
Time and Space Coordination of Mobile Agents

Gabriel Ciobanu

“A.I.Cuza” University, Faculty of Computer Science
Blvd. Carol I no.11, 700506 Iași, Romania
Romanian Academy, Institute of Computer Science
gabriel@info.uaic.ro

Summary. We study the evolution of agents able to move by using local knowledge and various migration means. We present two models of distributed systems with an explicit notion of location and some quantitative notions including time and capacity. We use these models for controlling mobility in open distributed systems by means of timers, bounded capacities and a simple migration primitive.

1.1 Introduction

The technology of agent systems, both hardware and software, is rather advanced. However design principles and techniques to define and verify their correct behaviour are at a more primitive stage. One approach in modelling the system behaviour has been the design of formal calculi in which the fundamental concepts underlying agent systems can be described and studied. In this paper we propose two formalisms for describing the behaviour of mobile agents in a distributed world. They are based on two existing formalisms called distributed π -calculus and ambient calculus, to which we add a network layer, a migration action, and other quantitative notions useful for coordination in time and space of the mobile agents.

A first approach could be given by defining a simple formal language for describing the systems in terms of their structure, namely how they are constructed from individual interacting agents. A semantic theory is defined in order to understand the behaviour of systems described in such a language, in terms of their ability to interact. Here a system consists of a finite number of agents which intercommunicate using a fixed set of named communication channels. This set of channels constitutes a connection topology through which all communication takes place; it includes both communication between agents, and between the system and its users.

The current agent systems are highly dynamic. Moreover, it is possible to create new communication links with other entities, and perhaps relinquish existing links. The π -calculus [10] and ambient calculus [4] are two formalisms seeking to address at least some dynamic aspects of such agents. The π -calculus includes the dynamic generation of communication channels and thus allows the underlying connection

topology to vary as systems evolve; it also allows private communication links to be established and maintained between agents. Ambient calculus is more oriented to dynamic aspects, working with ambients which represents units of movement.

Many concepts of the distributed systems are at most implicit (if not existing) in the π -calculus. Perhaps the most useful missing concept is that of domain (location), understood generally as a locus for computational activity. Thus one could view a distributed system as consisting of a collection of domains, each capable of hosting computational processes (agents), which in turn can migrate between domains. We use an extension of the π -calculus in which these domains have an explicit representation as locations. Distributed π -calculus is a formalism for agent systems in which dynamically created domains are hosts to resources which may be used by agents, and agents reside in domains, migrating eventually between domains for the purpose of using locally defined resources. Moreover, we use types to manage access control to resources in distributed systems. A domain may wish to restrict to certain agents the access to certain resources; we can think of resources having capabilities associated with them. Then domains may wish to distribute selectively to agents such capabilities on its local resources, and have agents manipulating these capabilities. Therefore in distributed π -calculus a system consists of a collection of domains, hosting agents, which can autonomously migrate between domains. These domains also host local channels on which agents communicate; but more generally these may be used to model local resources, accessible to agents currently located at the host domain. We describe the mobile agents by using a new primitive *go l. P* which enables migration between domains. If an agent executing *go l. P* is currently residing at *k*, it migrates to the domain *l* and continue there with the execution of *P*.

We define some quantitative ingredients over the systems described by the distributed π -calculus in order to coordinate the mobile agents in time and space. We add timers to distributed π -calculus. We assume a global notion of time, but we rather use a relative time of interactions given by timers. The global clock advances the time, and the interactions happen whenever the involved resources are available according to an interval of time given by timers of the interacting components (timers define timeouts for various resources, making them available only for a determined period of time). Using timers, we can control the concurrency of the components, and we can select between different choices in the system evolution. This provides a natural and flexible synchronization technique able to integrate and regulate dynamically the possible evolutions of the components. In a second formalism given by an extension of the mobile ambients, interaction in space relates to the ability to identify a specific domain in which a required and available resource exists. We use timers and timeout recovery processes over mobile ambients such that the resulting formalism provides a flexible coordination even in an open distributed environment where a number of possibly unknown entities are interacting. Several challenges related to these systems include finding the appropriate mechanisms to deliver adequate responses to time-critical demands, appropriate interaction methods, a decentralized control taking into account the time and space resources.

1.2 Timed Distributed π -calculus

Distributed π -calculus ($D\pi$) is an extension with types and locations of the π -calculus [10]. $D\pi$ is presented in [9], and provides a theoretical framework for describing communications between distributed processes with restricted resource access. In [6] we extend the distributed π -calculus by introducing timers over channel names in order to define timeouts for communications. The resulting formalism is called timed distributed π -calculus ($tD\pi$). Over this formalism we define a coordination of the whole system by assigning specific values to timers and defining a set of time constraints [7].

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

The syntax of *Input* and *Output* communication uses a pair of processes. For instance, an *Input* expression $a^{\Delta t}?(X : T).(P, Q)$ evolves to P whenever a communication is established during the interval of time given by Δt ; otherwise it evolves to Q . The variable X is considered bound only in P and we should provide its type T ; the type system is presented in [6].

Table 1: Syntax of timed distributed π -calculus

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

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

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

We define type environment Γ as sets of *location types*. The purpose of the type environment associated with a specific process is to restrict the range of accessible resources the process can access. Formally, $\Gamma \subseteq \mathcal{L} \times K$ is a relation associating to a location name a location type. A location type is a set of *location capabilities* which may contain *channel types*, *move* capability (i.e., permission to migrate to that location), or *channel creation* capability (i.e., permission to create channels).

We extend the channel types of $D\pi$ with timers of the form Δt . Communication is now permitted on channels only in the interval of time given by the timer value t (i.e. until the timer of the channel type expires). These timers define the existence of the channel types inside the type environment. Timers decrease with each "tick" of an universal clock (we assume that we have an universal clock). Upon expiration, the channel types are discarded. Timers are created once with the channel types, and are activated when the types are added to the type environment. When the processes receive new channel names, types for the new channels become available. It means that the processes can communicate on the new channels according to the new types. For example, if a process receives through an input channel a located name $a@k$, then it gains the capability to move to location k , and to communicate on channel a .

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

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

Well-typedness of processes is defined by a set of static rules (a detailed presentation of the static typing rules is given in [6]). These rules express the behaviour of a process with regard to its types. The subtyping relation $<$: is similar to the subtyping relation of $D\pi$, and $\Gamma(k)$ represents a type environment at location k . If a process tries to communicate on a channel for which it has no capability, it can still be well-typed if the alternative process Q is well-typed. Q is called the *safety process*.

Definition 1. (Tagged time-stepping function)

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

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

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

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

Lemma 1. (Well-typedness is preserved by the cleanup function)

If $\Gamma \Vdash l[[P]]_\Delta$ then $\Gamma \Vdash \psi(l[[P]]_\Delta)$.

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

$$\begin{aligned} \text{(R}_\Gamma\text{-IDLE)} \quad & \frac{l[[P]]_\Gamma \rightarrow}{l[[P]]_\Gamma \rightarrow \phi_\Delta(l[[P]]_\Gamma)} \\ \text{(R}_\Gamma\text{-COM1)} \quad & \frac{\Gamma(l, a) <: res\{r\langle T \rangle\}}{l[[\alpha^{At}! \langle v \rangle \cdot (P, Q)]]_\Delta \mid l[[\alpha^{At}?(X : T) \cdot (P', Q')]]_\Gamma \rightarrow \psi(l[[P]]_\Delta) \mid \psi(l[[P'\{^V_X\}]]_{\Gamma \cap \{v @ l : T\}})} \\ \text{(R}_\Gamma\text{-COM2)} \quad & \frac{\Gamma(l, a) <: res\{ro\langle T \rangle\}}{l[[\alpha^{At}! \langle v \rangle \cdot (P, Q)]]_\Delta \mid l[[\alpha^{At}?(X : T) \cdot (P', Q')]]_\Gamma \rightarrow \psi(l[[P]]_\Delta) \mid \psi(l[[P'\{^V_X\}]]_\Gamma)} \end{aligned}$$

$$\begin{array}{c}
(\text{R}_f\text{-PAR}) \frac{N \rightarrow N' \quad M \rightarrow M'}{N \mid M \rightarrow N' \mid M'} \quad (\text{R}_f\text{-RES}) \frac{N \rightarrow N'}{(va@l : T)N \rightarrow (va@l : T)N'} \\
(\text{R}_f\text{-CONG}) \frac{N \equiv N' \quad N \rightarrow M \quad M \equiv M'}{N' \rightarrow M'}
\end{array}$$

We have several results, and here we mention two of them (more details are in [6]). These results claim that the passage of time does not interfere with the typing system, and that once well-typed, a process remains well-typed.

Proposition 1. *If $\Gamma \Vdash l[[P]]_\Delta$ then $\Gamma \Vdash \phi_\Delta(l[[P]]_\Delta)$.*

Theorem 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 $N \rightarrow N'$ then $\Gamma \Vdash N$ if and only if $\Gamma \Vdash N'$.*

1.2.1 Interaction in Timed Distributed π -calculus

Throughout this paper we consider the following example which can motivate and illustrate our approach. We assume the situation of a student finishing his lectures, moving out the university building and looking for an available vehicle in order to move to another location. In front of the university there are a tram stop, a bus stop, and taxicabs. The student has the possibility to use any of the three types of vehicles: bus, tram and cab to reach the target location. The student can have a priority among them: the tram has the highest priority, followed by the bus, and finally the cab. Our scenario involves a tram, a bus and two cabs (a hired cab and an available cab). The trams and the buses are moving according to predetermined schedules which are known by the student.

In what follows we describe a system that contains two location and a bus. The other elements of the initial system (tram and cabs) could be described in a similar manner.

$$\begin{aligned}
in_{univ} &= bus_{univ}^{At_2} ?(i). bus_{univ}^{At_1} !\langle stud_{univ} \rangle. in_{univ} \\
in_{camp} &= bus_{camp}^{At_3} ?(i). bus_{camp}^{At_4} !\langle stud_{camp} \rangle. in_{camp} \\
bus &= go\ camp. bus_{camp}^{At_6} !\langle stud_1 \rangle. bus_{camp}^{At_7} ?(i). go\ univ. \\
&\quad bus_{univ}^{At_8} !\langle stud_2 \rangle. bus_{univ}^{At_5} ?(i). bus \\
system &= univ[[in_{univ}]] \mid bus \mid camp[[in_{camp}]]
\end{aligned}$$

where

- $stud_1$ represents the number of students getting out of bus at $camp$,
- $stud_2$ represents the number of students getting out of bus at $univ$,
- $stud_{univ}$ represents the number of students getting into bus at $univ$, and
- $stud_{camp}$ represents the number of students getting into bus in $camp$.

A possible evolution of the system is given by

$$system = camp[[in_{camp}]] \mid univ[[in_{univ} \mid bus]] =$$

$$\begin{aligned}
&= \text{camp}[[in_{\text{camp}}]] \mid \text{univ}[[bus_{\text{univ}}^{At_1}!\langle stud_{\text{univ}} \rangle . bus_{\text{univ}}^{At_2}?(i).in_{\text{univ}} \mid bus_{\text{univ}}^{At_5}?(i). \\
&\quad go \text{ camp}.bus_{\text{camp}}^{At_6}!\langle stud_1 \rangle . bus_{\text{camp}}^{At_7}?(i).go \text{ univ}.bus_{\text{univ}}^{At_8}!\langle stud_2 \rangle . bus]] \\
&\rightarrow \text{camp}[[in_{\text{camp}}]] \mid \text{univ}[[bus_{\text{univ}}^{At_2}?(i).in_{\text{univ}} \mid go \text{ camp}. \\
&\quad bus_{\text{camp}}^{At_6}!\langle stud_1 \rangle . bus_{\text{camp}}^{At_7}?(i).go \text{ univ}.bus_{\text{univ}}^{At_8}!\langle stud_2 \rangle . bus]] \\
&\rightarrow \text{camp}[[bus_{\text{camp}}^{At_3}?(i).bus_{\text{camp}}^{At_4}!\langle stud_{\text{camp}} \rangle . in_{\text{camp}} \mid bus_{\text{camp}}^{At_6}!\langle stud_1 \rangle . \\
&\quad bus_{\text{camp}}^{At_7}?(i).go \text{ univ}.bus_{\text{univ}}^{At_8}!\langle stud_2 \rangle . bus]] \mid \text{univ}[[bus_{\text{univ}}^{At_2}?(i).in_{\text{univ}}]] \\
&\rightarrow \text{camp}[[bus_{\text{camp}}^{At_4}!\langle stud_{\text{camp}} \rangle . in_{\text{camp}} \mid \\
&\quad bus_{\text{camp}}^{At_7}?(i).go \text{ univ}.bus_{\text{univ}}^{At_8}!\langle stud_2 \rangle . bus]] \mid \text{univ}[[bus_{\text{univ}}^{At_2}?(i).in_{\text{univ}}]] \\
&\rightarrow \text{camp}[[in_{\text{camp}} \mid go \text{ univ}.bus_{\text{univ}}^{At_8}!\langle stud_2 \rangle . bus]] \mid \text{univ}[[bus_{\text{univ}}^{At_2}?(i).in_{\text{univ}}]] \\
&\rightarrow \text{camp}[[in_{\text{camp}}]] \mid \text{univ}[[bus_{\text{univ}}^{At_2}?(i).in_{\text{univ}} \mid bus_{\text{univ}}^{At_8}!\langle stud_2 \rangle . bus]] \\
&\rightarrow \text{camp}[[in_{\text{camp}}]] \mid \text{univ}[[in_{\text{univ}} \mid bus]] = \text{system}
\end{aligned}$$

We use different (coloured) letters to emphasize the interacting elements.

We do not have a notion of *capacity* for the mobile process bus. In this description there is no way to remember the number of persons inside the bus, or the number of the persons who entered or exited the bus at each stop. Moreover, the notion of space used in $tD\pi$ is flat. A more realistic account of physical distribution is obtained using a hierarchical representation of space (locality) as in the ambient calculus [4].

1.3 Mobile Ambients with Quantitative Ingredients

Ambient calculus is a formalism for describing distributed and mobile computation in terms of ambients [4]. In contrast with other formalisms for mobile processes such as the π -calculus [10] where the computational model is based on the notion of *communication*, the ambient calculus is based on the notion of *movement*. An ambient is the unit of movement. The mobility of an ambient is controlled by the capabilities *in*, *out*, and *open*. In an ambient we have processes which may exchange messages. Several variants (e.g., [3, 13]) have been proposed by adding and/or removing features of the original calculus [4]. We extend mobile ambients by adding timers to ambients, capabilities, input and output actions. A timer Δt of each temporal resource makes the resource available only for a determined period of time t . We use timeout recovery processes acting when a timer expires. We define the available resources by a notion of capacity, and the resources consumed by an ambient when it moves into another ambient. A notion of domain defines the local computation resources making the

ambient able to plan its movements and its navigation means. We think of a domain when we speak about proximity, local computation support or local knowledge. A movement changes the domain (local computation support, knowledge), and the ambient has a different domain when arriving at the destination. The ambients can adapt their behaviour according to the additional ingredients, depending mainly on timers, capacity and local computation support (knowledge). We denote by $n_{(l,h,d)}^{\Delta t}[P]$ the fact that an ambient n has assigned a timer Δt , a *capacity* l representing the number of its free resources, a *weight* h counting the resources needed by a process to be executed, and a domain d . According to a global clock, the timer is decreasing its value; the timer Δt expires when $t = 0$. We use a pair $(n_{(l,h,d)}^{\Delta t}[P], Q)$ to denote a timed ambient, where Q represents a *safety process* used whenever the timer expires. If nothing happens in t units of time, the ambient $n_{(l,h,d)}^{\Delta t}$ is dissolved, process P running inside the ambient is reduced to $\mathbf{0}$, and safety process Q is executed. If $Q = \mathbf{0}$ we can simply write $n_{(l,h,d)}^{\Delta t}[P]$ instead of $(n_{(l,h,d)}^{\Delta t}[P], Q)$. The behaviour of an untimed mobile ambient is given by using ∞ instead of Δt , and 0 instead of h , l and d . We use a pair of processes for the input, output and movement processes. The process $open^{\Delta t}n.(P, Q)$ evolves to P whenever in time Δt the process becomes sibling to an ambient n ; otherwise evolves to Q . The process $!\langle m \rangle^{\Delta t}.(P, Q)$ evolves to P whenever in time Δt the process becomes sibling to a process which is willing to capture the name m ; otherwise evolves to Q .

We define various new technical ingredients. The domain of an ambient n represents the local knowledge given by the computation support (operating system, specific software); it includes information about the ports of movement, specific transport means and interaction links. The domain d is different for each location where the ambient can move using its movement capabilities.

The syntax of the coordinated mobile ambients (cMA) is defined in the following table:

Table 2: <i>Syntax of coordinated mobile ambients</i>				
n, m		names	$P, Q ::=$	processes
x, y		variables	$\mathbf{0}$	inactivity
M	$::=$	capabilities	$M^{\Delta t}.(P, Q)$	movement
	$in\ n$	can enter n	$(n_{(l,h,d)}^{\Delta t}[P], Q)$	ambient
	$out\ n$	can exit n	$P \mid Q$	composition
	$open\ n$	can open n	$(\nu m)P$	restriction
			$!\langle m \rangle^{\Delta t}.(P, Q)$	output action
			$?(x)^{\Delta t}.(P, Q)$	input action
			$*P$	replication
			$P + Q$	choice

The coordinated mobile ambients use a discrete time. The passage of time is described by a (discrete) time-stepping function ϕ_{Δ} defined over the set \mathcal{P} of cMA-processes. The possible actions are performed at every tick of a universal clock. ϕ_{Δ} affects the ambients and their capabilities which are not consumed. The consumed capabilities disappear together with their timers. If a capability or ambient has the timer equal to ∞ , we use the equality $\infty - 1 = \infty$ when applying the time-stepping

function ϕ_A . This function modifies a process accordingly with the global passage of time.

Definition 2. We define the time-stepping function $\phi_A : \mathcal{P} \rightarrow \mathcal{P}$ by

$$\phi_A(P) = \begin{cases} M^{A(t-1)}.(R, Q) & \text{if } P = M^{At}.(R, Q), t > 0 \\ Q & \text{if } P = M^{At}.(R, Q), t = 0 \\ \phi_A(R) \mid \phi_A(Q) & \text{if } P = R \mid Q \\ (vn)\phi_A(R) & \text{if } P = (vn)R \\ (n_{(l,h,d)}^{A(t-1)}[\phi_A(R)], Q) & \text{if } P = (n_{(l,h,d)}^{At}[R], Q), t > 0 \\ Q & \text{if } P = (n_{(l,h,d)}^{At}[R], Q), t = 0 \\ P & \text{if } P = *R \text{ or } P = \mathbf{0} \\ \phi_A(R) + \phi_A(Q) & \text{if } P = R + Q \end{cases}$$

The local communication given by the interaction between $!\langle m \rangle^{At}.(P, Q)$ and $?(x)^{At}.(R, Q)$ inside the same ambient does not consume time, so it is not included in the definition of ϕ_A . To see how this function is used, look at the reduction rules (Table 4).

The semantics of the timed mobile ambients is given by two relations: a structural congruence relation and a reduction relation. Structural congruence provides a way of re-arranging expressions such that the interacting parts can be brought together. The structural relation \equiv_p over the timed mobile processes is the least relation satisfying the axioms and rules from the following Table:

Table 3: Structural congruence

(S-Refl)	$P \equiv_p P$	(S-Sym)	$P \equiv_p Q$ implies $Q \equiv_p P$
(S-Trans)	$P \equiv_p R, R \equiv_p Q$ implies $P \equiv_p Q$		
(S-Res)	$P \equiv_p Q$ implies $(vn)P \equiv_p (vn)Q$		
(S-LPar)	$P \equiv_p Q$ implies $R \mid P \equiv_p R \mid Q$		
(S-RPar)	$P \equiv_p Q$ implies $P \mid R \equiv_p Q \mid R$		
(S-Repl)	$P \equiv_p Q$ implies $*P \equiv_p *Q$		
(S-Amb)	$P \equiv_p Q$ and $R \equiv_p R'$ implies $(n_{(l,h,d)}^{At}[P], R) \equiv_p (n_{(l,h,d)}^{At}[Q], R')$		
(S-Cap)	$P \equiv_p Q$ and $R \equiv_p R'$ implies $M^{At}.(P, R) \equiv_p M^{At}.(Q, R')$		
(S-Par Com)	if $weight(P) = 0$ then $P \mid Q \equiv_p Q \mid P$		
(S-Par Assoc)	$(P \mid Q) \mid R \equiv_p P \mid (Q \mid R)$		
(S-Repl Par)	$*P \equiv_p P \mid *P$		
(S-Res Res)	$(vn)(vm)P \equiv_p (vm)(vn)P$ if $n \neq m$		
(S-Res LPar)	$(vn)(P \mid Q) \equiv_p P \mid (vn)Q$ if $(n) \notin fn(P)$		
(S-Res RPar)	$(vn)(P \mid Q) \equiv_p (vn)P \mid Q$ if $(n) \notin fn(Q)$		
(S-Res Amb)	$(vn)(m_{(l,h,d)}^{At}[P], Q) \equiv_p (m_{(l,h,d)}^{At}[(vn)P], Q)$ if $n \neq m$		
(S-Zero Par)	$P \mid \mathbf{0} \equiv_p P$	(S-Zero Res)	$(vn)\mathbf{0} \equiv_p \mathbf{0}$
		(S-Zero Repl)	$*\mathbf{0} \equiv_p \mathbf{0}$

Resources are preserved through reduction only in a *closed* system, namely a system which is surrounded by an ambient which cannot be opened, and any ambient can pass through it. In an *open* system, the resources are not preserved through reduction because a process may acquire new resources from the environment, or transfer resources to the environment. Some resources may become restricted, and so unavailable for any other process, e.g. $(vn)(n_{(l,h,d)}^{At}[P], Q)$.

For processes $M^{dt}.(P, Q)$, $!\langle m \rangle^{dt}.(P, Q)$ and $?(x)^{dt}.(P, Q)$ the timers of P are activated only after the disappearance of M^{dt} , $!\langle m \rangle^{dt}$, and $?(x)^{dt}$, respectively. To preserve the timers of P , we define the time-preserving function $\psi : \mathcal{P} \rightarrow \mathcal{P}$ by $\phi_A(\psi(P)) = P$. We denote by \rightarrow the fact that rules (R-In), (R-Out), (R-Open) and (R-Com) cannot be applied. Note that the ambients can interact using these rules only if they are on the same domain (this expresses a certain topological requirement suggesting that they are close enough to interact), and the network support allows the application of these rules. The evolution is given by the reduction rules of Table 4. The function ϕ_A decreases the timers, and for the expired timers it discards the actions, capabilities and ambients. If one process evolves by one of the rules (R-In), (R-Out), (R-Open) and (R-Amb), while another one does not perform any reduction, then one of the rules (R-LPar), (R-RPar) should be applied. If more than one process evolve in parallel by applying one of the rules (R-In), (R-Out), (R-Open) and (R-Amb), then rule (R-Par) should be applied. When rules (R-In), (R-Out), (R-Open) and (R-Com) cannot be applied anymore, the rule (R-GTProgress) is applied to simulate the global passage of time, and so to permit the ambients to participate in other reductions in the next unit of time.

Table 4: Reduction rules

(R-GTProgress)	$\frac{P \rightarrow}{P \rightarrow \phi_A(P)}$	
(R-In)	$\frac{h' \leq l''}{\frac{(n_{(l', h', d)}^{dt} [in^{dt} m.(P, P') Q], S') (m_{(l'', h'', d)}^{dt''} [R], S'') \rightarrow}{(m_{(l' - h', h'', d)}^{dt''} [(n_{(l', h', d)}^{dt} [\psi(P) Q], S') R], S'')}}}$	
(R-Out)	$\frac{h'' \leq l}{\frac{(k_{(l, h, d)}^{dt} [(m_{(l', h', d')}^{dt'} [(n_{(l'', h'', d'')}^{dt''} [out^{dt} m.(P, P') Q], S'') R], S'), S) \rightarrow}{(k_{(l - h'', h, d)}^{dt} [(n_{(l'', h'', d'')}^{dt''} [\psi(P) Q], S'') (m_{(l' + h'', h', d')}^{dt'} [R], S'), S)}}$	
(R-Open)	$\frac{-}{\frac{(m_{(l', h', d')}^{dt'} [open^{dt} n.(P, P') (n_{(l'', h'', d'')}^{dt''} [Q], S'')], S') \rightarrow}{(m_{(l' + l'', h', d')}^{dt'} [\psi(P) Q], S')}}}$	
(R-Com)	$\frac{-}{!\langle m \rangle^{dt}.(P, Q) ?(x)^{dt'}.(P', Q') \rightarrow P P' \{m/x\}}$	
(R-Amb)	$\frac{P \rightarrow Q}{(n_{(l, h, d)}^{dt} [P], R) \rightarrow (n_{(l, h, d)}^{dt} [\psi(Q)], R)}$	
(R-LPar)	$\frac{P \rightarrow Q}{R P \rightarrow R Q}$	(R-RPar) $\frac{P \rightarrow Q}{P R \rightarrow Q R}$
(R-Par)	$\frac{P \rightarrow Q, P' \rightarrow Q'}{P P' \rightarrow Q Q'}$	(R-Res) $\frac{P \rightarrow Q}{(vn)P \rightarrow (vn)Q}$
(R-Struct)	$\frac{P' \equiv_p P, P \rightarrow Q, Q \equiv_p Q'}{P' \rightarrow Q'}$	

We denote by $P \xrightarrow{\phi_A}_t Q$ the fact that process P evolves to process Q after applying the rule (R-GTProgress) for $t \geq 0$ times, and with $t\phi_A(R)$ the fact that function ϕ_A is applied t times to R . The passage of time cannot cause a nondeterministic behaviour.

Proposition 2. *If $P \equiv_p Q$, $P \xrightarrow{\phi_A}_t P'$ and $Q \xrightarrow{\phi_A}_t Q'$ then $P' \equiv_p Q'$.*

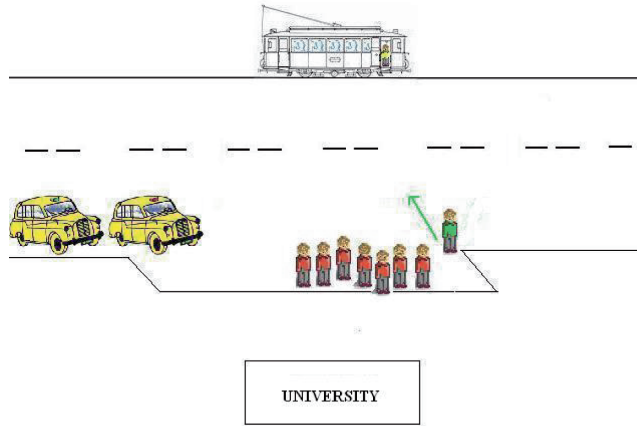
We can compare the behaviour of the agent systems by defining a bisimulation in cMA which requires processes to match their time passages. A binary relation \mathcal{R} over processes is a strong simulation if whenever $(P, Q) \in \mathcal{R}$, if $P \xrightarrow{\phi_i}_t P'$ then there exists Q' such that $Q \xrightarrow{\phi_i}_{t'} Q'$, $t = t'$ and $(P', Q') \in \mathcal{R}$. A binary relation \mathcal{R} is said to be a strong bisimulation if both \mathcal{R} and \mathcal{R}^{-1} 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 $t \geq 0$. We say that P and Q are weakly bisimilar, written $P \approx_t Q$, if there exists a weak bisimulation \mathcal{R} such that $P\mathcal{R}Q$.

Proposition 3.

- i) If $P \equiv_p Q$ then $P \sim_t Q$, for all $t \geq 0$.
- ii) \sim_t is an equivalence relation, and $P \sim_t Q$ implies $P \approx_t Q$.
- iii) \approx_t is an equivalence relation.

1.4 Evolution in Coordinated Mobile Ambients

We use the following example as a metaphor which motivates and illustrates our approach. We assume the situation of a process (student) having the goal of moving to a well-defined location. There are several transportation means with different properties: tram, bus, and taxi. Let us consider that the student can reach a tram stop, a bus stop, and two cabs. The student has the possibility to use any of the three types of transportation to reach the target location. The bus and the tram could move according to a predetermined schedule. Based on the local computation resources, the student can plan the movement to the target location. For an ambient A , $free_resources(A)$ represents the (dynamically evolving) capacity l of A .

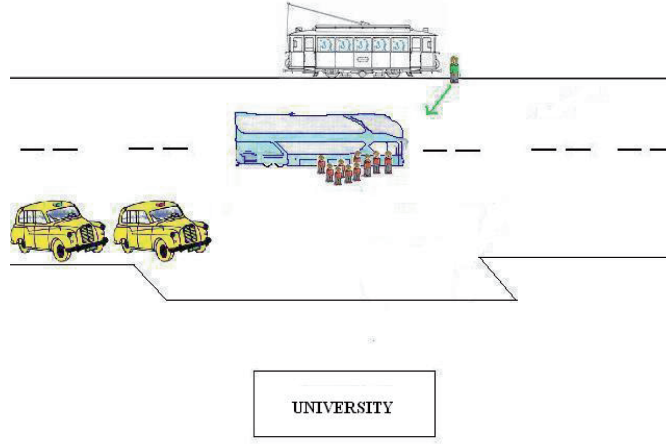


We encode the agents involved in this example; each of them is encoded as an ambient having the corresponding label. An ambient *student* has the possibility to use any

of the three transport ambients *tram*, *bus* and *cab* to reach the destination. Since the transport means have different costs, it is possible to define a priority among them: the *tram* has the highest priority, followed by the *bus*, and finally by the *cab*. Based on the local computation resources, an agent can plan its movement from the current location to the target location where a specific goal should be fulfilled.

$univ_{(l_univ, h_univ, d_univ)}^{\infty} [student_{(1,1,d_student)}^{\infty} [in^{At_{12}} tram]] |$
 $tram_{(l_tram, h_tram, d_tram)}^{\infty} [out^{At_5} univ]]$ and $d_student \neq d_tram$

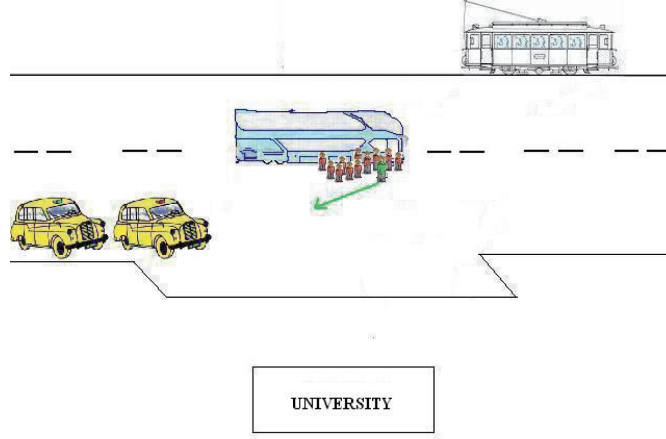
Even if the student has a capability $in^{At_{12}} tram$ expressing the wish of using the tram, the *tram* is not close enough to the *student* ($d_student \neq d_tram$), and so it cannot be used by the student. After t_5 units of time the tram activates its capability $out^{At_5} univ$ and moves out of the domain *univ*.



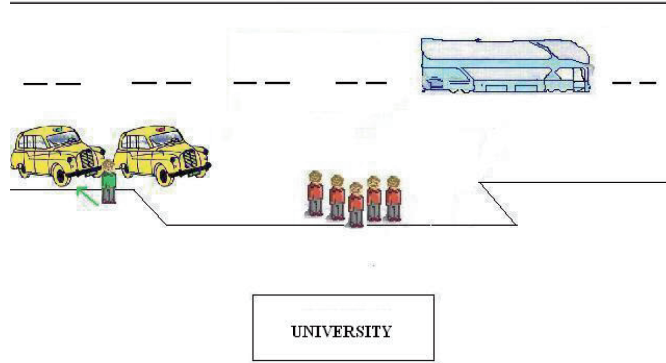
$univ_{(l_univ, h_univ, d_univ)}^{\infty} [student_{(1,1,d_student)}^{\infty} [in^{At_{10}} bus]] |$
 $bus_{(l_bus, h_bus, d_bus)}^{\infty} [out^{At_2} univ]] | tram_{(l_tram, h_tram, d_tram)}^{\infty} [in^{At_6} camp]$
 and $d_student = d_bus$

By the capability $in^{At_6} camp$ the tram is moving to a new location (domain) denoted by *camp*. The student has a capability $in^{At_{10}} bus$ expressing the wish of using the bus, and *bus* is close to the *student*. If the *bus* has free resources ($free_resource(bus) > 0$), then the *student* can use the *bus*. However the *bus* does not have free resources (its capacity is 0), and so the student looks for another transport means in his domain by using the local computation resources (knowledge).

$univ_{(l_univ, h_univ, d_univ)}^{\infty} [student_{(1,1,d_student)}^{\infty} [in^{At_{10}} bus]] |$
 $bus_{(0, h_bus, d_bus)}^{\infty} [out^{At_2} univ]]$ and $d_student = d_bus$



Then the *student* looks for a *cab*; if the *cab* is in the domain of the *student* and it has free resources $free_resources(cab) > 0$, then the *student* enters the *cab*. One cab is not available (its capacity is 0). The other one is available with capacity 4; once hired, its capacity becomes 0 (it does not depend on the number of passengers).

$$\begin{aligned}
 & univ_{(l_univ, h_univ, d_univ)}^{\infty} [student_{(1, 1, d_student)}^{\infty} [in^{At_{14}} cab] \mid \\
 & cab_{(4, 4, d_cab1)}^{\infty} [out^{At_7} univ] \mid cab_{(0, 4, d_cab2)}^{\infty} [out^{At_7} univ] \mid \\
 & bus_{(l_bus, h_bus, d_bus)}^{\infty} [in^{At_6} camp] \text{ and } d_student = d_cab1
 \end{aligned}$$


If none of the above steps are possible, the time-stepping function ϕ_A is applied, and then the evolution conditions are verified again.

This example describes the coordination in time and space given by the proposed formalism based on ambient calculus. The new ingredients used over mobile ambients are timers, timeout recovery processes, domains and capacity. A slight

improvement of this description can allow to know the number of persons inside the bus, or the number of the persons who entered or exited the bus at each stop, etc.

1.5 Conclusion

The purpose of a coordination model is to enable the integration of a number of components (processes, objects, agents) in such a way that the resulting ensemble can execute as a whole, forming a software system with desired characteristics and functionalities which possibly takes advantage of parallel and distributed systems. Such models for agent systems are closely related to other software engineering approaches such as service-oriented architectures, component-based systems and related middleware platforms. Coordination abstractions are perceived as essential to design and support the working activities of agent societies; in other cases, service coordination, orchestration, and choreography can become essential aspects of the next generations of systems based on Web services.

The formal approaches presented in this paper are based on certain new ingredients: explicit domains, timers, timeout recovery processes and capacity. We consider first a flat representation of the space (timed distributed π -calculus), and then a hierarchical representation of the space by using ambient calculus extended with the new ingredients. Formal semantics and some basic results are presented. A non-monotonic behaviour of the systems are given by the timeout recovery processes (agents). We use an example to describe the coordination in time and space of a moving agent.

We use these models for controlling mobility in open distributed systems by means of timers, bounded capacities and domains. Both timed distributed π -calculus and coordinated mobile ambients use a discrete and relative time given by timers, based on a global clock whose tick decreases the timers. Timers are used to restrict the interaction between components, and both types and timers are used to control the resource availability. Timeout can be specified for agent actions such as in and out. A capacity of an ambient roughly means how many ambients can enter it (more precisely, an ambient can enter another if the capacity of the latter is greater than the weight of the former). Finally, the domains manage the computation resources and information of an ambient in order to plan and realize a movement.

In both formalisms we have a separation of the coordination aspects from the computation aspects. Interaction in time and space can be coordinated by assigning specific values to timers and capacities, and by a set of rules restricting the evolution of the system. The recovery processes (agents) can define actions when we have more than one interaction choice.

A related paper is [1] where more technical results are presented. Timed distributed π -calculus is presented in [6] as an extension of the distributed π -calculus by timers over channel names in order to define timeouts for communications. Over this formalism we define a coordination of the whole system by assigning specific values to timers and defining a set of time constraints [7].

Acknowledgements

The author thanks to Bogdan Aman for his collaboration on the concerning timed mobile ambients, and to Cristian Prisacariu for his collaboration on the timed distributed π -calculus. Many thanks to Oana Agrigoroaiei for useful comments.

References

1. B. Aman, G. Ciobanu. Timers and Proximities for Mobile Ambients. *Lecture Notes in Computer Science* vol.4649, Springer, 2007.
2. N. Busi, P. Ciancarini, R. Gorrieri, G. Zavattaro. Coordination Models: A Guided Tour. *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer, 6–24, 2001.
3. M. Bugliesi, G. Castagna, S. Crafa. Boxed Ambients. *Theoretical Aspects of Computer Software*, *Lecture Notes in Computer Science* vol.2215, Springer, 38–63, 2001.
4. L. Cardelli, A. Gordon. Mobile Ambients. *Foundations of Software Science and Computation Structures*, *Lecture Notes in Computer Science* vol.1378, Springer, 140–155, 1998.
5. G. Ciobanu. Interaction in Time and Space. *Proceedings of Foundations of Interactive Computation*, 45–61, 2007.
6. G. Ciobanu, C. Prisacariu. Timers for Distributed Systems. *Electronic Notes in Theoretical Computer Science* vol.164(3), 81–99, 2006.
7. G. Ciobanu, C. Prisacariu. Coordination by Timers for Channel-Based Anonymous Communications. *Electronic Notes in Theoretical Computer Science* vol.175, 3–17, 2007.
8. M. Hennessy. *A Distributed pi-calculus*. Cambridge University Press, 2007.
9. M. Hennessy, J. Riely. Resource access control in systems of mobile agents, *Information and Computation* vol.173(1), 82–120, 2002.
10. R. Milner. *Communicating and Mobile Systems: the π -calculus*. Cambridge University Press, 1999.
11. G.A. Papadopoulos. Models and Technologies for the Coordination of Internet Agents: A Survey. *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer, 25–56, 2001.
12. G.A. Papadopoulos, F. Arbab. Coordination Models and Languages. *Advances in Computers* vol.46, Academic Press, 329–400, 1998.
13. D. Teller, P. Zimmer, D. Hirschhoff. Using Ambients to Control Resources. *Concurrency Theory*, *Lecture Notes in Computer Science* vol.2421, Springer, 288–303, 2002.

Advances in Intelligent and Distributed Computing
Proceedings of the 1st International Symposium on
Intelligent and Distributed Computing IDC 2007,
Craiova, Romania, October 2007
Badica, C.; Paprzycki, M. (Eds.)
2008, XIV, 310 p. 98 illus., Hardcover
ISBN: 978-3-540-74929-5