(P | Q) \equiv P | (\nu m)Q if n \notin fn(P) |
| P \equiv Q implies *P \equiv *Q | (\nu m)[mP] \equiv m[(\nu m)P] if n \neq m |
| P \equiv Q implies n[P] \equiv n[Q] | P | 0 \equiv P |
| P \equiv Q implies M.P \equiv M.Q | (\nu m)0 \equiv 0 |
| P \equiv Q implies ?(x). P \equiv ?(x). Q | *0 \equiv 0 |

The reduction relation is defined as the least relation over processes satisfying the following set of axioms:
Table 3: Reduction rules

| (In)   | n[in m. P | Q] | m[R] → m[n[P | Q] | R] |
| (Out)  | m[n.out m. P | Q] | R] → n[P | Q] | m[R] |
| (Open) | open n. P | n[Q] | P → P | Q |
| (Com)  | ![m]. P | ?(x). P' → P | P'{m/x} |
| (Res)  | P → Q implies (νn) P → (νn) Q |
| (Amb)  | P → Q implies n[P] → n[Q] |
| (Par)  | P → Q implies P | R → Q | R |
| (Struct) | P' ≡ P, P → Q, Q ≡ Q' |
|        | P' → Q' |

The first four rules are the one-step reductions for **in**, **out**, **open** and **communication**. The next three rules propagate reductions across scopes, ambient nesting and parallel composition. The final rule allows the use of structural congruence during reduction.

3 Mobile ambients with time constraints

In timed Mobile Ambients (tMA) the communication channels, capabilities and ambients are used as temporal resources; if nothing happens in a predefined interval of time, the waiting process goes to another state. The timer $\Delta t$ of each temporal resource makes the resource available only for a determined period of time $t$. We add timers to ambients, capabilities, and both input and output channels.

When we consider a timer $\Delta t$ for an ambient $n$, and this is denoted by $n^{\Delta t}[P]$, it acts as $n[P]$ while $t > 0$. Since the timer $\Delta t$ can expire ($t = 0$) we use a pair $(n^{\Delta t}[P], Q)$, where $Q$ is a safety process. If nothing happens in $t$ units of time, the ambient $n$ is dissolved, the process $P$ running inside the ambient is reduced to $0$, and the process $Q$ is executed. If $Q = 0$ we can simply write $n^{\Delta t}[P]$ instead of $(n^{\Delta t}[P], Q)$. If we want to simulate the behaviour of an ambient in untimed mobile ambients, then we use $\infty$ instead of $\Delta t$.

Similarly, the syntax for **Input** and **Output** communication, as well as for consumption of capabilities, uses a pair of processes. The process $\text{open}^{\Delta t} n.(P, Q)$ evolves to $P$ whenever, in the period of time $\Delta t$, the process has a sibling ambient $n$; otherwise evolves to $Q$. The process $!(m)^{\Delta t}.(P, Q)$ evolves to $P$ whenever, in the period of time $\Delta t$, the process has a sibling process which is willing to capture the name $m$; otherwise evolves to $Q$.

Within an ambient, multiple processes can freely execute input and output actions. Since the messages are undirected, it is easily possible for a process to utter a message that is not appropriate for some receiver. To be sure that a message reaches the appropriate receiver, which is denoted with $?_{\text{Amb}}[(\Gamma)]^{\Delta t}.(P, Q)$, we introduce a notion of type expressed by $\text{Amb}[\Gamma]$. If a correct message is received into a period of time $\Delta t$, then the process evolves to $P$; otherwise evolves to $Q$. $\text{Amb}[\Gamma]$ is also used in a restriction $(\nu n : \text{Amb}[\Gamma]) P$, which means that $n$ of type $\text{Amb}[\Gamma]$ is new in $P$. The set of types is introduces into Table 5, being inspired from [5].

The syntax of the timed Mobile Ambients is defined in the following table.
Mobile ambients with time constraints

Table 4: Syntax of tMA

<table>
<thead>
<tr>
<th>n, m, p</th>
<th>names</th>
<th>P, Q ::=</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y, z</td>
<td>variables</td>
<td>(vn : Amb[Γ])P</td>
</tr>
<tr>
<td>M ::=</td>
<td>capabilities</td>
<td></td>
</tr>
<tr>
<td>in n</td>
<td>can enter n</td>
<td>P ∣ Q</td>
</tr>
<tr>
<td>out n</td>
<td>can exit n</td>
<td>(nΔt[P], Q)</td>
</tr>
<tr>
<td>open n</td>
<td>can open n</td>
<td>MΔt.(P, Q)</td>
</tr>
<tr>
<td>!⟨m⟩Δt.(P, Q)</td>
<td>capability action</td>
<td></td>
</tr>
<tr>
<td>?(x : Amb[Γ])Δt.(P, Q)</td>
<td>input action</td>
<td></td>
</tr>
<tr>
<td>*P</td>
<td>replication</td>
<td></td>
</tr>
</tbody>
</table>

In the process ?(x : Amb[Γ])Δt.(P, Q), the variable x is considered bounded only in P and we should provide its type Amb[Γ]. We also need to worry about the consumption of a capability open which might open an ambient and unleash its set of types inside the surrounding ambient.

Table 5: Types

Set of types:

<table>
<thead>
<tr>
<th>Γ ::=</th>
<th>∅</th>
<th>Amb[Γ]</th>
<th>Cap[Γ]</th>
<th>Γ ∩ Γ’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amb[Γ]</td>
<td>ambient name containing a set of types Γ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cap[Γ]</td>
<td>capability that can handle a set of types Γ</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

∅ denotes an empty set of types. The intuitive behaviour of the subtyping relation < is the inverse of the inclusion of sets (Γ <: Γ’ for types means Γ ⊇ Γ’ for sets and Γ ∩ T for types means Γ ∪ T for sets).

3.1 Semantics

The main feature of tMA is given by time. The passage of time is described by a time-stepping function φΔ defined over the set P of processes. The possible actions are performed at every tick of a universal clock. Active channels are those which could be involved in a communication, and active capabilities are those which can be consumed. φΔ affects the capabilities which are not consumed, the active channels which do not communicate at the tick of the universal clock (the channels involved in communication and the consumed capabilities disappear together with their timers), and the ambients. If a channel, capability or ambient has the timer equal to ∞ we use the equality ∞ − 1 = ∞ when applying the function φΔ. We define φΔ to modify a process accordingly with the passage of time.

DEFINITION 3.1 (Time-stepping function) We define φΔ : P → P, by:
Mobile ambients with time constraints

interacting parts can be brought together. This relation provides a way of rearranging expressions so that reduction relation syntactic representations of the same process; it is used to define the reduction relation. The congruence relation and reduction relation. The definition of stands for !⟨m⟩ or ?⟨x : Amb[Γ]⟩.

The communication operation does not consume time, so we do not include it in the definition of φΔ function. To see how this function is used see the reduction table (Table 7).

The semantics of the timed Mobile Ambients is given by two relations: structural congruence relation and reduction relation. The structural congruence P ≡ Q relates different syntactic representations of the same process; it is used to define the reduction relation. The reduction relation P → Q describes the processes evolution.

Processes are grouped into equivalence classes by the following equivalence relation, ≡, called structural congruence. This relation provides a way of rearranging expressions so that interacting parts can be brought together.

<table>
<thead>
<tr>
<th>Table 6: Structural congruence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S-Ref) P ≡ P</td>
</tr>
<tr>
<td>(S-Sym) P ≡ Q implies Q ≡ P</td>
</tr>
<tr>
<td>(S-Trans) P ≡ R, R ≡ Q implies P ≡ Q</td>
</tr>
<tr>
<td>(S-Res) P ≡ Q implies (νn : Amb[Γ])P ≡ (νn : Amb[Γ])Q</td>
</tr>
<tr>
<td>(S-Par) P ≡ Q implies P</td>
</tr>
<tr>
<td>(S-Repl) P ≡ Q implies *P ≡ *Q</td>
</tr>
<tr>
<td>(S-Amb) P ≡ Q implies (n^Δt[P], R) ≡ (n^Δt[Q], R)</td>
</tr>
<tr>
<td>(S-Cap) P ≡ Q implies M^Δt.(P, R) ≡ M^Δt.(Q, R)</td>
</tr>
<tr>
<td>(S-Input) P ≡ Q implies ?⟨x : Amb[Γ]⟩^Δt.(P, R) ≡ ?⟨x : Amb[Γ]⟩^Δt.(Q, R)</td>
</tr>
<tr>
<td>(S-Par Com) P</td>
</tr>
<tr>
<td>(S-Par Assoc) (P</td>
</tr>
<tr>
<td>(S-Repl Par) *P ≡ P</td>
</tr>
<tr>
<td>(S-Res Res) (νn : Amb[Γ])(νn : Amb[Γ′])P ≡ (vm : Amb[Γ′])(vm : Amb[Γ])P if n ≠ m</td>
</tr>
<tr>
<td>(S-Res Par) (vm : Amb[Γ])(P</td>
</tr>
<tr>
<td>(S-Res Amb) (vm : Amb[Γ])(m^Δt[P], Q) ≡ (m^Δt[(vm : Amb[Γ])P], Q) if n ≠ m</td>
</tr>
<tr>
<td>(S-Zero Par) P</td>
</tr>
<tr>
<td>(S-Zero Res) (vm : Amb[Γ])0 ≡ 0</td>
</tr>
<tr>
<td>(S-Zero Repl) *0 ≡ 0</td>
</tr>
</tbody>
</table>
For the processes \( M^{\Delta t}(P, Q) \) and \( \diamondsuit^{\Delta t}(P, Q) \) the timers of \( P \) activates only after the consumption of \( M^{\Delta} \) or \( \diamondsuit^{\Delta} \). To preserve the timers of \( P \) we introduce a function which prevents the application of the time-stepping function for \( P \).

**DEFINITION 3.2 (Time-preserving function)** We define \( \psi : P \rightarrow P \), by:

\[
\phi_{\Delta}(\psi(P)) = P
\]

We denote by \( \not\rightarrow \) the fact that rules (In), (Out), (Open) and (Com) cannot be applied. The behavior of processes is given by the following reduction rules:

<table>
<thead>
<tr>
<th>Table 7: Reduction rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R-Idle) ( P \not\rightarrow ) ( P \rightarrow \phi_{\Delta}(P) )</td>
</tr>
<tr>
<td>(R-In) ( (n^{\Delta t}[in^{\Delta t}m.(P, P')</td>
</tr>
<tr>
<td>(R-Out) ( (m^{\Delta t}[out^{\Delta t}m.(P, P')</td>
</tr>
<tr>
<td>(R-Open) ( m : Amb[\Gamma] )</td>
</tr>
<tr>
<td>(R-Com) ( m : Amb[\Gamma] )</td>
</tr>
<tr>
<td>(R-Res) ( (\nu n : Amb[\Gamma]) P \rightarrow (\nu n : Amb[\Gamma]) Q )</td>
</tr>
<tr>
<td>(R-Amb) ( (n^{\Delta t}[P], R) \rightarrow \phi_{\Delta}((n^{\Delta t}[\psi(Q)], R)) )</td>
</tr>
<tr>
<td>(R-Par) ( P \rightarrow Q )</td>
</tr>
<tr>
<td>(R-Struct) ( P' \equiv P, P \rightarrow Q, Q \equiv Q' )</td>
</tr>
</tbody>
</table>

In the rules (R-In), (R-Out), (R-Open) and (R-Amb) the time-stepping function is applied because the time is passing whether a reduction is done or not in the system. In (R-In), (R-Out), (R-Open) the timers of \( P \) are preserved by using the function \( \psi \) because the process is activated after the consumption of a capability. In (R-Amb) the timers of \( Q \) are preserved because \( Q \) has already participated in a reduction. The function \( \phi_{\Delta} \) decreases the timers, and for the expired timers the function discards the channels, capabilities and ambients. In rule (R-Par) a process \( R \) reduces to \( \phi_{\Delta}(R) \) because \( R \) has no internal reduction but the time is passing.
3.2 Well-typed relation and results

*Well-typedness* of processes is defined by a set of rules. The typing rules of Table 8 express the conditions which must be satisfied for each syntactic construction of a process in order to be well-typed. These rules describe the behaviour of a process with respect to its types, providing the static semantics of \( tMA \).

<table>
<thead>
<tr>
<th>Table 8: Typing rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T-Null) ( \Gamma \vdash 0 )</td>
</tr>
<tr>
<td>(T-Write) ( \Gamma \cap \text{Amb}[\Gamma'] \vdash P ), ( \Gamma \vdash Q )</td>
</tr>
<tr>
<td>(T-Read) ( \Gamma \cap \text{Amb}[\Gamma'] \vdash ?(x : \text{Amb}[\Gamma'])? ( (P, Q) )</td>
</tr>
<tr>
<td>(T-Repl) ( \Gamma \vdash *P )</td>
</tr>
<tr>
<td>(T-New) ( \Gamma \vdash (\nu n : \text{Amb}[\Gamma'])P )</td>
</tr>
<tr>
<td>(T-Par) ( \Gamma \vdash P ), ( \Gamma \vdash Q )</td>
</tr>
</tbody>
</table>

**Lemma 3.1** If \( \Gamma \vdash (\nu n : \text{Amb}[\Gamma'])P \) and \( n \notin fn(P) \) then \( \Gamma \vdash P \).

**Lemma 3.2** If \( \Gamma \cap \text{Amb}[\Gamma'] \vdash P \), \( x, m : \text{Amb}[\Gamma'] \), \( x \in bn(P) \) and \( m \notin fn(P) \) then \( \Gamma \cap \text{Amb}[\Gamma'] \vdash P \{m/x\} \).

We write \( \Gamma \vdash P \) and say that *process P is well-typed with respect to the set of types \( \Gamma \).* In order to say that \( !\langle n \rangle^{\Delta t} (P, Q) \) is well-typed, with respect to the set of types \( \Gamma \), the following statements should hold: (i) \( n : \text{Amb}[\Gamma'] \), which means that \( n \) is an ambient name which contains the set of types \( \Gamma' \); (ii) \( \Gamma \ll : \text{Amb}[\Gamma'] \), which means that \( \Gamma \) contains the type \( \text{Amb}[\Gamma'] \); (iii) \( \Gamma \vdash P \); \( \Gamma \vdash Q \), which means that \( P \) and \( Q \) are well-typed with respect to the set of types \( \Gamma \).

If an ambient \( n \) contains a set of types \( \Gamma \), then the capability *open n* may unleash these set of types into the surrounding ambient. We say that *open* \( n : \text{Cap}[\Gamma] \), meaning that it can unleash the set of types \( \Gamma \) when used. The capability types \( \text{Cap}[\Gamma] \) do not keep track of any information concerning *in* and *out* capabilities; only the effect of *open* is tracked.

The following proposition states that the application of the time-stepping function \( \phi_\Delta \) to a process \( P \) will not change its property of being well-typed.

**Proposition 3.3 (Time passage)** If \( \Gamma \vdash P \) then \( \Gamma \vdash \phi_\Delta(P) \).

**Proof:** In this stage of the calculus the time passage affects only the timers by application of the \( \phi_\Delta \) function. The proof will proceed by induction on the composition of \( P \) in the same style used by Milner in [16]. We must take into account all the cases which enter in the definition of \( \phi_\Delta \).

*Case inferred from \( P = (\nu n : \text{Amb}[\Gamma'])R \).* From the statement of the proposition we have \( \Gamma \vdash (\nu n : \text{Amb}[\Gamma'])R \). This must have been derived from (T-New) with \( \Gamma \cap \text{Amb}[\Gamma'] \vdash R \). By
induction we get $\Gamma \cap \text{Amb}[\Gamma'] \vdash \phi_\Delta(R)$, and applying the rule (T-New) we get the expected result $\Gamma \vdash (\nu n : \text{Amb}[\Gamma'])(\nu n : \text{Amb}[\Gamma'])P \equiv Q$ which is the same as $\Gamma \vdash \phi_\Delta(P)$.

**Case** inferred from $P = M_\Delta(R, Q), t > 1$. The syntax is a general notation to capture all the capabilities because their behaviour is the same in this context. As a consequence, depending on what kind of capability we have, one of the rules (T-In,Out) or (T-Open) is applied and the expected result $\Gamma \vdash M_\Delta(t)(R, Q)$ is obtained which is the same as $\Gamma \vdash \phi_\Delta(P)$.

**Case** inferred from $P = M_\Delta(R, Q), t \leq 1$. Is the one as the same before.

**Case** inferred from $P = \sigma_\Delta(R, Q), t > 1$. From the statement of the proposition we have $\Gamma \vdash (n_\Delta[R], Q)$, by applying the function $\phi_\Delta$ we obtain the process $(n_\Delta(t\times[R], Q)$ which is well-typed over any set of types, in particular $\Gamma \vdash (n_\Delta(t\times[R], Q)$ which is the same as $\Gamma \vdash \phi_\Delta(P)$.

**Case** inferred from $P = (n_\Delta[R], Q), t \leq 1$. Is the one as the same before.

**Case** inferred from $P = *R$ or $P = 0$ is obvious.

The following proposition states that if a process $P$ is well-typed then all the processes from its equivalence class are well-typed.

**PROPOSITION 3.4 (Subject Congruence)** If $P \equiv Q$ then $\Gamma \vdash P \iff \Gamma \vdash Q$.

**Proof:** We proceed by structural induction.

(S-Refl), (S-Sym) Trivial.

(S-Trans) We have $P \equiv R, Q \equiv S$ for some $R$. By induction if $P \equiv R$ then $\Gamma \vdash P \iff \Gamma \vdash R$ and similarly if $R \equiv Q$ then $\Gamma \vdash R \iff \Gamma \vdash Q$ from where we obtain that if $P \equiv Q$ then $\Gamma \vdash P \iff \Gamma \vdash Q$.

(S-Res) We have $P = (\nu n : \text{Amb}[\Gamma'])(P')$ and $Q = (\nu n : \text{Amb}[\Gamma'])(Q')$ with $P' \equiv Q'$. Assume $\Gamma \vdash P$. This must have been derived from (T-New) with $\Gamma \cap \text{Amb}[\Gamma'] \vdash P'$. By induction we obtain $\Gamma \cap \text{Amb}[\Gamma'] \vdash Q'$ and applying (T-New) we obtain that $\Gamma \vdash Q$. Similarly the other implication.

(S-Par) We have $P = P' \parallel R$ and $Q = Q' \parallel R$ with $P' \equiv Q'$. Assume $\Gamma \vdash P$. This must have been derived from (T-Par) with $\Gamma \vdash P'$ and $\Gamma \vdash R$. By induction we obtain that $\Gamma \vdash Q'$ and applying (T-Par) we obtain that $\Gamma \vdash Q$. Similarly the other implication.

(S-Repl) We have $P = *P'$ and $Q = *Q'$ with $P' \equiv Q'$. Assume $\Gamma \vdash P$. This must have been derived from (T-Repl) with $\Gamma \vdash P'$. By induction we have that $\Gamma \vdash Q'$ and applying (T-Repl) we obtain that $\Gamma \vdash Q$. Similarly the other implication.
(S-Amb) We have that $P = (n^\Delta[P'], R)$ and $Q = (n^\Delta[Q'], R)$ with $P' \equiv Q'$. Assume $\Gamma \vdash P$. This must have been derived from (T-Amb) with $\Gamma \vdash R$. We have that $n^\Delta[P']$ and $n^\Delta[Q']$ are well-typed under all sets of types. By applying the rule (T-Amb) we obtain that $\Gamma \vdash Q$. Similarly the other implication.

(S-Cap) We have that $P = M^\Delta.(P', R)$ and $Q = M^\Delta.(Q', R)$ with $P' \equiv Q'$. Assume $\Gamma \vdash P$. We have two subcases from which this must have been derived: (T-In,Out) and (T-Open). We study only one case the other being treated similarly. For (T-In,Out) we have derived from (T-Repl) with $(S-Zero Repl)$ implication. For (T-Par) we have derived from (T-Par) with $(S-Zero Par)$.

We have that $\Gamma \vdash Q$. Similarly the other implication.

(S-Res Amb) We have that $P = (m^\Delta[n^\Delta](P'), P')$ and $Q = (m^\Delta[n^\Delta](P'), P')$ with $n \neq m$. Assume $\Gamma \vdash P$. This must have been derived from (T-New) and (T-Amb) with $\Gamma \sqcap Amb[\Gamma'] \vdash P''$. Because $n$ does not affect the process $P''$ by applying the rule lemma 3.1 we have that $\Gamma \vdash P''$. By applying (T-Amb) we obtain that $\Gamma \vdash Q$. Similarly the other implication.

(S-Zero Repl) We have that $P = *0$ and $Q = 0$. Assume $\Gamma \vdash P$. This must have been derived from (T-Repl) with $\Gamma \vdash 0$ which is exactly $\Gamma \vdash Q$. Similarly the other implication.
The following proposition states that if a process $P$ is well-typed then the process obtain after applying a reduction rule is well-typed.

**PROPOSITION 3.5 (Subject Reduction)** If $P \rightarrow Q$ then $\Gamma \vdash P$ iff $\Gamma \vdash Q$.

**Proof:** We proceed by induction on the derivation of $P \rightarrow Q$.

**(R-Idle)** This case was studied in proposition 3.3.

**(R-In), (R-Out), (R-Open)** Assume $\Gamma \vdash P$. In any of the three rules we have that $P$ and $Q$ are an ambient or a parallel composition of ambients, which means that are well typed under any set of types. In particular we have that $\Gamma \vdash Q$. Similarly the other implication.

**(R-Com)** We have that $P = \lambda (m) (P, Q) | ?(x : Amb[\Gamma])^\Delta$. $(P', Q')$ and $Q = P | P' \{m/x\}$. Assume $\Gamma \vdash P$. This must have been derived from (T-Par) with $\Gamma \vdash \lambda (m) (P, Q)$ and $\Gamma \vdash (? (x) \Delta (P', Q'))$ and by applying the rules (T-Write) and (T-Read) we obtain that $\Gamma \vdash P$, $\Gamma \vdash P'$ and $\Gamma <: Amb[\Gamma]$. By applying the lemma 3.2 and the rule (T-Par) we obtain that $\Gamma \vdash P | P' \{m/x\}$. Similarly the other implication.

**(R-Res)** We have that $P = \nu n : Amb[\Gamma'] P'$ and $Q = \nu n : Amb[\Gamma'] Q'$ with $P' \rightarrow Q'$. Assume $\Gamma \vdash P$. This must have been derived from (T-New) with $\Gamma \cap Amb[\Gamma'] \vdash P'$ and proceeding by induction we have that $\Gamma \cap Amb[\Gamma'] \vdash Q'$. Applying the rule (T-New) we obtain that $\Gamma \vdash (\nu n : Amb[\Gamma']) Q'$ which is similarly to $\Gamma \vdash Q$. Similarly the other implication.

**(R-Amb)** We have that $P = \phi \Delta (\nu n : Amb[\Gamma'] P')$ and $Q = \phi \Delta (\nu n : Amb[\Gamma'] Q')$ with $P' \rightarrow Q'$. Assume $\Gamma \vdash P$. This must have been derived from (T-Amb) with $\Gamma \vdash P'$, $\Gamma \vdash \phi \Delta$, and proceeding by induction we have that $\Gamma' \vdash Q'$. Applying the rule (T-Amb) we obtain that $\Gamma \vdash Q$. Similarly the other implication.

**(R-Par)** We have that $P = P' \mid R$ and $Q = Q' \mid \phi \Delta (R)$ with $P' \rightarrow Q'$. Assume $\Gamma \vdash P$. This must have been derived from (T-Par) with $\Gamma \vdash P'$, $\Gamma \vdash R$, where $\Gamma$ and proceeding by induction we have that $\Gamma \vdash Q'$. Applying the rule (T-Idle) we have that $\Gamma \vdash \phi \Delta (R)$ and the rule (T-Par) we have that $\Gamma \vdash Q$. Similarly the other implication.

**(R-Struct)** We have that $P = P'$, $Q \equiv Q'$ with $P' \rightarrow Q'$. Assume $\Gamma \vdash P$. By applying proposition 3.4 and proceeding by induction we have that $\Gamma \vdash Q$. Similarly the other implication.

We describe here the error system of $tMA$, denoting by $\xrightarrow{err}$ the generation of an error. A runtime error is possible only when a process tries to do something against the types accumulated in its type environment. Note that if a process wants to communicate a value of a type which is not in the type system of the upper ambient, it can still be well-typed if the alternative process $Q$ is well-typed; $Q$ is called the safety process. When a type is not in the set of types of the process, the safety process is chosen by $\phi \Delta$ function.

We denote by $n \not\in Amb[\Gamma]$ the fact that ambient $n$ does not have the type $Amb[\Gamma]$. 
Table 9: Runtime errors

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E-Com)</td>
<td>( n :: \text{Amb}[\Gamma] )</td>
</tr>
<tr>
<td></td>
<td>( !\langle n \rangle^{\Delta t}.(P,Q) ) ( ?(x : \text{Amb}[\Gamma])^{\Delta t}.(P',Q') ) ( \xrightarrow{\text{err}} ) (E-Amb) ( P \xrightarrow{\text{err}} ) ( n^{\Delta t}[P,Q] ) ( \xrightarrow{\text{err}} )</td>
</tr>
<tr>
<td>(E-Open)</td>
<td>( n :: \text{Amb}[\Gamma] ) or ( m :: \text{Amb}[\Gamma] ) and ( n :: \text{Amb}[\Gamma] ) and ( \Gamma' \not\vdash \Gamma ) ( (m^{\Delta t'[\text{open}^{\Delta t}n].(P,P')</td>
</tr>
</tbody>
</table>

The following proposition states that if a process is well typed, then it does not generate errors.

**PROPOSITION 3.6** If \( \Gamma \vdash P \) then \( P \xrightarrow{\text{err}} \).

**Proof:** The proof considers the opposite of the fact that if \( P \) gives rise to a runtime error \( \langle P \xrightarrow{\text{err}} \rangle \), then \( P \) cannot be well-typed under any set of types \( \Gamma \) (\( \Gamma \not\vdash P \), for all \( \Gamma \)). We use induction on the structure of \( P \) and consider a proof cases for each rule in Table 9.

- **(E-Com)** We consider that there exist a set of types \( \Gamma \) such that \( \Gamma \vdash R \), where \( R = !\langle n \rangle^{\Delta t}.(P,Q) \) \( ?(x : \text{Amb}[\Gamma'])^{\Delta t}.(P',Q') \). This must have been derived from (T-Par) with \( \Gamma \vdash !\langle n \rangle^{\Delta t}.(P,Q) \) and \( \Gamma \vdash ?(x : \text{Amb}[\Gamma'])^{\Delta t}.(P',Q') \). Applying (T-Write), (T-Read) we have that \( \Gamma \vdash _{\text{Amb}}[\Gamma'] \) and \( n : \text{Amb}[\Gamma'] \), which is in contradiction with the hypothesis of the rule (E-Com), and so we have that \( R \xrightarrow{\text{err}} \).

- **(E-Open)** We consider that there exist a set of types \( \Gamma \) such that \( \Gamma \vdash R \), where \( R = (m^{\Delta t'[\text{open}^{\Delta t}n].(P,P') | (n^{\Delta t''[Q,S'']},S') \xrightarrow{\text{err}} \) (E-New) \( P \xrightarrow{\text{err}} \) (E-Par) \( P | Q \xrightarrow{\text{err}} \) (E-Str) \( P \equiv Q \) \( Q \xrightarrow{\text{err}} \) |

So in consequence \( (n^{\Delta t}[P,Q] \) cannot be well-typed under any set of rules, and so we have that \( (n^{\Delta t}[P,Q] \) \( \not\vdash \) (S-Repl Par).

We have the same idea as at (E-Amb).

### 3.3 Properties related to \( \phi_\Delta \)

We denote by \( P \xrightarrow{\phi_\Delta} Q \) the fact that process \( P \) evolves to process \( Q \) after applying the rule (R-Idle) for \( t \geq 0 \) times, and with \( t\phi_\Delta(R) \) the fact that function \( \phi_\Delta \) is applied \( t \) times to \( R \). We denote by \( \equiv' \) the relation which respects all the rules of Table 6 except the rule (S-Repl Par).

We claim that the passage of time cannot cause a nondeterministic behaviour.
PROPOSITION 3.7 If \( P \equiv' Q \), \( P \overset{\phi}{\rightarrow} P' \) and \( Q \overset{\phi}{\rightarrow} Q' \) then \( P' \equiv' Q' \).

**Proof:** We proceed by structural induction.

(S-Refl), (S-Sym) Trivial.

(S-Trans) We have \( P \equiv R \), \( R \equiv Q \) for some \( R \). By induction if \( P \equiv R \), \( P \overset{\phi}{\rightarrow} P' \) and \( R \overset{\phi}{\rightarrow} R' \) then \( P' \equiv R' \) and similarly if \( R \equiv Q \), \( R \overset{\phi}{\rightarrow} R' \) and \( Q \overset{\phi}{\rightarrow} Q' \) then \( R' \equiv' Q' \), from where we obtain that if \( P' \equiv' Q' \).

(S-Res) We have \( P = (\nu n : \text{Amb}[\Gamma']) \) \( P' \) and \( Q = (\nu n : \text{Amb}[\Gamma']) \) \( Q' \) with \( P \equiv Q \). By induction we have that if \( P' \equiv Q' \), \( P' \overset{\phi}{\rightarrow} P'' \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) then \( P'' \equiv Q'' \). By applying (R-Res) to both \( P' \overset{\phi}{\rightarrow} P'' \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) we obtain that \( P \overset{\phi}{\rightarrow} (\nu n : \text{Amb}[\Gamma']) \) \( P'' \) and \( Q \overset{\phi}{\rightarrow} (\nu n : \text{Amb}[\Gamma']) \) \( Q'' \). By applying (S-Res) to \( P'' \equiv Q'' \) we obtain that \( (\nu n : \text{Amb}[\Gamma']) \) \( P'' \equiv' (\nu n : \text{Amb}[\Gamma']) \) \( Q'' \).

(S-Par) We have \( P = P' \) \( R \) and \( Q = Q' \) \( R \) with \( P \equiv Q \). By induction we have that if \( P' \equiv Q' \), \( P' \overset{\phi}{\rightarrow} P'' \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) then \( P'' \equiv Q'' \). By applying (R-Par) to both \( P' \overset{\phi}{\rightarrow} P'' \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) we obtain that \( P \overset{\phi}{\rightarrow} P'' \) \( t_\Delta(R) \) and \( Q \overset{\phi}{\rightarrow} Q'' \) \( t_\Delta(R) \). By applying (S-Par) to \( P'' \equiv Q'' \) we obtain that \( P'' \) \( t_\Delta(R) \) \( Q'' \) \( t_\Delta(R) \).

(S-Repl) We have \( P = *P' \) and \( Q = *Q' \) with \( P \equiv Q \). The time-stepping function does not affect the replication so the processes \( P \) and \( Q \) remain the same and is obvious that \( P \equiv Q \).

(S-Amb) We have that \( P = (n\Delta[t][P'], R) \) and \( Q = (n\Delta[t][Q'], R) \) with \( P \equiv Q \). If \( t' > t \), by induction we have that if \( P' \equiv Q' \), \( P' \overset{\phi}{\rightarrow} P'' \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) then \( P'' \equiv Q'' \). By applying (R-Amb) to both \( P' \overset{\phi}{\rightarrow} P'' \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) we obtain that \( P \overset{\phi}{\rightarrow} (\nu n : (n\Delta[t][t_\psi(P'])) \) \( R \) \( R \) \( (n\Delta[t][t_\psi(Q'])) \) \( R \) \( R \). By applying (S-Amb) to \( P'' \equiv Q'' \) we obtain that \( \phi_\Delta((\nu n : (n\Delta[t][t_\psi(P'])) \) \( R \) \( R \) \( (n\Delta[t][t_\psi(Q'])) \) \( R \) \( R \)). If \( t' \leq t \) both processes evolve to \( R \) after \( t \) units of time and the demonstration is obvious.

(S-Cap) We have that \( P = M\Delta.(P', R) \) and \( Q = M\Delta.(Q', R) \) with \( P \equiv Q \), \( P' \equiv Q' \). If \( t' > t \) applying (S-Cap) to \( P' \) and \( Q' \) with \( M\Delta(t'-t) \) we obtain \( M\Delta.(P', R) \equiv M\Delta(t'-t).Q', R \). If \( t' \leq t \), by induction we have that if \( P' \equiv Q' \), \( P' \overset{\phi}{\rightarrow} P'' \) \( M\Delta(t'-t) \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) then \( P'' \equiv Q'' \).

(S-Input) Similarly to (S-Cap).

(S-Par Com) We have that \( P = P' \) \( P'' \) and \( Q = P'' \) \( P' \) with \( P \equiv Q \). We have that \( P \overset{\phi}{\rightarrow} t_\Delta(P') \) \( t_\Delta(P'') \) and \( Q \overset{\phi}{\rightarrow} t_\Delta(P') \) \( t_\Delta(P'') \) \( t_\Delta(P'). \) By applying (S-Par Com) we obtain that \( t_\Delta(P') \) \( t_\Delta(P'') \) \( t_\Delta(P'). \)

(S-Par Assoc) Similarly to (S-Par Com).

(S-Res Res) We have that \( P = (\nu n : \text{Amb}[\Gamma']) \) \( (\nu m : \text{Amb}[\Gamma']) \) \( R \) and \( Q = (\nu m : \text{Amb}[\Gamma']) \) \( (\nu n : \text{Amb}[\Gamma']) \) \( R \) with \( P \equiv Q \). By induction we have that if \( P' \equiv Q' \), \( P' \overset{\phi}{\rightarrow} P'' \) and \( Q' \overset{\phi}{\rightarrow} Q'' \) then \( P'' \equiv Q'' \). By applying (R-Res) twice to both \( P' \overset{\phi}{\rightarrow} P' \) and \( Q' \overset{\phi}{\rightarrow} Q' \) we obtain that \( P \overset{\phi}{\rightarrow} (\nu m : \text{Amb}[\Gamma']) \) \( P'' \) and \( Q \overset{\phi}{\rightarrow} (\nu m : \text{Amb}[\Gamma']) \) \( Q'' \).


By applying \((S-\text{Res})\) twice and \((T-\text{Par})\) once to \(P'' \equiv' Q''\) we obtain that \((\nu n : \text{Amb}[\Gamma'])(\nu m : \text{Amb}[\Gamma''])P'' \equiv' (\nu m : \text{Amb}[\Gamma''])Q''\).

\((S-\text{Res Par})\) We have that \(P = (\nu n : \text{Amb}[\Gamma'])(P' | P'')\) and \(Q = P' | (\nu n : \text{Amb}[\Gamma''])P''\) with \(n \notin fn(P')\) and \(P \equiv Q\). We have that \(P \stackrel{\phi_{\Delta}t}{\rightarrow} (\nu n : \text{Amb}[\Gamma'])\) \((t\phi_{\Delta}(P') | t\phi_{\Delta}(P''))\) and \(Q \stackrel{\phi_{\Delta}t}{\rightarrow} (\nu n : \text{Amb}[\Gamma'])t\phi_{\Delta}(P'').\) By applying \((S-\text{Res Par})\) we obtain that \((\nu n : \text{Amb}[\Gamma'])t\phi_{\Delta}(P'') \equiv t\phi_{\Delta}(P'')\).

\((S-\text{Res Amb})\) Same as \((S-\text{Amb})\), but applying also \((T-\text{Res})\) and \((S-\text{Res Amb})\). \((S-\text{Zero Par})\), \((S-\text{Zero Res})\), \((S-\text{Zero Repl})\) Trivial.

A process \(P\) simulates another process \(Q\) if whatever an event \(Q\) may perform, \(P\) may mimic this event and evolve into a new state which continues to be in the same simulation relation with the new state of \(Q\). Bisimilarity of two processes is defined by requiring that the simulation relation is symmetric, that is, each process can mimic any event of the other while remaining in the simulation relation with the new state of the former process. Since we have a global clock, it is possible to define a bisimulation in tMA which requires processes to match their time passages.

**DEFINITION 3.3** A binary relation \(\mathcal{R}\) over processes is a strong simulation if whenever \((P, Q) \in \mathcal{R}\), if \(P \stackrel{\phi_{\Delta}t}{\rightarrow} P'\) then there exists \(Q'\) such that \(Q \stackrel{\phi_{\Delta}t}{\rightarrow} Q'\), \(t = t'\) and \((P', Q') \in \mathcal{R}\). A binary relation \(\mathcal{R}\) is said to be a strong bisimulation if both \(\mathcal{R}\) and its converse are strong simulations. We say that \(P\) and \(Q\) are strongly bisimilar, written \(P \sim_t Q\), if there exists a strong bisimulation \(\mathcal{R}\) such that \(P \mathcal{R} Q\).

In this way we have a strong bisimulation for each \(t \geq 0\).

**PROPOSITION 3.8** If \(P \equiv' Q\) then \(P \sim_t Q\), for all \(t \geq 0\).

**DEFINITION 3.4** A process context \(\mathcal{C}\) is a process containing a hole, represented by \([],\). Process contexts are given by the following syntax:

\[
\mathcal{C} ::= [\cdot] | (\nu n : \text{Amb}[\Gamma])\mathcal{C} | P \cdot \mathcal{C} | \mathcal{C} \cdot P | (n^{\Delta t}[\cdot], Q)
\]

Let \(\mathcal{C}(P)\) be the process obtained by filling each hole in \(\mathcal{C}\) with a copy of \(P\); names free in \(P\) may become bound. The elementary contexts are \((\nu n : \text{Amb}[\Gamma])[\cdot], [\cdot] R, R [\cdot],\) and \((n^{\Delta t}[\cdot], R)\). We say that an equivalence relation is a congruence if it is preserved by all elementary contexts.

**PROPOSITION 3.9** \(\sim_t\) is an equivalence relation.

**PROPOSITION 3.10** \(\sim_t\) is a congruence.

**Proof:** We must show that if \(P \sim_t Q\) then:

1. \((\nu n : \text{Amb}[\Gamma])P \sim_t (\nu n : \text{Amb}[\Gamma])Q\)
We shall consider just 2; the others are similar. We prove that 
\[ \mathcal{R} = \{(P|R, Q|R) \mid P \sim_t Q\} \]
is a strong bisimulation. Let \( P|R \overset{\phi_\Delta}{\to} t U \). We must find \( V \) such that \( Q|R \overset{\phi_\Delta}{\to} t V \) and \((U,V) \in \mathcal{R}\). We have that \( U = t\phi_\Delta(P)|t\phi_\Delta(Q), V = t\phi_\Delta(Q)|t\phi_\Delta(R) \). From \( P \sim_t Q \) we have that \( t\phi_\Delta(P) \sim_t t\phi_\Delta(Q) \), which means that \((U,V) \in \mathcal{R}\).

**DEFINITION 3.5** A binary relation \( \mathcal{R} \) over processes is a weak simulation if whenever \((P, Q) \in \mathcal{R}\), if \( P \overset{\phi_\Delta}{\to} t P' \) then there exists \( Q' \) and \( t' \geq 0 \) such that \( Q \overset{\phi_\Delta}{\to} t Q' \) and \((P', Q') \in \mathcal{R}\). A binary relation \( \mathcal{R} \) is said to be a weak bisimulation if both \( \mathcal{R} \) and its converse are weak simulations. We say that \( P \) and \( Q \) are weakly bisimilar, written \( P \approx_t Q \), if there exists a weak bisimulation \( \mathcal{R} \) such that \( PRQ \).

**PROPOSITION 3.11** \( P \sim_t Q \) implies \( P \approx_t Q \).

**PROPOSITION 3.12** \( \approx_t \) is an equivalence relation.

### 4 Example

The cab protocol is described by using ambients in [11]. Roughly speaking is about a city with various sites, cabs and clients willing to go from one site to another. The cab protocol is presented as an example of a graphical implementation for mobile ambients at [http://www-sop.inria.fr/mimosa/ambicobjs/taxis.html](http://www-sop.inria.fr/mimosa/ambicobjs/taxis.html). The implementation is written in Java, and it presents the ambients as named and coloured circles, whose limits act as real boundaries for what is inside. A capability in \( c \) is described by an anchor which remains in the ambient \( a \), and an arrow outside which is linked to any ambient with name \( c \). When such an arrow finds an ambient \( c \), ambient \( a \) is entirely moved inside \( c \). A capability out \( c \) is described by an anchor pointing outside. A capability open \( c \) is represented as a small square trying to find an ambient with the same name. If it does, the boundaries are dissolved and the contents of that ambient are released. A snapshot is presented in the following figure:
We extend this protocol by introducing new operations which describe a recall for a taxi when a certain period of time has passed, the payment of the trip. We use timers over ambients, capabilities and communication actions.

A message emitted by a client at a site from to call a cab is described by

$$load	ext{ client} = load^\Delta t_1[out^\Delta t_2\text{ cab. in}^\Delta t_3\text{ client}]$$

$$call = call^\Delta t_7[out^\Delta t_8\text{ client. out}^\Delta t_9\text{ from. in}^\Delta t_10\text{ cab. in}^\Delta t_11\text{ from. load client}]$$

$$recall = recall^\Delta t_12[out^\Delta t_13\text{ cab. in}^\Delta t_14\text{ from. in}^\Delta t_15\text{ client}]$$

$$call\ from\ client = (call, recall)$$

In order to call a cab, the client send a call ambient. This ambient must enter a cab, where it gets opened and releases the process load client. If the timer $\Delta t_7$ expires, a process recall is released which enters the ambient client and announce that he can make another call. This process of recalling is repeated until the process load client is released. As a consequence, the cab goes to from in order to meet its client, and releases the ambient loading. Once loading has been released, it enters the ambient client.

The instruction given to the driver by a client to go from site from to site to, and the payment of the trip are described by

$$pay\ driver = pay^\Delta t_{16}[in^\Delta t_{17}\text{ c. in}^\Delta t_{18}\text{ wallet. in}^\Delta t_{19}\text{ money}]$$

$$trip\ from\ to\ c = trip^\Delta t_{20}[out^\Delta t_{21}\text{ client. out}^\Delta t_{22}\text{ from. in}^\Delta t_{23}\text{ to. pay driver}]$$

Whenever the client opens loading it means that the cab is present, and therefore the client may enter it. Consequently, the client enters the cab and releases an ambient trip, which the cab receives and opens. The process which is released moves the cab to its destination where it releases another synchronization ambient pay to tell the client to pay the trip.

The client itself, willing to go from from to to is described by

$$paid\ driver = paid^\Delta t_{28}[out^\Delta t_{29}\text{ money. out}^\Delta t_{30}\text{ wallet. out}^\Delta t_{31}\text{ driver. in}^\Delta t_{32}\text{ c}]$$

$$money\ client = money^\infty[open^\Delta t_{33}\text{ pay. out}^\Delta t_{34}\text{ wallet. out}^\Delta t_{35}\text{ c. in}^\Delta t_{36}\text{ driver.}]$$

$$in^\Delta t_{37}\text{ wallet. paid driver}]$$

$$wallet\ client = wallet^\infty[\text{ money client }] \ldots [\text{ money client }]$$

$$bye\ cab = bye^\Delta t_{24}[out^\Delta t_{25}\text{ c. in}^\Delta t_{26}\text{ cab. out}^\Delta t_{27}\text{ to}]$$
client from to = (νc)c∞[*(open^Δt38 recall. call from c) | recall^Δt39[]]  
open^Δt40 loading. in^Δt41 cab. trip from to c | open^Δt42 paid.  
out^Δt43 cab. bye cab | wallet client]

The ambient pay enters the client wallet and moves an ambient money to the driver wallet.
Once the ambient money enters the driver wallet, an ambient paid is released and send into client telling him to get out from the cab. The client opens it, leaves the cab, and sends the last synchronization ambient bye to the cab, instructing it to leave the site to.

The cab and the city are described by

driver = driver^∞[wallet^∞[money^∞[] | ... | money^∞[]]]
cab = cab^∞[rec X. open^Δt44 call. open^Δt45 trip. open^Δt46 bye. X | driver]
city = city^∞[cab | ... | cab | site_i^∞[client site_1 site_i | client site_1 site j | ...]]  
| ... | site_i^∞[...]]

In the example presented above, we have supposed that only the timer Δt7 for the ambient call expires, which determines the execution of the safety process recall. This was made only for the sake of simplicity. In order to simulate other possible scenarios, we can suppose that other timers may also expire:

- Δt1 - means that the loading ambient did not reach the ambient client, and a safety process should be released in order to announce cab to create another loading ambient;
- Δt16 - means that the pay ambient did not reach the ambient client, and a safety process should be released in order to announce cab to create another pay ambient;
- Δt20 - means that the trip ambient did not reach the ambient cab, and a safety process should be released in order to announce the client to create another trip ambient;
- Δt28 - means that the paid ambient did not reach the ambient client, and a safety process should be released in order to announce cab to create another paid ambient;
- Δt24 - means that the bye ambient did not reach the ambient cab, and a safety process should be released in order to announce the client to create another bye ambient;
- various timers over capabilities can be introduced.
5 Conclusion

Process algebra is the general study of distributed concurrent systems in an algebraic framework. In the past few years, some successful models have been formulated within this framework: CCS [14], Dπ [10], CSP [12], ACP [2], MA [4]. Each of these approaches includes restrictions on the nature of the systems which they attempt to model; for example, the inability to naturally describe properties of timing, probability and priority of events performed by the components of the system being modelled. In [1, 6, 7, 8, 9, 17], process algebraic methods are used to describe the temporal properties of distributed concurrent systems.

We have extended the mobile ambients by adding time constraints which transform communication channels, capabilities and ambients into temporal resources. Our work is strongly motivated by the existence of timers in TCP/IP communication protocols; the timers fit very well to the description of messages as mobile ambients. We extend with time restrictions a formalism designed for mobility in order to study various aspects related to time. We have provided a static semantic given by a typing system, and an operational semantics by a reduction relation. To describe passage of time we have given a (discrete) time-stepping function. We prove that the structural congruence and passage of time do not interfere with the typing system. A subject reduction results ensures that once well-typed, an ambient remains well-typed. We introduce a bisimulation relation related to the passage of time, and prove several results including that the passage of time cannot cause a nondeterministic behaviour.

To illustrate how the new formalism works, we present a timed extension of the cab protocol. For the sake of simplicity we have presented a variant of the protocol in which only one timer can expire. Other scenarios where other timers may expire are briefly described.
References


