Electronic Notes in Theoretical Computer Science 85 No. 1 (2004)
URL: http://www.elsevier.nl/locate/entcs/volume85.html 25 pages

Formalisms for Mobile Resource Control

David Teller [1,2,3]

LIP
ENS Lyon
France

Abstract

Failing to control resources in mobile, concurrent and distributed systems may lead to important breakdowns or Denial of Service-like attacks. In order to address this problem, we present enhanced versions of several calculi for mobile and distributed computing, namely NBA, Seals, Nomadic π and Kells. In each case, we make the formalism resource-conscious and define a type system in order to guarantee statically compliance with resource control policies. Comparing the solutions we proposed for these calculi, we try and define the necessities of resource-control in mobile and distributed formalisms.

1 Introduction

The latest generations of computer software and hardware make use of numerous new technologies in order to enhance flexibility or performances. Most current systems may be dynamically reconfigured or extended, allow concurrency or use it, and can communicate with other systems. This flexibility, however, induces the multiplication of subsystems and protocols. In turn, this multiplication greatly increases the possibility of bugs, the feasibility of attacks and the sensitivity to possible breakdown of individual subsystems.

Several formalisms attempt to provide mechanisms adapted to mobility and distribution, as well as to address possible circumstances of bugs, attacks and sensitivity to failures. One important problem which current formalisms do not address directly is that of resource control: to guarantee (statically) some bounds on resources such as memory, so as to remove the risk of Denial of Service attacks or similar misbehaviors in distributed systems.

[1] We would like to thank Daniel Hirschkoff for his time and his insightful suggestions during this work.
[2] This work is partially supported by IST Global Computing PROFUNDIS and the *Action spécifique Méthodes formelles pour la mobilité*.
[3] Email: David.Teller@ens-lyon.fr

By *resource*, we mean any entity which may *at will* be acquired, used, then released. Thus, this notion encompasses ports, CPUs, computers or RAM, but not time, or (presumably) money. A *resource-controlled system* is a system in which no subsystem will ever require more resources than may be available. In [16], we introduced *Controlled Ambients* as an enrichment of Cardelli and Gordon's Mobile Ambients (MA) [4] for the specification and static checking of resource control policies for mobile and distributed systems. The main points of our approach are:

i The use of *cocapabilities* to make both the *source site* and *target site* of each movement aware of the mobility, resulting in a 3-way synchronization which actually corresponds to mechanisms used when implementing MA, as observed in [9].

ii A resource control policy to define the number of resources occupied by each site in its parent site, as well as the total amount of resources available in each site for use by its *direct* subsites.

iii A type system to enforce resource control policies. Correctness of this type system guarantees that no evolution of the system will end up in a situation where more (sub)ambients than allowed are present simultaneously in a given ambient.

Recently, a broad range of calculi have been designed for the handling of localities and process mobility. Some calculi are based on site-subsite relations, site migrations and "mostly local" communications, such as Seals [18] or Boxed Ambients [2], later extended to NBA [3]. Some others are based on migrations of active/passive processes with pattern-based n-way communications, such as Distributed Join [8], then evolved into the M-Calculus [13] and later into Kell [14]. Some more are based on site-agents relations, distant communications and agent migrations, such as Nomadic π [17], the extension of π which serves as the basis of the Nomadic Pict programming language. All these formalisms offer a notion which can easily model resource usage and resource availability. Others, such as Dπ [10], are based on distant communication of names and distant process migration, which makes them more difficult to control using our definition of resources – see the conclusion for further discussion on this.

In the present paper, we try to reach a better understanding of the mechanisms for resource control in distributed settings, and to determine the set of primitives necessary to provide such a control. In order to do this, we extend and adapt the concepts and techniques we have devised for *Controlled Ambients* to adapt them to a range of other process calculi, hence presenting *Controlled NBA*, *Controlled Seals*, *Controlled Kell* and *Controlled Nomadic* π. For each calculus, we present the required enrichments to operational semantics and introduce a type system for resource control. Due to lack of space, we only present the main features of each calculus and type system; full definitions, as well as proofs, are available in a full version of this re-

port [15]. The resource control policies that we propose are illustrated on a simple but sufficiently expressive running example that is common to all formalisms. In defining these calculi and type systems, we have tried to keep a uniform approach, staying faithful to the principles $(i) - (iii)$ outlined above, while keeping our modifications as minimal as possible. In each case, we motivate our choices and discuss alternative solutions where appropriate, in order to shed light on the specifics of each calculus.
We successively treat NBA (Sect. 2), Seals (Sect. 3) Nπ (Sect. 4) and Kells (Sect. 5). The last section sums up the lessons learned from the comparison of these calculi and type systems, and discusses possible future extensions.

2 Controlling NBA

NBA, introduced in [3], are an evolution of Boxed Ambients (BA) [2] which allows mobility control and reduces communication interferences. As in BA and Mobile Ambients, a system is represented by a hierarchy of sites called *ambients*. Each ambient may contain concurrent processes. Processes may spawn new ambients locally, execute *capabilities*, hence moving their enclosing ambient upwards (out of their enclosing ambient) or downwards (in a brother ambient), and they can communicate locally both names and capabilities. As in BA and as opposed to MA, ambients *cannot be opened* and processes may also communicate with their parent ambient or with a subambient.

The main changes between NBA and BA are the semantics of communication, much less interference-prone, which we will not detail, and the mobility capabilities $\texttt{enter}\langle n, k\rangle$, $\texttt{exit}\langle n, k\rangle$ and *cocapabilities* $\overline{\texttt{enter}}(x, k)$, $\overline{\texttt{exit}}(x, k)$: while $\texttt{enter}\langle n, k\rangle$ instructs the enclosing ambient m to enter its brother ambient n, *using password* k, $\overline{\texttt{exit}}(x, k)$ authorizes this entry and binds x to m.

This password-based mobility allows a measure of control in the mobility: as opposed to ambient names, which should change seldom if ever, access keys may be easily kept secret, canceled and renewed at any time. Additionally, by typing the keys, one may statically determine the type of entering/exiting ambients. Using several keys, well-typed complex communication protocols may be defined. This hierarchical structure of sites and this control of mobility make this calculus comparable to Controlled Ambients and give a natural notion of resources: *sites are resources*, subsites require some space in their parent sites. Controlling resources means ensuring that no site will have more subsites than allowed by some resource policy.

2.1 The calculus of Controlled NBA

The (monadic) calculus of Controlled NBA (CNBA) extends NBA with more control on mobility and resources. Its syntax is identical to that of monadic NBA with replication replaced by recursion, which is easier to control.

$$P, Q \ ::= \mathbf{0} \mid P|Q \mid \mathtt{rec}\, X.P \mid (\nu n : T)P \mid M\,[P] \mid M.P \mid \langle M \rangle^{\eta} P \mid (x)^{\eta} P \mid X$$
$$M, N ::= M.N \mid \mathtt{enter}\langle M, N \rangle \mid \overline{\mathtt{enter}}(x, N) \mid \mathtt{exit}\langle M, N \rangle \mid \overline{\mathtt{exit}}(x, N) \mid k \mid a \mid x$$

For space constraints, we shall present neither structural congruence nor the rules for reduction which we left unchanged. Full definitions are available in a full version of this report [15].

The semantics of `rec` is

Rec $$\mathtt{rec}\, X.P \longrightarrow P\{X \leftarrow \mathtt{rec}\, X.P\}$$

The meaning of capabilities and cocapabilities is slightly modified, along the lines of Controlled Ambients' usage of (co)capabilities. Instead of a two-way synchronization between $\mathtt{enter}\langle n, k \rangle$ and $\overline{\mathtt{enter}}(x, k)$, and of another two-way synchronization between $\mathtt{exit}\langle n, k \rangle$ and $\overline{\mathtt{exit}}(x, k)$, we introduce three-way synchronizations, which allow us to account for all types of movements, with the following rules:

Enter $$a\,[\mathtt{enter}\langle b, k \rangle.P \mid Q] \mid b\,[\overline{\mathtt{enter}}(x, k).R \mid S] \mid \overline{\mathtt{exit}}(x, k).T \longrightarrow b\,[a\,[P \mid Q] \mid R\{x \leftarrow a\} \mid S] \mid T\{x \leftarrow a\}$$

Exit $$a\,\left[b\,[\mathtt{exit}\langle a, k \rangle.P \mid Q] \mid \overline{\mathtt{exit}}(x, k).R \mid S\right] \mid \overline{\mathtt{enter}}(x, k).T \longrightarrow a\,[R\{x \leftarrow b\} \mid S] \mid b\,[P \mid Q] \mid T\{x \leftarrow b\}$$

Do note that we use both $\overline{\mathtt{exit}}(x, k)$ and $\overline{\mathtt{enter}}(x, k)$ in two distinct roles. We could also have introduced new cocapabilities such as $\overline{\mathtt{exit}}(x, k)^{\downarrow}$ and $\overline{\mathtt{enter}}(x, k)^{\uparrow}$, without any important change to the language or the typing system.

Also note that, although we focus on mobility, communications influence resource requirements. In order to make resource control more straightforward, we could have introduced seals-like communication channels. However, this is not required since NBA's communications primitives along with mobility control allow this kind of monitoring on communications.

2.2 An example: distant request, Denial of Service and garbage collection

Let us consider the following system, which we will use as our main running example. A server ($SERVER$) is ready to receive requests from a requester ($FLOODER$), and to send an answer to the site specified in the request. Here, this destination will always be the trashcan ($TRASHCAN$). An additional router process ($ROUTER$) permits answers to be transmitted to the trashcan. $FLOODER$ may produce any number of requests, only one will be answered at a time.

This scenario illustrates simultaneously several important mechanisms: a (simple) Denial of Service attack by $FLOODER$ on $SERVER$, a (simple)

resource-control policy in $SERVER$ and a the trashcan component, which may be used in garbage-collection.

In this version, requests are represented by ambients named r and answers by ambients named a. After being fulfilled, r goes with a to the destination t, the trashcan. Since there are no cocapabilities to allow any ambient to leave t, ambients are effectively isolated from the outside universe and can be considered garbage-collected.

$$
\begin{aligned}
SERVER &= s\,[\mathtt{rec}\,W.\overline{\mathtt{enter}}(\_, k_r).\ (a\,[\overline{\mathtt{enter}}(y, k_r).(M : T_M)^y \mathtt{exit}\langle s, k_a\rangle.M.\mathbf{0}] \\
&\qquad \mid \overline{\mathtt{exit}}(\_, k_r).\overline{\mathtt{exit}}(\_, k_a).W)] \\
&\quad \mid ROUTER \\
ROUTER &= \mathtt{rec}\,Y.(Y \mid \overline{\mathtt{exit}}(\_, k_r) \mid \overline{\mathtt{enter}}(\_, k_a) \mid \overline{\mathtt{exit}}(\_, k_a)) \\
FLOODER &= \mathtt{rec}\,X.(X \mid r\left[\mathtt{enter}\langle s, k_r\rangle.\mathtt{enter}\langle a, k_r\rangle.\,\langle \mathtt{enter}\langle t, k_a\rangle\rangle^{\uparrow}\right]) \\
TRASHCAN &= t\,[\mathtt{rec}\,Z.(Z \mid \overline{\mathtt{enter}}(\_, k_a))] \\
UNIVERSE &= SERVER \mid FLOODER \mid TRASHCAN \mid ROUTER
\end{aligned}
$$

The *resource policy* we wish to ensure is that s may contain at most one subambient named a at any time, which corresponds to the fact that $SERVER$ succeeds in protecting itself against the Denial of Service attack. In other words, the *resource* is the maximal number of a in each ambient at any time. In turn, each a occupies exactly one of these resources.

2.3 Typing resource control

This section is devoted to the presentation of a basic type system for resource control in CNBA. The type system allows one to statically divide available resources between parallel processes, and check that resources will be controlled along movements and communications. *Typing environments are resource-control policies*: each locality has a *capacity* and a *weight*. The capacity is its inner size, i.e. the number of resources the site offers to its subsites, while the weight is its outer size, i.e. the number of resources the site requires from its parent site. In turn, keys are typed according to the type of ambients they may allow to move. Do note that, by changing the affectation of capacities and weights, we may check several different policies on one system.

$$
\begin{array}{lll}
A ::= Amb(s, e)[W] & s \in \mathbb{N} \cup \{\infty\}, e \in \mathbb{N} & K ::= Key(A) \\
C ::= Cap(m, \gamma, A) & m \in \mathbb{Z}, \gamma \text{ environment} & T ::= Proc(t, A)[W]\ t \in \mathbb{N} \cup \{\infty\} \\
W ::= A \mid C \mid K \mid Ssh & &
\end{array}
$$

A ranges over type associated to ambients, K over keys, M over capabilities, T over processes and W over topics of conversation. A type $Amb(s, e)[W]$

is assigned to an ambient named a to denote that the capacity of a is s, that its weight is e and that it may communicate with parent processes on topics of conversation W. A type $Key(A)$ is associated with a key to denote that it allows movement of ambients with type A. A capability or cocapability is typed $Cap(m, \gamma, A)$ when its execution requires m resources, binds variables to types as specified by γ (thus γ must somehow provide parts of a typing environment) and must be executed only in an ambient of type A. If a process P has type $Proc(t, A)[W]$, then P shall not require more than t resources to be executed, P may be executed only in an ambient of type A and whenever P contains occurrences of local communication, its topic of conversation is W.

We write $\Gamma \vdash_{\text{CNBA}} F : G$ for "according to environment Γ, expression F of belonging to the language of Controlled NBA has type G." The typing rules are detailed in figure 1 – rules for local and downwards communication are identical to their counterparts in NBA and were omitted. The expected properties of the type system, namely resource control and subject reduction, are valid. Full proofs may be found in a full version of this report [15].

Theorem 2.1 (Resource control) *If* $\Gamma \vdash_{\text{CNBA}} P : Proc(\_, \_)[\_]$ *and if* a *is an ambient in* P*, then the resource allocation of* a *complies with the resource control policy specified in* Γ*.*

Lemma 2.2 (Resource increase) *If* $\Gamma \vdash_{\text{CNBA}} P : Proc(t, A)[U]$*, then for any* $u \geq t$*,* $\Gamma \vdash_{\text{CNBA}} P : Proc(u, A)[U]$*.*

Theorem 2.3 (Subject reduction) *If* $\Gamma \vdash_{\text{CNBA}} P : T$ *and* $P \longrightarrow Q$*, then* $\Gamma \vdash_{\text{CNBA}} Q : T$*.*

Typing the example

Let us return to the running example. As mentioned earlier, the *resource policy* we wish to ensure is that s may contain at most one subambient named a at any time. In other words, the *resource* is the maximal number of a in each ambient at any time. In turn, each a occupies exactly one of these resources. This is specified by environment Γ:

$$\begin{cases} \Gamma(s) & = Amb(1,0)[Ssh] & \Gamma(t) & = Amb(\infty,0)[Ssh] \\ \Gamma(a) & = T_A = Amb(0,1)[Ssh] & \Gamma(k_a) & = Key(T_A) \\ T_M & = Cap(0,\emptyset,T_A) & \Gamma(r) & = T_R = Amb(0,0)[T_M] \\ \Gamma(k_r) & = Key(T_R) & & \end{cases}$$

From the rules, we deduce $\Gamma \vdash_{\text{CNBA}} UNIVERSE : Proc(\_)[\_]$. In other words, we proved that $UNIVERSE$ complies with our resource policy.

2.4 Discussion

The main entity in both NBA and CNBA, as in CA, is the ambient. If one of these calculi is used to describe a system, ambients and their hierarchy may be used to describe many entities – in our running example, the server, the trashcan, the requests and the answers. In turn, this flexibility gives a natural notion of resource: *resources are ambients*.

Controlling resources is made possible by the structure of capabilities and cocapabilities, which allow a site to *refuse entry dynamically* when necessary and to *react to allocation and deallocation*. In turn, the structure of three-way synchronization through *typed channels*, which is made possible by the use of typed keys in capabilities and cocapabilities, permits the effective typing of mobility, which is the core of this resource control.

3 Controlling Seals

As NBA, the Seal calculus, introduced in [18] and then refined in [5], is a calculus for mobility based on a hierarchy of sites called *seals*. Each seal may contain concurrent processes. Processes may spawn new seals locally, they may communicate either locally, upwards, or downwards. As opposed to NBA, these communications are *channelled*. Processes may also move, copy or rename seals along channels, either locally, upwards or downwards. The semantics of movement is very close to that of CNBA, although mobility in Seals is fully objective (two-ways synchronization between the source site and the destination site) [4], somewhat polyadic, and has different treatment of binding.

As in CNBA, this channel-based mobility allows a great measure of control in the mobility. Similarly, by typing the channels, it is possible to statically determine the type of entering/exiting seals. Using several keys, well-typed complex communication protocols may be defined. Finally, notions of resources and resource control as defined in CA and CNBA are also natural in Seals.

3.1 The calculus of Controlled Seals

The (monadic) calculus of *Controlled Seals* is identical to the latest version of monadic Seals, as introduced in [5], with replication replaced by recursion, easier to control.

[4] Actually, before CA, the first version of Seals had a three-way synchronization mechanism dual to that of CA with fully objective movement. The notion of *portal* disappeared with later versions of Seals.

$$P, Q ::= \mathbf{0} \mid P|Q \mid \mathtt{rec}\, X.P \mid (\nu x : T)P \mid x\,[P] \mid \alpha.Q \mid X$$
$$\alpha \quad ::= \overline{x}^{\eta}(y) \mid x^{\eta}(\lambda y) \mid \overline{x}^{\eta}\{y\} \mid x^{\eta}\{\overrightarrow{y}\} \qquad \eta \quad ::= \quad \uparrow \mid x \mid *$$

The semantics of recursion is identical to CNBA and the other rules are unchanged from Seals, as well as the definition of structural congruence. As in the revisited Seals calculus, the language may be parametrized on an interaction pattern, and movements are caused by the synchronization of $\overline{x}^{\eta_1}\{y\}$ and $x^{\eta_2}\{\overrightarrow{y}\}$, where the actual values of η_1 and η_2 depend on the interaction pattern. More details may be found in [5].

3.2 Running example

Let us return to our running example. It is expressed here in the *Shared* dialect of Controlled Seals, as per [5] but it could have been expressed in the *Local* dialect with minor modifications.

Here, requests are channelled by channel c_r and answers by channel m_a. A special channel named c_g allows an answer to specify its destination. After being fulfilled, requests are simply deleted. The trashcan is a seal t, which effectively deletes any seal which enters it.

$$\begin{aligned}
SERVER &= s\left[\mathtt{rec}\, X.c_r^{\uparrow}(\lambda r).(a\left[\overline{c_g}^{\uparrow}(r)\right] \mid \overline{m_a}^{\uparrow}\{a\}.X)\right] \\
FLOODER &= \mathtt{rec}\, X.(X \mid \overline{c_r}^{s}(t) \mid (\nu x : T_a)m_a^{s}\{x\}.c_g^{x}(\lambda d).\overline{m_a}^{d}\{x\}) \\
TRASHCAN &= t\left[\mathtt{rec}\, X.(X \mid m_a^{\uparrow}\{\}.\mathbf{0})\right] \\
UNIVERSE &= SERVER \mid FLOODER \mid TRASHCAN
\end{aligned}$$

3.3 Typing resource control

The type system for Controlled Seals shares its objectives and many concepts with that of Controlled NBA. Once again, typing environments are resource-control policies and each locality has a *capacity* and a *weight*. Channels are typed according to the type of entities they may carry.

$$\begin{aligned}
S &::= Seal(s, e)\ s \in \mathbb{N} \cup \{\infty\}, e \in \mathbb{N} \qquad C ::= IChan(W) \qquad W ::= S \\
T &::= Proc(t) \quad t \in \mathbb{N} \cup \{\infty\} \qquad\qquad \mid\ MChan(S) \qquad\qquad \mid\ C
\end{aligned}$$

S ranges over types associated to seals, C over channels, T over processes and W over topics of conversation. A type $Seal(s, e)$ is assigned to a seal named n to denote that the capacity of n is s and that its weight is e. A type $IChan(W)$ is associated to channel c to denote that it may carry *information* (i.e. names) of type W. A channel has type $MChan(S)$ whenever it may *move*

seals of type S. If a process P has type $Proc(t)$, then P shall not require more than t resources to be executed.

The typing rules are given in figure 2. Rules VarName, New and Seal are mostly identical to their counterparts in CNBA (see figure 1) and were not included. As in CNBA, we have Resource control and Subject Reduction. Full proofs may be found in a full version of this report [15].

Theorem 3.1 (Resource control) *If* $\Gamma \vdash_{\text{CSl}} P : Proc(\_)$ *and if* a *is a seal in* P*, then the resource allocation of* a *complies with the resource control policy specified in* Γ*.*

Lemma 3.2 (Resource increase) *If* $\Gamma \vdash_{\text{CSl}} P : Proc(t)$*, then for any* $u \geq t$*,* $\Gamma \vdash_{\text{CSl}} P : Proc(u)$*.*

Theorem 3.3 (Subject reduction) *If* $\Gamma \vdash_{\text{CSl}} P : T$ *and* $P \longrightarrow Q$*, then* $\Gamma \vdash_{\text{CSl}} Q : T$*.*

Typing the example

Let us return to the running example. As earlier, the *resource policy* we wish to enforce is that s may contain at most one a at any time. In other words, each a occupies exactly one resource and s offers only one resource, as per environment Γ:

$$\begin{cases} \Gamma(s) = Seal(1,0) & \Gamma(c_r) = IChan(Seal(0,0)) \\ \Gamma(a) = Seal(0,1) & \Gamma(m_a) = MChan(Seal(0,1)) \\ \Gamma(t) = Seal(0,0) & \Gamma(c_g) = IChan(Seal(0,0)) \end{cases}$$

We then deduce $\Gamma \vdash_{\text{CSl}} UNIVERSE : Proc(\_)$. In other words, we proved that $UNIVERSE$ complies with our resource policy.

3.4 Discussion

Controlling resources in Controlled Seals is very similar to the same task in CNBA, only easier. As in CNBA, Seals offer a natural notion of resource: *resource are seals*. Controlling is made possible by the structure of movements, which allow a site to *refuse entry dynamically* and to react to allocation and deallocation, and the typing of mobility is made possible by the structure of objective movements through *typed channels*.

4 Controlling Nomadic π

The Nomadic π calculus (Nπ), introduced in [17] is a calculus of *mobile agents*. A system is represented by a set of concurrent *located processes*, all of them located in *agents*, a set of uniquely named agents, all of them located in *sites* and a set of uniquely named, immutable sites. There is only one level of

sites. Processes may spawn new agents locally, *migrate* their enclosing agent to another site and communicate, either locally or to possibly distant agents. As in Seals, communications are *channelled*. Migrations, however, are fully subjective.

As opposed to Seals or CNBA, this free-for-all migrations prevents any control of mobility, not to mention resources. Similarly, the absence of sites hierarchy prevents notions of resources from being directly imported from CA, CNBA or Controlled Seals. Fortunately, as in Controlled Seals, keys on communication allow well-typed complex communication protocols to be defined. Additionally, this two-level structure of agents in sites may be translated into another, different yet comparable, notion of resources: *sites offer resources, agents use them*. Controlling resources means ensuring that no site will have more agents than allowed by some resource policy.

4.1 The calculus of Controlled Nomadic π

The (monadic) calculus of Controlled Nomadic π (CNπ) extends synchronous Nπ with control over migration and agent creation. The syntax we use is quite different from the classical syntax of Nπ, as presented in [17]. For example, if agent a is running the set of processes $\{P_1, \ldots, P_n\}$, the original syntax writes $@_a P_1 \mid \cdots \mid @_a P_n$, whereas we prefer $a[P_1 \mid \cdots \mid P_n]$. The reason of this change is that our syntax is much closer to that of the other calculi we describe in this work, and easier to type for resource-control.

Universes — *Sites contents*

$U, V ::= \epsilon \mid l[A] \mid U; V$ — $A, B ::= \emptyset \mid a\{P\} \mid A, B$

Agents contents

$P, Q ::=$ (π-like constructs)

$() \mid P|Q \mid \texttt{new}\ c : T\ P \mid c!v.P \mid c?p \to P$

(routed communication)

$\mid \langle a \rangle\, c!v.P \mid \langle a@s \rangle\, c!v.P \mid \texttt{iflocal}\ \langle a \rangle\, c!v\ \texttt{then}\ P\ \texttt{else}\ Q$

(agent spawning and migration)

$\mid \texttt{agent}\ p\ \texttt{as}\ b = P\ \texttt{in}\ Q \mid \texttt{migrate}\ a\ \texttt{to}\ l\ P \mid$
$\overline{\texttt{enter}}\ a\ P \mid \overline{\texttt{exit}}\ a\ P \mid \overline{\texttt{create}}\ k\ P$

(recursion and exceptions)

$\mid \texttt{rec}\, X.P \mid X \mid \texttt{try}\ c?p = P\ \texttt{timeout}\ n \to Q$

In CNπ, we add cocapabilities to agent creation and migration. Instead of an uncontrolled creation by $\texttt{agent}\ a = P\ in\ Q$, there is now a two-way synchronization between $\texttt{agent}\ k\ \texttt{as}\ a = P\ \texttt{in}\ Q$ and $\overline{\texttt{create}}\ k$, where k is a

creation key. Similarly, instead of an uncontrolled migration by `migrate to` s, there is now a three-way synchronization between `migrate` a `to` s , $\overline{\texttt{exit}}\ a$ and $\overline{\texttt{enter}}\ a$, where a is the name of the migrating agent. The operational semantics rules that are specific to CNπ are the following:

$$\text{R-Create} \qquad a\,\{\texttt{agent}\ k\ \texttt{as}\ b = P\ \texttt{in}\ Q \mid R\}\,, c\,\{\overline{\texttt{create}}\ k\ S \mid T\} \longrightarrow a\,\{Q\{x \leftarrow f\} \mid R\}\,, c\,\{S \mid T\}\,, b\,\{P\{x \leftarrow f\}\} \quad (f \text{ fresh})$$

$$\text{R-Migrate} \qquad s[a\,\{\texttt{migrate}\ a\ \texttt{to}\ t\ P\}\,; b\,\{\overline{\texttt{exit}}\ a\ R \mid S\}\,; A],\ t[c\,\{\overline{\texttt{enter}}\ a\ T \mid U\}\,; B] \longrightarrow s[b\,\{R \mid S\}\,; A], t[c\,\{T \mid U\}\,; a\,\{P\}\,; B]$$

See [15] for a full presentation of the operational semantics.

An important difference between (C)Nπ and other calculi is that there is no specific "parent" thread to implement the control. The authorization to exit or create is thus executed from within a brother agent residing at the same site as the moving agent, while the authorization to enter is executed from within any agent residing at the target site. This agent may be either an immobile controller, but it does not have to. We thus need to control resource authorizations brought by migrating agents.

Do note that we impose the migrating process to be *single threaded*, which represents a restriction w.r.t. [17]'s original calculus, where an agent of the form $a\,\{\texttt{migrate to}\ t\ P \mid Q\}$ can perform a migration. This is due to the fact that processes such as $Q = \overline{\texttt{exit}}\ b\ Q'$ actually require *more* resources after the execution of $\overline{\texttt{exit}}\ b$ than before. Without introducing this restriction, we were not able to enforce any sensible control in presence of migration. The induced programming discipline is that, as opposed to MA, NBA or Seals, an agent shall *join* its concurrent activities before migration.

Also note that, as opposed to Nomadic π, distant communication is a required low-level primitive, since it must be used to transmit the name of migrating agents.

4.2 Running example

To handle our running example in CNπ, we use channel c_r to carry requests while answers migrate to the destination. A special key k_a is used to allow the spawning of new answers while a special channel m_a allows an answer to specify its destination and a special channel m_t allows the trashcan to receive the name of incoming agents.

$$
\begin{aligned}
SERVER &= s\{\texttt{rec}\, X.c_r?r \rightarrow \texttt{agent}\ k_a\ \texttt{as}\ a = (\langle b\rangle\, m_a!a.\ \texttt{in} \\
&\qquad \langle o@r\rangle\, m_t!a.\texttt{migrate}\ a\ \texttt{to}\ r\ ())X\} \\
ROUTER &= b\left\{\texttt{rec}\, X.\overline{\texttt{create}}\ k_a\ .m_a?a \rightarrow \overline{\texttt{exit}}\ a\ X\right\} \\
FLOODER &= c\left\{\texttt{rec}\, X.(X \mid \langle s@z\rangle\, c_r!t)\right\} \\
TRASHCAN &= o\left\{\texttt{rec}\, X.(X \mid m_t?a \rightarrow \overline{\texttt{enter}}\ a\ )\right\} \\
UNIVERSE &= z[SERVER \mid ROUTER] \mid f[FLOODER] \mid t[TRASHCAN]
\end{aligned}
$$

Do note that, as opposed our example given in other calculi, agents sent to the trashcan may possibly come back from t if helped by an agent also present in t and another agent present in the destination site.

4.3 Type system

The grammar for types is defined as follows:

$$
\begin{aligned}
A &::= Ag(g,e) \qquad g,e \in \mathbb{N} \qquad R ::= Run(e)\ \ e \in \mathbb{N} \cup \{\infty\} \\
&\mid\ Site(s) \qquad s \in \mathbb{N} \qquad T ::= Proc(t)\ t \in \mathbb{N} \cup \{\infty\} \\
C &::= Channel(A)
\end{aligned}
$$

A ranges over types associated to agent and site names, C over types of channel names, R over types of agents and T over types of processes (that are parts of agents). A type $Ag(g,e)$ is assigned to an agent name a to denote the fact that a requires e resources in the site it's residing in, and that at the moment of creation (instruction $\texttt{agent}\ k\ \texttt{as}\ a = P\ \texttt{in}\ Q$) or migration (instruction $\texttt{migrate}\ a\ \texttt{to}\ t$), agent a contains a process whose effect is bounded by g. A process P has type $Proc(t)$, where t is a bound on the *effect* of P in terms of resource consumption.

Typing rules are presented on figure 3. We omitted rules Nil, New, Rec, Par, VarProc, VarName, Receive-Info and Emit-Info, identical to their counterparts in Seals.

Do note that, in rule Enter, we add the effect t of the process to the weight e of its containing agent, which shows an important difference w.r.t. other controlled calculi. This is due to the fact that, in other calculi, the typing system may find one controller process for each location, whereas in CNπ, this process is distributed between the agents.

Resource Control and Subject Reduction are different, from the other calculi, since they apply only at the level of *Universe*s and since fresh names are generated at agent creation.

Theorem 4.1 (Resource control) *If $\Gamma \vdash_{\text{CN}\pi} A$ and if a is a site in A, then the resource allocation of a complies with the resource control policy specified in Γ.*

Theorem 4.2 (Subject Reduction) *If* $\Gamma \vdash_{\mathrm{CN}\pi} U$ *and* $U \longrightarrow V$ *then either* $\Gamma \vdash_{\mathrm{CN}\pi} V$ *or, in the case where* $U \longrightarrow V$ *corresponds to the creation of an agent* f *with key* k, $\Gamma, f : \Gamma(k) \vdash_{\mathrm{CN}\pi} V$.

The extension of the environment during Subject Reduction may seem somewhat strange. This is due to the creation of *fresh names* for new agents. This phenomenon and the statement of Subject Reduction are reminiscent of simple types in the π-calculus under a LTS-based operational semantics, in the case where a name is extruted to the context, as in [11]. Full proofs may be found in a full version of this report [15].

Typing the example

The resource policy we wish to ensure is that site z will never contain more than one agent created with k_a. In other words, agents created with k_a occupy exactly one resource and z offers exactly one resource.

$$\begin{cases} \Gamma(z) & = Site(1) & \Gamma(f) & = Site(0) \\ \Gamma(t) = T_t & = Site(\infty) & \Gamma(k_a) = T_a & = Ag(1,0) \\ \Gamma(m_t) = \Gamma(m_a) & = Channel(T_a) & \Gamma(c_r) & = Channel(T_t) \\ \Gamma(s) = \Gamma(b) = \Gamma(c) = \Gamma(o) & = Ag(0,0) & & \end{cases}$$

From the rules, we deduce $\Gamma \vdash_{\mathrm{CN}\pi} UNIVERSE$. In other words, the system complies with the resource-control policy. Do note, however, that the example is written with one immobile controller agent per site. If we had, for instance, distributed permissions between two router agents, one with $\overline{\texttt{create}}$ and the other one with $\overline{\texttt{exit}}$, we could not have typed the system.

4.4 Discussion

Though comparable, resource-control in CNπ and in CA, CNBA or CSeals are not identical tasks. In CNπ, *resources are offered by sites and used by agents* and only combinations of sites and agents permit the description of entities such as the server, the trashcan and the answers.

Once again, controlling resources is made possible by the structure of (co)capabilities, which gives the possibility to *refuse entry dynamically* when necessary and to *react to allocation and deallocation*. The three-way synchronization through *named authorizations*, close to those of CA, allow the effective typing of mobility, but its semantics make it stricter and more limited than the *channeled* mobility of CNBA or CSeals. The key-based creation of new agents, on the other hand, is close to other key-based concepts from CNBA or CSeals.

Presumably, if one were to work on CNπ to enhance Nomadic Pict, a good first step would be to add keys to migration as well as agent creation, hence

making the calculus slightly more complicated and the writing of protocols and programs much easier. Early experiments seem to indicate that the restriction on migration could also be alleviated, at the cost of a slightly more complicated type system.

5 Controlling Kells

The Kell calculus, introduced in [14] is a fully asynchronous, explicitly higher-order formalism for distributed components. Kells represent an evolution of the M-calculus [13], which in turn builds on the distributed Join Calculus and borrows ideas from Mobile Ambients. As in MA or Seals, a system is represented by a hierarchy of kells. Each site may be either active (i.e. a site) or passive (i.e. a communication channel) and may contain concurrent processes. Processes may spawn new kells and communicate either locally, upwards or downwards through the use of higher-order *triggers*, similar to Join Patterns in the Join Calculus [8]. Communications are fully asynchronous and controlled by the target while mobility is a special case of communication. The actual language for Kell patterns is left as a parameter of the calculus.

As opposed to Nπ, this lack of synchronization does not really hamper mobility control, since the target is in charge of migration. Adding the hierarchical structure of kells, we find a natural notion of resources: *Kells are resources*, subkells, whether they are active or passive, require some resources from their parent kell. Unfortunately, as in Nπ, the lack of synchronization prevents reactivity: in the general case, a site will not be aware of a subsite leaving. Hence, it will not be able to allow new entries as a consequence of resources being released.

5.1 The calculus of Controlled Kells

The calculus of Controlled Kells (CK) extends Kells with reactivity to movements. As in Kells, the unit of computation is a *kell*: $a\langle P\rangle$ is a passive (frozen) kell [5] while $a[P]$ is an active (site-like) kell, with process P running at address a . At some points, we shall write $a * P$ (or $a_i *_i P_i$) for a kell, either active or passive, named a and containing process P, and $\prod_i P_i$ for parallel composition of processes.

$$P, Q \quad ::= \quad 0 \mid P \,|\, Q \mid a \mid x \mid \xi \rhd P \mid \nu a.P \mid a \lhd P \mid a[P] \mid a\langle P\rangle \,.$$

The semantics of triggers is slightly modified to allow reactivity, in a way comparable to Seals' two-way synchronization semantics. As in the regular Kell calculus, $\xi \rhd P$ is a *Kell-trigger*, comparable to a Join-pattern with the added possibilities of matching processes and crossing kells boundaries. In Kells, any expression matched by ξ is consumed from its source site and may

[5] We are using recent notations for Kells – in [14], passive kells are written $a \circ P$ while active kells are written $a \bullet P$.

be reinjected in P through the use of *name and processes markers*. However, while the uncontrolled Kell controls mobility from the target site (i.e. the trigger) only, we add to Controlled Kells the *Kell-release* $a \lhd Q$, which *allows* any trigger to match a and specifies that Q shall be released whenever a is matched (and consumed). This Kell-release is similar to cocapability $\overline{\texttt{exit}}\ a$ in CNπ. The formal semantics of Triggers and Releases is given in rules K.Red$\downarrow$ (presented below) and K.Red$\updownarrow$.

As mentioned earlier, the pattern language used is unspecified. Actually, Kell is more a family of calculi parametrized by its pattern language and in [14], three different pattern languages are presented: *Simple patterns*, *Complex patterns* and *μdK patterns*. Despite this lack of specification, we do need to refer to the names in a pattern that are involved in a reduction step. This is formalized using the notion of *requested names,* defined as follow:

Definition 5.1 [Requested names] For a given pattern ξ, we define $\mathsf{rn}(\xi)$, the *multiset* of *requested names* of ξ, as follows: let $\mathcal{P}$ be the set of positions corresponding to occurrences of free names in ξ, and let $\mathcal{P}_{top} \subseteq \mathcal{P}$ be the set of positions that are not underneath an element of $\mathcal{P}$. $\mathsf{rn}(\xi)$ is obtained as the multiset containing the names appearing at the positions defined in $\mathcal{P}_{top}$.

If ξ is defined by $\xi = (a)\langle(b)\langle(x)\rangle \mid (b')\langle(y)\rangle\rangle \mid (a')[(z)] \mid m[t] \mid (a')[(c)]$, then we have $\mathsf{rn}(\xi) = \{a, a', a', m\}$.

The following rule details the semantics of local and upwards communication (we use $\sqcup$ for multiset union):

$$\text{K.Red}\downarrow\ \frac{\mathsf{rn}(\xi) = L \sqcup D_1 \sqcup \ldots \sqcup D_n \qquad \xi\theta \equiv \prod_{i\in L} b_i *_i A_i \mid \prod_{1\le k\le n} \prod_{i\in D_i} b_{k,i} *_{k,i} A_{k,i}}{\xi \rhd P \mid \prod_{i\in L}(b_i *_i A_i \mid b_i \lhd Z_i) \mid \prod_{1\le k\le n} a_k[R_k \mid \prod_{i\in D_k}(b_{k,i} *_{k,i} A_{k,i} \mid b_{k,i} \lhd Z_{k,i})] \longrightarrow P\theta \mid \prod_{i\in L} Z_i \mid \prod_{1\le k\le n} a_k[R_k \mid \prod_{i\in D_k} Z_{k,i}]}$$

As its counterpart in uncontrolled Kell-calculus, because of the generic treatment of (possibly very rich) patterns, the rule looks quite complicated. As specified by $\xi\theta \equiv \ldots$, it may only be invoked whenever ξ matches some process, through substitution θ. We first decompose the required names of ξ into those corresponding to matching kells that will be found locally (L), and to matching kells that will be found in subkells located at $a_1, \ldots, a_n$ (multisets $D_1, \ldots, D_n$). For each of the matching requested names $b_\_$, we note the corresponding nature of the kell (active or passive) $*_\_$, the corresponding kell body $A_\_$ and the corresponding Kell-release $b_\_ \lhd Z_\_$. Using these notations, the source process of the reduction is the parallel composition of the involved Kell-trigger and the reacting agents, both locally and in relevant subkells. The target process is the parallel composition of the application of substitution θ and of the agents after reaction, where matching kells have been removed and releases have been executed.

Another rule, K.Red$\updownarrow$, enriches K.Red$\downarrow$ to take into account the possibility for patterns to be triggered by components located in the parent kell. We do not detail this rule, since it is similar, only more complex. Structural congruence and other rules are defined in a standard way (see [14]). Do note that the semantics of ν is defined as a rule and not within structural congruence.

Another point of interest is that the addition of Kell-releases may help with the problem of interferences between triggers. This problem, pointed out in [14], hampers distributed implementations of Kells, as some difficult circumstances require some form of distributed consensus across the network. In Controlled Kells, since no kell may be selected unless a Kell-release allows this selection, well-programmed components shall present few or none of these unwanted interferences.

5.2 Running example

Let us return to our running example. It is expressed here in the *Simple pattern*-language(see [14]) but it could have been expressed using other pattern-languages.

Let us first introduce the following auxiliary macros:

$$\begin{aligned} COREC(X,P,t) &= (\nu u : T_{u,X})(t\langle (x : T_{x,X})\rangle \triangleright \\ &\qquad (u\langle t\langle x\rangle \mid x \mid t \triangleleft \mathbf{0}\rangle \mid u \triangleleft \mathbf{0}) \mid u\langle (X : T_{X,X})\rangle \triangleright P) \\ REC(X,P) &= (\nu t : T_{t,X})(t\langle COREC(X,P,t)\rangle \mid COREC(X,P,T) \mid t \triangleleft \mathbf{0}) \\ &\quad (x, u, t \text{ free in } P) \end{aligned}$$

Due to space limitations, we shall not discuss in detail these macros. Once again, more details may be found in a full version of this report [15]. Type annotations shall be discussed later. Note, however, that the existence of these macros illustrates the power of Controlled Kells.

Here, requests are represented by passive kells named r and answers by active kells named a. As in Seals, after being fulfilled, requests are simply deleted. Special passive kells named *exit* and *enter* permit routing of the answer from the server to its destination. The trashcan is a kell t, which effectively deletes any kell which enters it.

$$\begin{aligned} SERVER &= s\,[REC(X, (r\langle (d : T_d)\rangle) \triangleright (exit\langle carry\langle a[]\rangle \mid to\langle d\rangle\rangle \mid exit \triangleleft X))] \\ FLOODER &= REC(Y, Y \mid r\langle t\rangle) \\ ROUTER &= REC(Z, Z \mid (exit\langle carry\langle (c : T_c)\rangle \mid to\langle (d : T_d)\rangle\rangle)\triangleright \\ &\qquad enter\langle carry\langle c\rangle \mid to\langle d\rangle\rangle) \\ TRASHCAN &= t\,[REC(W, W \mid (enter\langle carry\langle (c : T_c)\rangle \mid to\langle t\rangle\rangle) \triangleright \mathbf{0})] \\ UNIVERSE &= SERVER \mid FLOODER \mid ROUTER \mid TRASHCAN \end{aligned}$$

5.3 Typing resource control

The type system for Controlled Kells is reminiscent to that of Controlled Seals. The first main difference is that kells, *whether they are active or passive*, have a capacity and a weight. The second main difference is that we cannot type patterns since we do not know the language they are written in. We must rely on some types provided by some oracle – or by annotations.

$$K ::= Kell(s, e)\ s \in \mathbb{N} \cup \{\infty\}, e \in \mathbb{N} \qquad T ::= Proc(t)\ t \in \mathbb{N}$$
$$H ::= Pat(\gamma) \qquad \gamma \text{ environment}$$

K ranges over types associated to Kells, T over processes and H over patterns. A type $Kell(s, e)$ is assigned to a kell named n to denote that the capacity of n is s and that its weight is e, whether n is passive or active. As usual, if a process P has type $Proc(t)$, then P shall not require more than t resources to be executed. As for patterns types, ξ has type $Pat(\gamma)$ if all process and name markers of ξ may only be bound to expressions with types as specified in γ. As mentioned earlier, the type of patterns must be given by some meta-level oracle.

Based on this oracle, the typing rules for resource control in Kells are given on figure 4. Most rules were identical to their counterparts in Seals (given on figure 2) and were not included.

Due to the oracle nature of pattern types, we must introduce a notion of typed reduction to state our Subject Reduction property.

Definition 5.2 [Typed reduction] If $\Gamma \vdash_{\text{CK}} P : T$, then $P \longrightarrow Q$ is a step of typed reduction for Γ, noted $P \longrightarrow_\Gamma Q$, if, for any application of rules K.Red$\downarrow$ or K.Red$\updownarrow$, noting θ the substitution involved, and for any name marker or process marker (m) in ξ, we have $\Gamma \vdash_{\text{CK}} \theta(m) : \gamma(m)$.

For unspecified pattern languages, this property is very strong. However, according to our experiments in languages such as *Simple-*, *Complex-* or μdK-Patterns, Typed Reduction seems to translate to a simple sufficient condition on patterns at the cost of a slightly more complex type system. Namely, provided that the system is properly typed and that all markers in each pattern are labelled by an enclosing kell, it seems that reduction is always typed. In other words, it seems sufficient that, as in ML languages, *pattern matching must be applied to constructors.* Note that this condition is always satisfied in μdK.

We then have the following properties:

Theorem 5.3 (Resource control) *If $\Gamma \vdash_{\text{CK}} P : Proc(\_)$ and if a is a kell in P, then the resource allocation of a complies with the resource control policy specified in Γ.*

Lemma 5.4 (Resource increase) *If $\Gamma \vdash_{\text{CK}} P : Proc(t)$ then $\forall u \geq t, \Gamma \vdash_{\text{CK}} P :$*

$Proc(u)$.

Theorem 5.5 (Subject reduction) *If* $\Gamma \vdash_{\mathrm{CK}} P : T$ *and* $P \longrightarrow_{\Gamma} Q$ *then either* $\Gamma \vdash_{\mathrm{CK}} Q : T$ *or, in the case where* $P \longrightarrow_{\Gamma} Q$ *corresponds to the reduction of* $(\nu m : U)R$ *then there exists some* n *such that* $\Gamma, n : U \vdash_{\mathrm{CN}\pi} Q : T$.

This extension of Γ, similar to that of Nomadic Pict, takes into account the specific treatment of ν in Kells.

Typing the example

For readability reasons, since the type of a pattern is nothing else than the type of all its markers, instead of writing the full type, we shall write type annotations on the markers. For example, rather than writing that pattern $(x)|(y)$ has type $Pat(\{x : T_x, y : T_y\})$ with some values instead of T_x and T_y, we shall rewrite the pattern as $(x : T_x)|(y : T_y)$ and give the values of T_x and T_y later.

We wish to prove that s may contain at most one a at any time. This will be captured by having each a occupy exactly one resource and s offer only one resource. Since a is actually nested within *carry* and carry within *exit*, we will also write that both *carry* and *exit* offer one resource and occupy one resource. This, as well as the types of patterns, is specified in the following environment:

$$\begin{cases} \Gamma(enter) = \Gamma(s) & = Kell(1,0) \\ T_a = T_c = \Gamma(a) & = Kell(0,1) \\ \Gamma(r) & = Kell(0,0) \\ \Gamma(exit) = \Gamma(carry) & = Kell(1,1) \\ T_d = \Gamma(t) = \Gamma(to) & = Kell(0,0) \\ T_{t,X} = T_{u,X} & = Kell(1,0) \\ T_{X,X} = T_{x,X} & = Proc(1) \\ T_{x,Y} = T_{x,Z} = T_{x,W} & = Proc(0) \\ T_{Y,Y} = T_{Z,Z} = T_{W,W} & = Proc(0) \\ T_{t,Y} = T_{t,Z} = T_{t,W} = T_{u,Y} = T_{u,Z} = T_{u,W} & = Kell(0,0) \end{cases}$$

From Γ and the typing rules, we may type $UNIVERSE$. In other words, we proved that $UNIVERSE$ complies with our resource policy. Also notice that the typing of our macro REC is actually very similar to the typing of `rec` in other resource-controlled calculi.

5.4 Discussion

Controlling resources in Controlled Kells is rather similar to the same task in Controlled Seals, with the added difficulty of the types being given by an oracle and the added necessity of typed reduction. As in CSeals, *Resources are*

Kells and controlling resources is made possible by the structure of movement along *typed channels*, which allows a site to *limit entry*, and by synchronization, which allows a site to *notice that resources have been deallocated*. Unfortunately, the unspecified pattern language does not allow direct typing of mobility and requires that some types to be provided by an oracle in addition to a property of typed reduction. However, experiments with test pattern-languages seem to show that simple properties of "cleanness" of patterns and type annotations are sufficient to replace the oracle types and ensure typed reduction.

6 Conclusion

Discussion

The languages of Controlled Ambients, Controlled NBA, Controlled Seals, Controlled Nomadic π and Controlled Kells have been introduced to allow the analysis and control of resources in mobile, concurrent and distributed settings, through an accurate programming of movements and synchronizations. We have also enhanced each of these calculi with a type system for static resource control. We have illustrated each calculus and type system on an example which could actually be statically controlled without any rewriting necessary beyond straightforward adaptation to the language.

Methods developed in [16] were applied successfully to all of these formalisms. In CA, CNBA, CSeals and CK, the primitive mobile entity is the site and sites can be nested, thus creating hierarchies. In CNπ, the primitive mobile entity is the agent, and agents can be nested within immobile sites. In each case, the mobile entity can be spawned and moved or destroyed, just as resources may be allocated and released. These entities are thus *close to our definition of resource*, "an entity which may at will be acquired, used, then released." In all of these calculi, *mechanisms permit to control mobility*: both mobility channels or keys (CNBA, CSeals) and strict entity-specific authorizations (CA, CNπ, CK) result in *rich type information about mobility* with synchronization between the source space and the target space, generally in addition to subjective moves, and give the possibility to *react to movements* and *refuse entry dynamically*.

Future work and related works

We have also thought of adapting our techniques to other calculi, such as the π-calculus or $D\pi$. The task is complex, since the structure of these calculi makes the definition of resources very different from what we have been working in in other calculi. However, we have been making some progress by using a new kind of cocapability and taking advantace of structural equivalence and garbage-collection.

Amongst extensions of this work, we are also working on specifying sets of conditions on Kell-patterns which would be appropriate for Controlled Kell.

We have also begun to try and understand what is required to control resources whenever a calculus contains replication, without resorting to replacing it by recursion. Another aspect we are working on is that of generalizing our notion of resources to encompass some (possibly complex) aspects of security as well as some forms of non-renewable resources such as time or money.

Another point we found to be interesting while adapting our examples to each calculus is that the adaptation of an expression from a resource-uncontrolled formalism to its resource-controlled counterpart was akin to some steps of a compilation process, as could be implemented in resource-conscious compilers for concurrent languages. This compilation-like process is also close to [12] which adds to Mobile Ambients many primitives directly related to resources, as would exist in a compiler's target language. Note that these primitives could probably be adapted to other languages as well and that they permit to control a modified form of replication, but that they seem less flexible than our additions when it comes to specifying resource policies.

Boxed Ambients with Guardians, as introduced in [7], are another attempt at controlling resources in Boxed Ambients. Guardians are process-like mobility controllers featuring cocapabilities-like prefixes with synchronizations similar to those of CNBA. However, the actual nature of controlling entities in CNBA and in Guardians are orthogonal: where CNBA's control is featured through regular processes, control in Guardians involves specific entities with their own semantics – the *Guardians*. However, to the best of our knowledge, no type system has been developped to account for resources statically in Guardians.

Some other form of accounting has also been introduced for Mobile Ambients in [6], but with very different objectives: to isolate finite-control fragment of the calculus with decidable model checking w.r.t. Ambient Logics.

Finally, the problem of bounding both time and space resources is approached in [1] in the case of lambda-calculus.

References

[1] R. M. Amadio. Max-plus quasi-interpretations. Research report 10-2002, LIF, Marseille, France, Dec 2002. http://www.lim.univ-mrs.fr/Rapports/10-2002-Amadio.html.

[2] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proc. TACS 2001*, LNCS 2215, pages 38–63. Springer Verlag, 2001.

[3] M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication Interference in Mobile Boxed Ambients. In *Proc. of FST-TCS'02*, volume 2556 of *LNCS*. Springer Verlag, 2002.

[4] L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference,*

FOSSACS '98, volume 1378, pages 140–155. Springer-Verlag, Berlin Germany, 1998.

[5] G. Castagna and F. Zappa. The seal calculus revisited. In *Proc. of 22th FSTTCS*, pages 85–96, 2002.

[6] W. Charatonik, A. D. Gordon, and J.-M. Talbot. Finite-control mobile ambients. In *Proc. of ESOP'02*, volume 2305 of *LNCS*, pages 295–313, 2002.

[7] G. Ferrari, E. Moggi, and R. Pugliese. Guardians for ambient-based monitoring. In V. Sassone, editor, *F-WAN: Foundations of Wide Area Network Computing*, number 66 in ENTCS. Elsevier Science, 2002.

[8] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421. Springer-Verlag, 1996.

[9] C. Fournet, J.-J. Levy, and A. Schmitt. An asynchronous, distributed implementation of mobile ambients. In *IFIP TCS*, pages 348–364, 2000.

[10] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 378–390, New York, NY, 1998.

[11] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[12] V. Sassone, F. Barbanera, M. Bugliesi, and M. Dezani. Capacity bounded computational ambients (not published yet). 2003.

[13] A. Schmitt and J.-B. Stefani. The M calculus: A Higher-Order Distributed Process Calculus. In *Proc. of POPL'03*. ACM Press, 2003.

[14] J.-B. Stefani. A calculus of Higher-Order Distributed Components. Technical Report 4692, INRIA, 2003.

[15] D. Teller. Formalisms for mobile resource control. Technical Report (draft), LIP, 2003. `http://www.ens-lyon.fr/~dtelle/recherche/Publications/mobileresourcesrr.ps.gz`.

[16] D. Teller, P. Zimmer, and D. Hirschkoff. Using Ambients to Control Resources. In *Proc. of CONCUR'02*, volume 2421 of *LNCS*. Springer Verlag, 2002.

[17] A. Unyapoth. *Nomadic Pi Calculi: Expressing and Verifying Infrastructure for Mobile Computation*. PhD thesis, Computer Laboratory, University of Cambridge, june 2001.

[18] J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Lecture Notes in Computer Science*, volume 1686 of *LNCS*. Springer Verlag, 1998.

$$\textsc{Nil}\ \Gamma \vdash_{\text{CNBA}} 0 : Proc(\_, \_)[W] \qquad \textsc{VarName}\ \frac{\Gamma(n) = W}{\Gamma \vdash_{\text{CNBA}} n : W}$$

$$\textsc{VarProc}\ \frac{\Gamma(X) = Proc(u, A)[W]}{\Gamma \vdash_{\text{CNBA}} X : Proc(t, A)[W]}\ u \geq t \qquad \textsc{New}\ \frac{\Gamma \vdash_{\text{CNBA}} (\nu x : T)P : U}{\Gamma, x : T \vdash_{\text{CNBA}} P : U}$$

$$\textsc{Par}\ \frac{\Gamma \vdash_{\text{CNBA}} P : Proc(t_P, l)[L] \qquad \Gamma \vdash_{\text{CNBA}} Q : Proc(t_Q, l)[L]}{\Gamma \vdash_{\text{CNBA}} P \mid Q : Proc(t_P + t_Q, l)[L]}$$

$$\textsc{Amb}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CNBA}} M : A \\ P : Proc(t, A)[\_] \qquad \Gamma \vdash_{\text{CNBA}} A = Amb(s, e)[\_]\end{array}}{\Gamma \vdash_{\text{CNBA}} M\,[P] : Proc(f, B)[W]}\ t \leq s, f \geq e$$

$$\textsc{Path}\ \frac{\Gamma \vdash_{\text{CNBA}} M : Cap(m, \gamma, l) \qquad \Gamma \vdash_{\text{CNBA}} N : Cap(n, \delta, l)}{\Gamma \vdash_{\text{CNBA}} M.N : Cap(m + n, (\gamma, \delta), l)}$$

$$\textsc{Prefix}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CNBA}} M : Cap(m, \delta, l) \\ \Gamma, \delta \vdash_{\text{CNBA}} P : Proc(t, l)[L]\end{array}}{\Gamma \vdash_{\text{CNBA}} M.P : Proc(u, l)[L]}\ u \geq m + t, u \geq 0$$

$$\textsc{Enter}\ \frac{\Gamma \vdash_{\text{CNBA}} k : Key(A) \qquad \Gamma \vdash_{\text{CNBA}} M : Amb(\_, \_)[\_] \qquad A = Amb(\_, \_,)[\_]}{\Gamma \vdash_{\text{CNBA}} \texttt{enter}\langle M, k\rangle : Cap(0, \emptyset, A)}$$

$$\textsc{Coenter}\ \frac{\Gamma \vdash_{\text{CNBA}} k : Key(A) \qquad A = Amb(\_, e)[\_] \qquad \delta = \{x : A\}}{\Gamma \vdash_{\text{CNBA}} \overline{\texttt{enter}}(x, k) : Cap(e, \delta, \_)}$$

$$\textsc{Exit}\ \frac{\Gamma \vdash_{\text{CNBA}} k : Key(A) \qquad \Gamma \vdash_{\text{CNBA}} M : Amb(\_, \_)[\_]}{\Gamma \vdash_{\text{CNBA}} \texttt{exit}\langle M, k\rangle : Cap(0, \emptyset, A)}$$

$$\textsc{Coexit}\ \frac{\Gamma \vdash_{\text{CNBA}} k : Key(A) \qquad A = Amb(s, e)[T] \qquad \delta = \{x : A\}}{\Gamma \vdash_{\text{CNBA}} \overline{\texttt{exit}}(x, k) : Cap(-e, \delta, \_)}$$

$$\textsc{Snd-Up}\ \frac{\Gamma \vdash_{\text{CNBA}} M : L \qquad \Gamma \vdash_{\text{CNBA}} P : T \qquad T = Proc(\_, Amb(\_, \_)[L])[\_]}{\Gamma \vdash_{\text{CNBA}} \langle M\rangle^{\uparrow}.P : T}$$

$$\textsc{Rcv-Up}\ \frac{\Gamma, x : U \vdash_{\text{CNBA}} P : T \qquad M : Amb(\_, \_)[U]}{\Gamma \vdash_{\text{CNBA}} (x : U)^{M} P : T}$$

$$\textsc{Rec}\ \frac{\Gamma, X : Proc(t, A)[U] \vdash_{\text{CNBA}} P : Proc(t, A)[U]}{\Gamma \vdash_{\text{CNBA}} \texttt{rec}\, X.P : Proc(u, A)[U]}\ u \geq t$$

Fig. 1. Controlled NBA – type system

$$\textsc{Nil}\ \Gamma \vdash_{\text{CSl}} 0 : Proc(t) \qquad \textsc{Par}\ \frac{\Gamma \vdash_{\text{CSl}} P : Proc(t_P) \qquad \Gamma \vdash_{\text{CSl}} Q : Proc(t_Q)}{\Gamma \vdash_{\text{CSl}} P \mid Q : Proc(t_P + t_Q)}$$

$$\textsc{VarProc}\ \frac{\Gamma(X) = Proc(t)}{\Gamma \vdash_{\text{CSl}} X : Proc(u)}\ u \geq t \qquad \textsc{Rec}\ \frac{\Gamma, X : Proc(t) \vdash_{\text{CSl}} P : Proc(t)}{\Gamma \vdash_{\text{CSl}} \texttt{rec}\, X.P : Proc(u)}\ u \geq t$$

$$\textsc{Receive-Info}\ \frac{\Gamma, z : W \vdash_{\text{CSl}} P : T \qquad \Gamma \vdash_{\text{CSl}} c : IChan(W)}{\Gamma \vdash_{\text{CSl}} c^{\eta}(\lambda z).P : T}$$

$$\textsc{Emit-Info}\ \frac{\Gamma \vdash_{\text{CSl}} P : T \qquad \Gamma \vdash_{\text{CSl}} c : IChan(W) \qquad \Gamma \vdash_{\text{CSl}} z : W}{\Gamma \vdash_{\text{CSl}} \overline{c}^{\eta}(z).P : T}$$

$$\textsc{Receive-Move}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CSl}} P : Proc(t) \\ \Gamma \vdash_{\text{CSl}} c : MChan(S) \\ \Gamma \vdash_{\text{CSl}} z_1 : S \qquad \cdots \qquad \Gamma \vdash_{\text{CSl}} z_n : S\end{array}}{\Gamma \vdash_{\text{CSl}} c^{\eta}\{z_1, \cdots, z_k\}.P : Proc(t + k * e)}\ k \geq 0, S = Seal(s, e)$$

$$\textsc{Emit-Move}\ \frac{\begin{array}{c}\Gamma \vdash_{\text{CSl}} P : Proc(t) \\ \Gamma \vdash_{\text{CSl}} c : MChan(S) \qquad \Gamma \vdash_{\text{CSl}} z : S\end{array}}{\Gamma \vdash_{\text{CSl}} \overline{c}^{\eta}\{z\}.P : Proc(t - e)}\ t \geq e, S = Seal(s, e)$$

Fig. 2. Controlled Seals – type system

$$\textsc{UnivEps}\ \Gamma \vdash_{\mathrm{CN}\pi} \epsilon \qquad \textsc{UnivPar}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} R_1 \qquad \Gamma \vdash_{\mathrm{CN}\pi} R_2}{\Gamma \vdash_{\mathrm{CN}\pi} R_1; R_2}$$

$$\textsc{Site}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} l : \mathit{Site}(s) \qquad \Gamma \vdash_{\mathrm{CN}\pi} a : \mathit{Run}(e)}{\Gamma \vdash_{\mathrm{CN}\pi} l[a]}\ e \leq s$$

$$\textsc{RunNil}\ \Gamma \vdash_{\mathrm{CN}\pi} \emptyset : \mathit{Run}(0)$$

$$\textsc{RunAgent}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} a : \mathit{Ag}(\_, e) \qquad \Gamma \vdash_{\mathrm{CN}\pi} P : \mathit{Proc}(t)}{\Gamma \vdash_{\mathrm{CN}\pi} a\,\{P\} : \mathit{Run}(e + t)}$$

$$\textsc{RunPar}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} A : \mathit{Run}(e_A) \qquad \Gamma \vdash_{\mathrm{CN}\pi} B : \mathit{Run}(e_B)}{\Gamma \vdash_{\mathrm{CN}\pi} A, B : \mathit{Run}(e_A + e_B)}$$

$$\textsc{Try}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} P : T \qquad \Gamma \vdash_{\mathrm{CN}\pi} Q : T}{\Gamma \vdash_{\mathrm{CN}\pi} \texttt{try}\ c?p = P\ \texttt{timeout}\ n \to Q : T}$$

$$\textsc{Create}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} a : \mathit{Ag}(g, e) \qquad \Gamma \vdash_{\mathrm{CN}\pi} P : \mathit{Proc}(t)}{\Gamma \vdash_{\mathrm{CN}\pi} \overline{\texttt{create}}\ a\ P : \mathit{Proc}(t + e + g)}$$

$$\textsc{NewAg}\ \frac{\begin{array}{c}\Gamma \vdash_{\mathrm{CN}\pi} k : \mathit{Ag}(g, e) \qquad \Gamma, a : \mathit{Ag}(g, e) \vdash_{\mathrm{CN}\pi} P : \mathit{Proc}(t_P) \\ \Gamma, a : \mathit{Ag}(g, e) \vdash_{\mathrm{CN}\pi} Q : \mathit{Proc}(t_Q)\end{array}}{\Gamma \vdash_{\mathrm{CN}\pi} \texttt{agent}\ k\ \texttt{as}\ a = P\ \texttt{in}\ Q : \mathit{Proc}(t_Q)}\ t_P \leq g$$

$$\textsc{Migrate}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} a : \mathit{Ag}(g, e) \qquad \Gamma \vdash_{\mathrm{CN}\pi} P : \mathit{Proc}(t) \qquad \Gamma \vdash_{\mathrm{CN}\pi} t : \mathit{Site}(\_)}{\Gamma \vdash_{\mathrm{CN}\pi} \texttt{migrate}\ a\ \texttt{to}\ t\ P : \mathit{Proc}(t')}\ t \leq g$$

$$\textsc{Enter}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} a : \mathit{Ag}(g, e) \qquad \Gamma \vdash_{\mathrm{CN}\pi} P : \mathit{Proc}(t)}{\Gamma \vdash_{\mathrm{CN}\pi} \overline{\texttt{enter}}\ a\ P : \mathit{Proc}(t + e + g)}$$

$$\textsc{Exit}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} a : \mathit{Ag}(\_, e) \qquad \Gamma \vdash_{\mathrm{CN}\pi} P : \mathit{Proc}(t)}{\Gamma \vdash_{\mathrm{CN}\pi} \overline{\texttt{exit}}\ a\ P : \mathit{Proc}(t - e)}\ t \geq e$$

$$\textsc{IfLocal}\ \frac{\Gamma \vdash_{\mathrm{CN}\pi} P : T \qquad \Gamma \vdash_{\mathrm{CN}\pi} Q : T}{\Gamma \vdash_{\mathrm{CN}\pi} \texttt{iflocal}\ \langle a \rangle\, c!v\ \texttt{then}\ P\ \texttt{else}\ Q\ : T}$$

Fig. 3. Controlled Nomadic π – type system

$$\textsc{K-NameProcess}\ \frac{\Gamma \vdash_{\mathrm{CK}} a : Kell(s,e)}{\Gamma \vdash_{\mathrm{CK}} a : Proc(t)} \qquad \textsc{K-Trig}\ \frac{\Gamma, \gamma \vdash_{\mathrm{CK}} P : T \qquad \Gamma \vdash_{\mathrm{CK}} \xi : Pat(\gamma)}{\Gamma \vdash_{\mathrm{CK}} \xi \rhd P : T}$$

$$\textsc{K-Rel}\ \frac{\Gamma \vdash_{\mathrm{CK}} a : Kell(s_a, e_a) \qquad \Gamma \vdash_{\mathrm{CK}} P : Proc(t_P)}{\Gamma \vdash_{\mathrm{CK}} a \lhd P : Proc(t_P - e_a)}\ t_P \geq e_a$$

Fig. 4. Controlled Kells – type system