ELEMENTS OF AN OBJECT-BASED MODEL FOR DISTRIBUTED AND MOBILE COMPUTATION*

Jean-Bernard STEFANI, Florence GERMAIN
France Telecom R&D
{jeanbernard.stefani, florence.germain}@francetelecom.fr

Elie NAJM
ENST Paris
najm@res.enst.fr

Abstract This paper introduces an abstract model for distributed and mobile computation, based on a notion of domains. The model is shown expressive enough to simulate faithfully several recent distributed process calculi such as the Mobile Ambient calculus and the DJoin calculus. This in turn hints at the relevance of the model as a basis for the development of a primitive formal model for distributed and mobile programming.

Keywords: Distributed and mobile computation, formal model, distributed process calculi, π-calculus, mobile ambients, DJoin calculus.

1. INTRODUCTION

Despite advances, during the last decade, on mobile process calculi, a primitive model for distributed and mobile programming remains elusive. For this reason, several recent works, e.g. [1, 4, 6, 8, 22], have introduced distributed process calculi, which aim to capture features such as the distribution of resources on different locations, process mobility, the impact of failures and security on the behavior of a system, etc.

These distributed process calculi share several characteristics :

- they are based on some asynchronous variant of the π-calculus [13];

*This work has been supported in part by the RNRT project Marvel.

- they introduce a notion of locality to capture different features of distributed programming.

Despite their similarities, it is difficult to directly compare these calculi, for the notions of localities they introduce obey very different rules, reflecting the different kinds of phenomena they are intended to capture (mobility across nodes, failure models, administrative domains, etc). Apart from [7], which defines (and proves correct) a translation of the Ambient calculus [4] into the join calculus [6], we know of no systematic attempt to relate these different models of distributed computation.

This paper intends to fill this void :

1 by defining a more abstract and more general model of distributed, object-based computation;

2 by showing how these process calculi can be directly and faithfully simulated using instances of our model, hinting at the relevance of our model as a general model of distributed (and mobile) computation.

The paper is organized as follows :

- Section 2 motivates the introduction of the notion of *domain* as a central feature of a model for distributed and mobile computation.
- Section 3 introduces our model.
- Section 4 presents faithful simulations of two distributed process calculi in our model.
- Section 5 identifies perspectives for further work.

2. DOMAINS

In this section we introduce the notion of *domain*, and we derive a number of requirements for a model of distributed computation centered around this notion.

2.1. THE NOTION OF DOMAIN

Large distributed systems such as the World Wide Web or wide area telecommunications networks share several characteristics:

- asynchrony: processing and communication in these systems occur at different speeds, without reliance on a common global clock.
- partial failures: parts of a large distributed system may fail independently, according to different failure modes.

- system partitions: a large distributed system comprises several, possibly overlapping sub-systems, organized around different contingencies such as location, security, failure modes, administration and management.

This suggests that, from a modeling point of view, a distributed system should primarily be understood as a partitioned system. Components of a distributed system can be grouped in different, possibly overlapping sets, generally under the control of a single object or entity. Adopting the terminology from [9], we shall call *domains* such sets of components.

Domains come in many varieties, for instance :

- resource domains, encompassing hardware and software resources under the control of a single resource manager (e.g. information processing nodes in a computer network);
- language domains, delineating address spaces dedicated to the execution of programs written in a single programming language (e.g. capsules [10] or operating system processes);
- failure domains, encompassing entities that may fail together, according to certain failure modes (e.g. fail silent machines);
- administrative domains, encompassing computing resources under the control and management of a single authority (e.g. network management domains);
- security domains, corresponding to sets of nodes controlled by security policies and firewalls;
- naming domains, corresponding to sets of entities designated according to a given naming policy;
- locations, encompassing entities located at a given address in a computer network;
- technology domains, encompassing systems that are built with, or use a common hardware or software technology (e.g. interconnected machines that use a common set of communication protocols).

Notions of ambients in the Ambient calculus [4], of seals in the Seal calculus [22], of localities in the Join calculus [6], in the π_{1l}-calculus [1], and in the Dπ-calculus [8], seem to correspond to variants of this general notion of domain. In these calculi, domains are used to m ake

explicit the spatial structure of computations (as flat or tree-shaped sets of domains). However they differ in the (implicit) particular behavior they attach to their respective notion of domains.

These calculi adopt a homogeneous view of domains, each domain being capable of a single behavior (e.g. with respect to failure or process migration). In a large distributed system, however, domains of various kinds coexist. It is thus important, in a model for distributed computation, to take into account different kinds of domains, with possibly very different behaviors. This leads us to our first requirement on a model for distributed computation:

Requirement 1 *A model of distributed computation should include, as a primitive concept, a notion of domain, understood as a means to spatially partition a distributed computation, and should allow the definition of arbitrary domain behaviors.*

2.2. BEHAVIOR OF DOMAINS

¿From the examples given above, it appears that a domain can be characterized by two elements :

- a set of objects belonging to the domain, which we shall call the *content* of the domain;
- a controlling object, attached to the domain, which we shall call the *container* of the domain.

Let us consider what general requirements apply to these elements. First, the container of a domain can play the role of a filter with respect to messages which are sent to the domain's content. This type of behavior is manifest, for instance, in firewalls (containers for security domains), or in so-called component containers such as e.g. in EJB (Enterprise Java Beans) [20] or CORBA Components [17]. Modeling network nodes with their protocol machinery, at various levels of abstractions, also requires the introduction of domains and of their associated containers, capable of intercepting and processing incoming and outgoing protocol data units. We record this as

Requirement 2 *A domain container should be allowed to intercept and process messages which are going to, or coming from, the domain content, possibly changing its own state in the process.*

Another requirement is for domain containers to evolve exactly as a standard object, by receiving and sending messages to their environment. System management applications, for instance, typically require

access to domain containers (e.g. for querying the state of a given machine, for shutting it down, for activating it, etc). These applications further illustrate the fact that state changes of a domain container may cause correlative changes in the domain content. For instance, shutting down a given machine, understood as both a resource domain and a failure domain, will cause the different computations it supports to be terminated (and, perhaps, moved to different storage containers in a "passive" state). We record this as

Requirement 3 *A domain container should be allowed to send and receive messages, thereby changing its state, and possibly causing state changes in the domain content as well.*

A final requirement pertains to the creation of new domains and to the migration of contents between domains. Dynamic reconfiguration in an open distributed system implies the ability to introduce new objects and new subsystems, e.g. to increase the capacity of the system under a changing load, or to update parts of the system with new (hardware or software) technology. Modeling mobile systems, i.e. systems with physically mobile subsystems (portable PCs, PDAs, mobile phones, etc) or mobile software objects (e.g. mobile agents), implies the ability to move objects between different domains, corresponding to different spatial locations (e.g. from one radio cell to another, from one host to another). This translates into the following requirement

Requirement 4 *It should be possible to dynamically create new domains, and to move all, or part of a domain content from one domain to another.*

3. MODEL

We introduce in this section an abstract computational model that takes into account the various requirements identified in section 2 above.

The model is abstract in that it does not provide an effective means to describe the individual behavior of domains. Instead, the model postulates, for each domain, a behavior description which is expressed in terms of sets of possible transitions (given by operator B below). However, we shall see in section 4 below that this model can easily be instantiated to yield concrete (sub-)calculi. We shall discuss in section 5 perspectives for the definition of a concrete process calculus based on a notion of domain.

Our model is both object-based and domain-based in that:

- a distributed system is described, using our model, as a collection of concurrent, interacting domains (a *configuration*);

- each domain is, in turn, constituted by the composition of an object, the *domain container*, with a configuration, the *domain content.*

The operational semantics of our model is defined in the Chemical Abstract Machine style [2], using a structural equivalence, $\equiv$, and a reduction relation, $\rightarrow$.

3.1. SORTS AND OPERATORS

We postulate the existence of different sorts and operators. We adopt a postfix notation for operators. Sorts of our model are as follows:

- N denotes the set of *names* in our model. Intuitively, each object in the model exhibits different interfaces (as per the ODP Reference Model [9, 10]) which are identified by unique names. An interface constitutes a communication port. We use u, v, and their decorated variants to refer to names.
- O denotes the set of *objects* in our model. We use ω, ϖ and their decorated variants to refer to objects.
- M denotes the set of *messages* in our model. We use m and its decorated variants to refer to messages, and $\aleph$ and its decorated variants to refer to finite (possibly empty) bags of messages.[1]

  Messages are units of interaction between an object and its environment. In this paper, we only consider asynchronous message exchange as a means of communication. An immediate extension of the model would consist in the introduction of so-called *bindings*, e.g. along the lines of [15].

- S denotes the set of *configurations.* We use C, D, E, F and their decorated variants to refer to configurations. We assume that S has two distinguished elements, $\emptyset$ and $\perp$, which denote, respectively, an empty configuration and an invalid configuration. A configuration corresponds to a parallel composition of domains and messages. With operators $\|$ and $[\,]$ below, an abstract syntax for configurations is given by the following grammar :

$$C ::= \emptyset \mid \perp \mid m \mid \omega[C] \mid C \,\|\, C \qquad (1)$$

Operators in our model are (we omit injection operators from O into S, and from M into S) :

[1] We also use $\aleph$ to refer to a configuration $m_1 \,\|\, \ldots \,\|\, m_p$ of messages This is legitimate, thanks to the associative and commutative properties of the parallel operator, $\|$.

- $\|: \mathsf{S} \times \mathsf{S} \to \mathsf{S}$

  Operator $\|$ is the parallel composition operator in our model.

- $[\,] : \mathsf{O} \times \mathsf{S} \to \mathsf{S}$

  Operator $[\,]$ is a composition operator that constructs a domain $\omega[C]$ by associating an object ω and a configuration C. ω is the *container* of the domain $\omega[C]$, while C is the *content* of domain $\omega[C]$.

- $tgt : \mathsf{M} \to \mathsf{N}$

  $m.tgt$ denotes the destination of message m.

- $arg : \mathsf{M} \to (\mathsf{N} \cup \mathsf{S})^*$

  $m.arg$ denotes arguments (in finite number) of message m. An argument of a message can be either a name, or a configuration. We note $u\langle\tilde{l}\rangle$ a message with target u and with arguments given by vector $\tilde{l}$.

- $L_1 : \mathsf{O} \to \wp_f(\mathsf{N})$ [2]

  $\omega.L_1$ denotes the set of names born by object ω.

- $L : \mathsf{O} \to (\mathsf{S} \to \wp_f(\mathsf{N}))$.

  $\omega.L(C)$ denotes the set of names born by domain $\omega[C]$, and which are visible by its environment.

  Operator L is extended into an operator $L : \mathsf{S} \to \wp_f(\mathsf{N})$ on configurations through the following inductive definition :

  - $\emptyset.L = \emptyset$
  - $m.L = \bigcup\{C.L \cdot C \in m.arg\}$
  - $\omega[C].L = \omega.L(C)$
  - $(C_1 \parallel C_2).L = C_1.L \cup C_2.L$

  A domain $\omega[C]$ exhibits a number of interfaces which are visible by its environments as a set of names (given by $\omega.L(C)$). These names may differ from the names born by configuration C, e.g. we can have $\omega.L(C) \cap C.L = \emptyset$: a domain's container controls the visibility of the domain's content from other objects in the environment of the domain. In general, we will also have $\omega.L_2[\emptyset] \neq \omega.L_1$. This means that interfaces of an object ω may in general

[2] We use $\wp_f(A)$ to denote the set of finite subsets of a set A, $Bag_f(A)$ to denote the set of finite multisets built with elements from set A, and $\wp(A)$ to denote the set of subsets of set A.

be split between internal interfaces, i.e. interfaces which are only visible from the content of a domain $\omega[C]$, and external interfaces, i.e. interfaces which are visible from the environment of a domain $\omega[C]$.

- $K : \mathsf{O} \to (\mathsf{S} \to \wp_f(\mathsf{N}))$

  $\omega.K(C)$ denotes the set of names known by domain $\omega[C]$. Operator K is extended into an operator $K : \mathsf{S} \to \wp_f(\mathsf{N})$ on configurations through the following inductive definition:

  - $\emptyset.K = \emptyset$
  - $m.K = \bigcup\{C.K \cdot C \in m.arg\} \cup \bigcup\{m.arg \cdot m.arg \in \mathsf{N}\}$
  - $\omega[C].K = \omega.K(C)$
  - $(C_1 \parallel C_2).K = C_1.K \cup C_2.K$

- $B : \mathsf{O} \to (\mathsf{S} \to \wp(\mathsf{O} \times \mathsf{Bag}_f(\mathsf{M}) \times \mathsf{S}))$

  $\omega.B$ denotes the behavior of object ω. Given a configuration C, the behavior of domain $\omega[C]$ is defined by a set of possible transitions, i.e. of triplets of the form $\langle \omega, \aleph, D \rangle$. In a possible transition $\langle \omega, \aleph, D \rangle$, $\aleph$ is a multiset of messages targeted at names born by $\omega[C]$, and D denotes a new configuration, created at the end of the transition. The behavior of an object must obey certain consistency conditions which are given in Section 3.3. We use t and its decorated variants to denote possible transitions.

3.2. STRUCTURAL EQUIVALENCE

The structural equivalence between configurations is defined inductively as being the smallest relation satisfying the following rules:

$$C \equiv C \tag{2}$$

$$\frac{C \equiv D}{D \equiv C} \tag{3}$$

$$\frac{C_1 \equiv C_2,\ C_2 \equiv C_3}{C_1 \equiv C_3} \tag{4}$$

$$\frac{C_1 \equiv C_2}{C_1 \parallel D \equiv C_2 \parallel D} \tag{5}$$

$$\frac{C \equiv D}{\omega[C] \equiv \omega[D]} \tag{6}$$

$$\frac{\forall i \in \{1, \ldots, p\} \cdot l_i = k_i \text{ if } l_i \in \mathsf{N} \quad l_i \equiv k_i \text{ if } l_i \in \mathsf{S}}{u\langle l_1, \ldots, l_p \rangle \equiv u\langle k_1, \ldots, k_p \rangle} \tag{7}$$

$$C \parallel D \equiv D \parallel C \tag{8}$$

$$(C_1 \parallel C_2) \parallel C_3 \equiv C_1 \parallel (C_2 \parallel C_3) \tag{9}$$

$$C \parallel \emptyset \equiv C \tag{10}$$

$$C \parallel \bot \equiv \bot \tag{11}$$

Relation $\equiv$ is an equivalence relation: rules (2), (3), (4). It is also a congruence on configurations: rules (5), (6), (7). The parallel operator is associative, commutative, has $\emptyset$ as a neutral element, and $\bot$ as an absorbing element: rules (8), (9), (10), (11).

The value of proposition $C \equiv \bot$ is given by the following inductive definition:

- $(\emptyset \equiv \bot) = \mathsf{False}$
- $(m \equiv \bot) = \bigvee_{C \in m.arg} C \equiv \bot \ \vee \ \bigcap_{C \in m.arg} C.L \neq \emptyset$
- $(\omega[C] \equiv \bot) = (\omega.L_1 \cap C.L \neq \emptyset \ \vee \ C \equiv \bot)$
- $(C_1 \parallel C_2 \equiv \bot) = (C_1.L \cap C_2.L \neq \emptyset \ \vee \ C_1 \equiv \bot \ \vee \ C_2 \equiv \bot)$

Configuration $\bot$ is a technical artifact which is used to enforce the unicity of (visible) names in a configuration. The unicity of names in a given configuration C can be simply stated as : $C \not\equiv \bot$. Note that the definition above disallows a recursive building of domains, e.g. in domain $\omega[C]$, object ω cannot appear as a domain container in C.

3.3. OBJECT BEHAVIOR

Possible transitions associated with the behavior of an object (as given by operator B) are subject to certain consistency conditions. Possible transitions $t = \langle \omega, \aleph, D \rangle$ associated with the behavior of an object ω can take two different forms, according to the structure of the resulting configuration D:

- Standard domain evolution:

$$D = \omega'[E] \parallel F \tag{12}$$

  where F is a configuration and $\omega.L_1 \subseteq \omega'.L_1$.

  Intuitively, object ω' corresponds to the "new state" of object ω after the transition. Interfaces of an object persist during a transition, which is captured by the condition $\omega.L_1 \subseteq \omega'.L_1$.

- Domain "passivation":

$$D = m(\omega'[E]) \parallel F \tag{13}$$

where F is a configuration, $m(\omega'[E])$ is a message that has $\omega'[E]$ as an argument, and $\omega.L_1 \subseteq \omega'.L_1$.

Intuitively, in this sort of transition, a domain "passivates" itself into a message, targeting a known interface. This feature is crucial for the expressive power of our model.

We shall denote by $H_1(t, C)$ the fact that these two possibilities are allowed:

$$H_1(\langle \omega, \aleph, D \rangle, C) \quad =_{def} \quad \exists \omega', E, F \cdot \omega.L_1 \subseteq \omega'.L_1 \\ \wedge \ (D = \omega'[E] \parallel F \ \vee \ D = \pi[\omega'[E]] \parallel F)$$

If t is a possible transition of the behavior of object ω with content C, i.e. $t \in \omega.B(C)$, then t must be of the form $\langle \omega, \aleph, D \rangle$ and must verify condition $H(t, C)$, defined by:

$$H(\langle \omega, \aleph, D \rangle, C) \quad =_{def} \quad D.K \subseteq \omega[C].K \cup \aleph.K \cup D.L \qquad (14)$$
$$\wedge \ \aleph.tgt \subseteq \omega[C].L \qquad (15)$$
$$\wedge \ D \not\equiv \bot \qquad (16)$$

Condition (14) stipulates that names known after a transition must have been known by the domain prior to the transition, must appear as arguments of incoming messages in $\aleph$, or must be born by new interfaces created during the transition. Condition (15) stipulates that the target of incoming messages in a transition are restricted to names born by the domain. Condition (16) stipulates that the configuration resulting from the transition must respect the unicity constraint on names.

To ensure compatibility with the structural equivalence defined in the preceding section, condition $H_2(t, C', \aleph', D')$ must be verified. Condition $H_2(t, C', \aleph', D')$ is defined thus, with $t = \langle \omega, \aleph, D \rangle$:

$$H_2(t, C', \aleph', D') \quad =_{def} \quad t \in \omega.B(C) \ \wedge \ C \equiv C' \ \wedge \ \aleph \equiv \aleph' \ \wedge \ D \equiv D' \\ \Rightarrow \ \langle \omega, \aleph', D' \rangle \in \omega.B(C')$$

Intuitively, $H_2(t, C', \aleph', D')$ states that the behavior of an object for a given configuration C remains the same for any equivalent configuration C'.

The consistency condition on object behavior thus takes the form:

$$\mathsf{Cons}(\omega) \quad =_{def} \quad \forall C \in \mathsf{S} \ \forall t \in \omega.B(C) \cdot H(t, C) \ \wedge \ H_1(t, C) \\ \wedge \ (\forall C', \aleph', D' \cdot H_2(t, C', \aleph', D'))$$

3.4. REDUCTION RELATION

In our model, a system corresponds to a transition system whose set of states is given by S, and whose transitions are given by the reduction

relation $\rightarrow$, inductively defined as the smallest relation derived using the following rules:

- Domain evolution:

$$\frac{\langle \omega, \aleph, D \rangle \in \omega.B(C)}{\omega[C] \parallel \aleph \rightarrow D} \quad \text{provided } \mathsf{Cons}(\omega) \tag{17}$$

This rule stipulates that a domain $\omega[C]$ evolves in accordance with the (consistently) defined behavior for object ω, given configuration C.

- Parallel evolution:

$$\frac{C \parallel D \not\equiv \perp \quad C \rightarrow C' \quad C' \parallel D \not\equiv \perp}{C \parallel D \rightarrow C' \parallel D} \tag{18}$$

This rule stipulates that configurations in parallel can evolve independently, provided the unicity of names is preserved.

- Evolution of equivalent configurations:

$$\frac{C \equiv D \quad D \rightarrow D' \quad D' \equiv C'}{C \rightarrow C'} \tag{19}$$

This rule stipulates that evolutions of equivalent configurations are identical.

3.5. DISCUSSION

The model we have just presented, which we shall call the κ model, meets in large part the requirements set out in section 2 :

1 The notion of domain is primitive in the κ model and allows for arbitrary domain behaviors (as manifested by operator B in the definition of an object behavior).

2 In the κ model, a domain container can intercept and process incoming and outgoing messages ultimately targeted at the domain content. This feature is illustrated in the simulations presented in section 4, e.g. in rules (28) and (30).

3 In the κ model, a domain container constitutes a a bona-fide object, with its own behavior and the capacity to receive and emit messages. Simulations in section 4 rely heavily on the modeling of

a domain container (ambient or DJoin locality) as an object (see in particular rules (28) to (35), and rules (50) to (55)).

4 Domain creation and domain passivation are integral features of the κ model, as defined in section 3.3. Again, these features are crucial for the simulations in section 4, as illustrated e.g. in rules (28), to (31).

Further evidence in meeting the requirements laid out in Section 2 can be found in the report [18], as well as a more detailed analysis of the requirements. In addition, it is shown that the κ model constitutes a conservative extension of the RM-ODP computational model [9], and that the κ model captures the main constructions involved in concurrent reflective systems such as ABCL/R [23] and Coda [12].

4. SIMULATIONS

We present in this section faithful simulations, in the κ model, of the Mobile Ambients calculus [4] and of the DJoin calculus [6, 7]. These simulations provide a direct characterization of the kinds of domains introduced by these two formalisms (ambients and localities, respectively). More precisely, they show that, using our model, it is possible to fully characterize notions of domains introduced in these calculi as objects (domain containers) endowed with specific behaviors.

Throughout the section, we use a simplified version of our model, where $\omega[C].L = \omega.L_1 \cup C.L$, and where $\omega[C].K = \omega[\emptyset].K \cup C.K$.

We describe object behaviors using an operational style, where possible transitions are given in the form of inference rules. Intuitively, a rule

$$\frac{Premiss}{Config_1 \rightarrow Config_2}$$

describes a possible transition of the object present in configuration $Config_1$, provided the condition captured by $Premiss$ is met.

4.1. SIMULATION OF MOBILE AMBIENTS

For simplicity, we consider the simple ambient calculus (i.e. without communication), as defined in [4]. The simulation of mobile ambients in our model uses the following elements:

- Every ambient of name n is simulated with an object A_n^u, where $u \in \mathsf{N}$.

- We denote by Name the set of names used in the ambient calculus. We use n, m, and their decorated variants to denote elements of Name.
- To each name n of the ambient calculus is associated a gateway object ϖ_n, which is required in order to simulate the presence of multiple ambients with the same name n.
- A simulation of an ambient calculus process makes use of a global container $\omega_0(U, V)$, which is parameterized with two sets of names $U \subseteq \mathsf{Name}$ and $V \subseteq \mathsf{N}$. This object is in charge of managing name creation.
- We assume given an injection ψ, from Name into N.
- We use the following distinguished names from N : ι, in, out, open, act, include, extrude.

Let P be an ambient calculus process. A simulation of P in the κ model is given by function T_0, which is defined by:

$$T_0(P) = \omega_0(U, V)[T(P) \parallel_{n \in U} \varpi_n[]] \tag{20}$$

where $U = \{n \cdot n \in FV(P)\}$, et $V = \psi(U)$.
Function T is defined inductively by:

- $T(0) = \emptyset$
- $T(P \mid Q) = T(P) \parallel T(Q)$
- $T(!P) = \mathsf{bang}[T(P)]$
- $T(n[P]) = \mathsf{amb}(n, P)$
- $T(in\, n.P) = \psi(n)\langle \mathsf{in}, T(P)\rangle$
- $T(out\, n.P) = \psi(n)\langle \mathsf{out}, T(P)\rangle$
- $T(open\, n.P) = \psi(n)\langle \mathsf{open}, T(P)\rangle$
- $T(\nu n.P) = \mathsf{new}(n, P)$

Additional objects used by the simulation are defined below.
The global containers $\omega_0(U, V)$ are defined by $\omega_0(U, V).L = \emptyset$ and the following behavior:

$$\frac{m \notin U \quad U' = U \cup \{m\} \quad V' = V \cup \{\psi(m)\}}{\omega_0(U, V)[\mathsf{new}(n, P)] \to \omega_0(U', V')[\varpi_m[] \parallel T(P\{m/n\})]} \tag{21}$$

$$\frac{u \notin V \cup \psi(\mathsf{Name}) \quad V' = V \cup \{u\}}{\omega_0(U, V)[\mathsf{amb}(n, P)] \to \omega_0(U, V')[A_n^u[T(P)] \parallel \psi(n)\langle \mathsf{act}, u\rangle]} \tag{22}$$

$$\frac{u \in V}{\omega_0(U, V)[\iota\langle \mathsf{gate}, C\rangle \parallel D] \to \omega_0(U, V)[u\langle \mathsf{gate}, C\rangle \parallel D]} \tag{23}$$

$$\omega_0(U,V)[\iota\langle \mathsf{gate}, C\rangle \parallel D] \to \omega_0(U,V)[C \parallel D] \tag{24}$$

$$\frac{C \to C'}{\omega_0(U,V)[C] \to \omega_0(U,V)[C']} \tag{25}$$

$$\frac{\omega_0(U,V)[C] \to \omega_0(U',V')[C']}{\omega_0(U,V)[C \parallel D] \to \omega_0(U',V')[C' \parallel D]} \tag{26}$$

$$\frac{\omega_0(U,V)[C] \to \omega_0(U',V')[C']}{\omega_0(U,V)[A_a^T[C]] \to \omega_0(U',V')[A_a^T[C']]} \tag{27}$$

Objects A_n^u are defined by $A_n^u.L = \{u\}$, and the following behavior:

$$A_n^u[C \parallel v\langle \mathsf{in}, D\rangle] \to v\langle \mathsf{include}, A_n^u[C \parallel D]\rangle \tag{28}$$

$$A_n^u[C] \parallel u\langle \mathsf{include}, D\rangle \to A_n^u[C \parallel D] \tag{29}$$

$$A_n^u[C \parallel v\langle \mathsf{out}, D\rangle] \to v\langle \mathsf{extrude}, A_n^u[C \parallel D]\rangle \tag{30}$$

$$A_n^u[C \parallel u\langle \mathsf{extrude}, D\rangle] \to A_n^u[C] \parallel D \tag{31}$$

$$A_n^u[C] \parallel u\langle \mathsf{open}, D\rangle \to \epsilon_u[] \parallel C \parallel D \tag{32}$$

$$\frac{C \to C'}{A_n^u[C] \to A_n^u[C']} \tag{33}$$

$$A_n^u[C \parallel \varpi_m[D]] \to A_n^u[C] \parallel \varpi_m[D] \tag{34}$$

$$A_n^u[C] \parallel u\langle \mathsf{gate}, D\rangle \to A_n^u[C \parallel D] \tag{35}$$

Object ϵ_u is just defined by $\epsilon_u.L = \{u\}$. Object $\mathsf{new}(n, P)$ is just defined by $\mathsf{new}(n, P).L = \emptyset$.

Object $\mathsf{bang}(P)$ is defined by $\mathsf{bang}(P).L = \emptyset$ and the following behavior:

$$\mathsf{bang}[T(P)] \to \mathsf{bang}[T(P)] \parallel T(P) \tag{36}$$

The gateway objects, ϖ_n are defined by $\varpi_n.L = \psi(n)$ and the following behavior:

$$\varpi_n[C] \to \iota\langle \mathsf{gate}, \varpi_n[C]\rangle \tag{37}$$

$$\varpi_n[C] \parallel \psi(n)\langle \mathsf{act}, u\rangle \to \varpi_n[C \parallel \psi(n)\langle \mathsf{act}, u\rangle] \tag{38}$$

$$\frac{\tilde{l} = l_1, \ldots, l_p \quad l_1 \in \{\mathsf{include}, \mathsf{extrude}, \mathsf{open}\}}{\varpi_n[C \parallel \psi(n)\langle \mathsf{act}, u\rangle] \parallel \psi(n)\langle \tilde{l}\rangle \to \varpi_n[C \parallel \psi(n)\langle \mathsf{act}, u\rangle] \parallel u\langle \tilde{l}\rangle} \tag{39}$$

Let $\xi[\cdot]$ be a context from κ, and P an ambient calculus process. Context $\xi[\cdot]$ and process P are said to be compatible if $\xi[\cdot]$ takes the form $\xi[\cdot] = \omega_0(U,V)[\|_{n\in U}\ \varpi_n[\aleph_n] \parallel \cdot \parallel_{u\in V_1}\ \epsilon_u[]]$, with $\psi(U) \subseteq V$, $V_1 \subseteq V$, $\aleph_n$ is a set of messages such that $\psi(n)\langle \mathsf{act}, u\rangle$, $u \in V$, and if P is such that $FV(P) \subseteq U$.

One can then show the following:

Proposition 1 *Let P and Q be two ambient calculus processes, such that $P \rightarrow_{ambient} Q$. Then, for all κ contexts $\xi[\cdot]$, such that $\xi[\cdot]$ and P are compatible, there is a κ context $\bar{\xi}[\cdot]$, such that $\bar{\xi}[\cdot]$ and Q are compatible, and $\xi[T(P)] \rightarrow_\kappa^* \bar{\xi}[T(Q)]$.*

Proof By induction on the form of derivation contexts in the ambient calculus. $\diamondsuit$

Let us just illustrate a proof step and the way the simulation works. We consider the following rule from the ambient calculus:

$$n[inm.P \mid Q] \mid m[R] \rightarrow m[R \mid n[P \mid Q]] \tag{40}$$

Let $A = n[inm.P \mid Q] \mid m[R]$ and $B = m[R \mid n[P \mid Q]]$. We have the following derivations, where $V' = V \cup \{u, v\}$:

$\xi[T(A)]$
$= \omega_0(U,V)[\varpi_n[\aleph_n] \parallel \varpi_m[\aleph_m] \parallel \mathsf{amb}(n, inm.P \parallel Q) \parallel \mathsf{amb}(m, R)]$
$\rightarrow^* \omega_0(U,V')[\varpi_n[\aleph_n] \parallel \varpi_m[\aleph_m] \parallel$
$A_n^u[T(inm.p \mid Q)] \parallel \psi(n)\langle \mathsf{act}, u\rangle \parallel A_m^v[T(R)] \parallel \psi(m)\langle \mathsf{act}, v\rangle]$
$= \omega_0(U,V')[\varpi_n[\aleph_n] \parallel \varpi_m[\aleph_m] \parallel$
$A_n^u[\psi(m)\langle \mathsf{in}\rangle \parallel T(P) \parallel T(Q)] \parallel \psi(n)\langle \mathsf{act}, u\rangle \parallel A_m^v[T(R)] \parallel \psi(m)\langle \mathsf{act}, v\rangle]$
$\rightarrow^* \omega_0(U,V')[\varpi_n[\aleph_n \parallel \psi(n)\langle \mathsf{act}, u\rangle] \parallel \varpi_m[\aleph_m \parallel \psi(m)\langle \mathsf{act}, v\rangle] \parallel$
$\psi(m)\langle \mathsf{include}, A_n^u[T(P) \parallel T(Q)]\rangle \parallel A_m^v[T(R)]]$
$\rightarrow \omega_0(U,V')[\varpi_n[\aleph_n \parallel \psi(n)\langle \mathsf{act}, u\rangle] \parallel \varpi_m[\aleph_m \parallel \psi(m)\langle \mathsf{act}, v\rangle] \parallel$
$v\langle \mathsf{include}, A_n^u[T(P) \parallel T(Q)]\rangle \parallel A_m^v[T(R)]]$
$\rightarrow \omega_0(U,V')[\varpi_n[\aleph_n \parallel \psi(n)\langle \mathsf{act}, u\rangle] \parallel \varpi_m[\aleph_m \parallel \psi(m)\langle \mathsf{act}, v\rangle] \parallel$
$A_m^v[T(R) \parallel A_n^u[T(P) \parallel T(Q)]]]$
$= \xi'[T(B)]$

4.2. SIMULATION OF THE DJOIN-CALCULUS

We consider the DJoin-calculus as defined in [6, 7]. Simulating this calculus in our κ model involves several elements:

- Each locality $a[]$ from the DJoin-calculus, for a an elementary locality, is simulated by a domain $A_a[]$. A higher-level locality $a_1 \ldots a_p[]$ is simulated by a set of embedded domains $A_{a1}[\ldots[A_{ap}[]]\ldots]$.
- To each receiver x of the Djoin-calculus is associated a gateway object ϖ_x. Simulating a definition in the DJoin-calculus involves

the creation of gateways for all the names which are defined at the first level in a Djoin definition D.

- We assume given an injection ϕ from the set Name of names of the DJoin-calculus, into N, the set of names of the κ model. ϕ associates to each name of the DJoin the name of the gateway or of the domain that corresponds to this name.
- The simulation of a DJoin calculus process makes use of a global container $\omega_0(U, V)$, which is parameterized by two sets $U \subseteq$ Name and $V \subseteq$ N.
- We use x, y, z to denote a channel in the DJoin calculus, α, β to denote a higher level locality in the DJoin, a, b to denote base level locality in the DJoin, and n, m to denote arbitrary localities (concatenations of base level localities a, b). We use u, v to denote names in κ, and l to denote an arbitrary message argument in κ.
- We use the following distinguished names from N : go, act, hab.

Let P be a DJoin-calculus process. We define a simulation of P in κ with function T_0, defined as:

$$T_0(P) = \omega_0(U, V)[\, T(P)\,] \tag{41}$$

where $U = \{n \cdot n \in FV(P)\}$ and $V = \{\phi(n) \cdot n \in U\}$.
Function T is defined inductively by:

- $T(x\langle\tilde{y}\rangle) = \phi(x)\langle\phi(\tilde{y})\rangle$
- $T(P_1 \mid P_2) = T(P_1) \parallel T(P_2)$
- $T(0) = \emptyset$
- $T(\mathsf{def}\, D \,\mathsf{in}\, P) = \mathsf{def}(D, P)$
- $T(J \triangleright P) = \mathsf{prep}(J, P)$
- $T(D_1 \wedge D_2) = T(D_1) \parallel T(D_2)$
- $T(m[D : P]) = \mathsf{loc}(m, D, P)$
- $T(\mathsf{go}(a); P) = \mathsf{go}(a, P)$

Objects which are used by the simulation are defined below.
Global containers $\omega_0(U, V)$ are defined by $\omega_0(U, V).L = \emptyset$, and the following behavior. In what follows we note

$$A(D) = \|_{a \in P^0} A_a[\, \|_{b \in P^1_a} A_b[\ldots[\, \|_{b \in P^{N(a)}_{a \ldots a_{N(a)-1}}} A_b[\,]\,]\ldots]\,]$$

with $P^0 = \{a_0 \cdot a_0 a_1 \ldots a_q \in M\}$, $P^k_u = \{a_k \cdot a_0 a_1 \ldots a_k \ldots a_q \in M,\ a_0 \ldots a_{k-1} = u\}$, and $N(a) = \max\{k \cdot P^k_u \neq \emptyset,\ u = a a_1 \ldots a_q\}$.

$$\frac{\mu = \{x \mapsto y \cdot y \notin U, x \in dv(D)\} \quad U' = U \cup \mathsf{ran}(\mu) \quad V' = V \cup \phi(\mathsf{ran}(\mu))}{\omega_0(U, V)[\mathsf{def}(D, P)] \leftrightarrow \omega_0(U', V')[\mathsf{defn}(D\{y/x\}_{(x,y)\in\mu}, P\{y/x\}_{(x,y)\in\mu})\,]} \tag{42}$$

$$\frac{D = \bigwedge_{i\in I}(J_i \triangleright P_i)\ \bigwedge_{m\in M} m[D_m : P_m]}{\omega_0(U,V)[\mathsf{defn}(D,P)] \leftrightarrow \omega_0(U',V')[T(D) \parallel T(P) \parallel_{x\in X} \varpi_x[] \parallel A(D)\ ]} \tag{43}$$

$$\frac{m = a_0a_1 \ldots a_p \quad n = a_1 \ldots a_p}{\omega_0(U,V)[\mathsf{loc}(m,D,P)] \leftrightarrow \omega_0(U,V)[\phi(a_0)\langle \mathsf{hab}, \mathsf{loc}(n,D,P)\rangle\ ]} \tag{44}$$

$$\omega_0(U,V)[\mathsf{loc}(a,D,P)] \leftrightarrow \omega_0(U,V)[\phi(a)\langle \mathsf{hab}, T(D) \parallel T(P)\rangle\ ] \tag{45}$$

$$\frac{\rho = \{x \mapsto u \cdot\ x \in dv(J),\ u \notin V \cup \phi(\mathsf{Name})\} \quad V' = V \cup \mathsf{ran}(\rho)}{\omega_0(U,V)[\mathsf{prep}(J,P)] \leftrightarrow \omega_0(U,V')[\mathsf{react}(J,P,\rho) \parallel_{x\in dv(J)} \phi(x)\langle \mathsf{act}, \rho(x)\rangle]\ ]} \tag{46}$$

$$\frac{C \rightarrow C'}{\omega_0(U,V)[C] \rightarrow \omega_0(U,V)[C']} \tag{47}$$

$$\frac{\omega_0(U,V)[C] \rightarrow \omega_0(U',V')[C']}{\omega_0(U,V)[C \parallel D] \rightarrow \omega_0(U',V')[C' \parallel D]} \tag{48}$$

$$\frac{\omega_0(U,V)[C] \rightarrow \omega_0(U',V')[C']}{\omega_0(U,V)[A_a[C]] \rightarrow \omega_0(U',V')[A_a[C']]} \tag{49}$$

An object A_a is defined by $A_a.L = \{\phi(a)\}$ and by the following behavior:

$$\frac{C \rightarrow C'}{A_a[C] \rightarrow A_a[C']} \tag{50}$$

$$A_a[C] \parallel \phi(a)\langle \mathsf{hab}, D\rangle \leftrightarrow A_a[C \parallel D] \tag{51}$$

$$\frac{u \in C.L}{A_a[C] \parallel u\langle \tilde{l}\rangle \rightarrow A_a[C \parallel u\langle \tilde{l}\rangle]} \tag{52}$$

$$A_a[C \parallel u\langle \tilde{l}\rangle] \rightarrow A_a[C] \parallel u\langle \tilde{l}\rangle \tag{53}$$

$$A_a[C \parallel \mathsf{go}(a,P)] \rightarrow \phi(a)\langle \mathsf{go}, A_a[C \parallel T(P)]\rangle \tag{54}$$

$$A_a[C] \parallel \phi(a)\langle \mathsf{go}, D\rangle \rightarrow A_a[C \parallel D] \tag{55}$$

Objects $\mathsf{prep}(J,P,\rho)$ are defined by $\mathsf{prep}(J,P,\rho).L = \emptyset$.
Objects $\mathsf{react}(J,P,\rho)$ are defined by $\mathsf{react}(J,P,\rho).L = \mathsf{ran}(\rho)$, and by the following behavior:

$$\frac{J = \parallel_{x\in X} x\langle \tilde{y}\rangle}{\mathsf{react}(J,P,\rho) \parallel_{x\in X} \phi(x)\langle \phi(\tilde{z})\rangle \rightarrow \mathsf{react}(J,P,\rho) \parallel T(P\{\tilde{z}/\tilde{y}\})} \tag{56}$$

Gateway objects, ϖ_x are defined by $\varpi_x.L = \{\phi(x)\}$and by the following behavior:

$$\varpi_x[C] \parallel \phi(x)\langle \mathsf{act}, u\rangle \leftrightarrow \varpi_x[C \parallel \phi(x)\langle \mathsf{act}, u\rangle] \tag{57}$$

$$\frac{\tilde{l} = l_1, \ldots, l_p \quad l_1 \neq \mathsf{act}}{\varpi_x[C \parallel \phi(x)\langle \mathsf{act}, u\rangle] \parallel \phi(x)\langle \tilde{l}\rangle \rightarrow \varpi_x[C \parallel \phi(x)\langle \mathsf{act}, u\rangle] \parallel u\langle \tilde{l}\rangle} \quad (58)$$

Let $\xi[\cdot]$ be a context from κ, and P be a DJoin-calculus process. Context $\xi[\cdot]$ and process P are said to be compatible if $\xi[\cdot]$ takes the form $\xi[\cdot] = \omega_0(U,V)[\|_{n\in U}\ \varpi_x[\aleph_x] \parallel \cdot \parallel_{a_{ij}\in U_1} A_{a_{i1}}[\ldots A_{a_{ip_i}}[]\ldots]]$, with $\phi(U) \subseteq V$, $U_1 \subseteq U$, $\aleph_x$ set of messages of the form $\phi(x)\langle \mathsf{act}, u\rangle$, $u \in V$, and if P is such that $FV(P) \subseteq U$.

We then have the following:

Proposition 2 *Let P and Q be two DJoin-calculus processes such that $P \rightarrow_{join} Q$. Then, for any κ context $\xi[\cdot]$ such that $\xi[\cdot]$ and P are compatible, there is a κ context $\bar{\xi}[\cdot]$, such that $\bar{\xi}[\cdot]$ and Q are compatible and $\xi[T_\rho(P)] \rightarrow^*_\kappa \bar{\xi}[T_{\bar{\rho}}(Q)]$.*

Proof By induction on the form of derivation contexts in the DJoin-calculus. $\diamondsuit$

We can illustrate a proof step and the way the simulation works with an instance of the GO rule of the Djoin-calculus:

$$d.b[D' : P'] \wedge e.a[D : P|go(b); Q] \rightarrow d.b[D' : P' \wedge d.b.a[D : P|Q]$$

Let $R = d.b[D' : P'] \wedge e.a[D : P|go(b); Q]$ and $S = d.b[D' : P'] \wedge d.b.a[D : P|Q]$, and let $\xi[\cdot] = \omega_0(U,V)[\ \eta \parallel A_d[A_b[]] \parallel A_e[A_a[]] \parallel \cdot\ ]$. We have the following derivations:

$\xi[T(R)]$
$= \omega_0(U,V)[\eta \parallel A_d[A_b[]] \parallel A_e[A_a[]] \parallel \mathsf{loc}(d.b, D', P') \parallel \mathsf{loc}(e.a, D, P \parallel \mathsf{go}(b); Q)]$
$\rightarrow^* \omega_0(U,V)[\eta \parallel A_d[A_b[T(D') \parallel T(P')]] \parallel A_e[A_a[T(D) \parallel T(P) \parallel \mathsf{go}(b, Q)]]]$
$\rightarrow \omega_0(U,V)[\eta \parallel A_d[A_b[T(D') \parallel T(P')]] \parallel A_e[\phi(b)\langle \mathsf{go}, A_a[T(D) \parallel T(P) \parallel T(Q)]\rangle]]$
$\rightarrow \omega_0(U,V)[\eta \parallel A_d[A_b[T(D') \parallel T(P')]] \parallel \phi(b)\langle \mathsf{go}, A_a[T(D) \parallel T(P)] \parallel T(Q)\rangle \parallel A_e[]]$
$\rightarrow \omega_0(U,V)[\eta \parallel A_d[A_b[T(D') \parallel T(P')] \parallel \phi(b)\langle go, A_a[T(D) \parallel T(P) \parallel T(Q)]\rangle] \parallel A_e[]]$
$\rightarrow \omega_0(U,V)[\eta \parallel A_d[A_b[T(D') \parallel T(P') \parallel A_a[T(D) \parallel T(P) \parallel T(Q)]]] \parallel A_e[]]$
$\rightarrow^* \omega_0(U,V)[\eta \parallel A_d[A_b[A_a[]]] \parallel A_e[] \parallel \mathsf{loc}(d.b, D', P') \parallel \mathsf{loc}(d.b.a, D, P \parallel Q)]$
$= \bar{\xi}[T(S)]$

4.3. DISCUSSION

The simulations presented above are very similar in their form: apart from objects performing ancillary functions (global container objects which are used essentially to implement name management in the simulated proces calculus, and gateway objects which are used to implement message exchange with non unique receivers required by the simulated process calculus), the core of each simulation lies in the definition of a particular type of container objects (objects A^u_n and rules (28) to (35) for ambients, objects A_a and rules (50) to (55) for DJoin localities).

This suppports our claim that the essential difference between these distributed calculi lies in the behavior they implicitly attach to their specific notion of domain and which is precisely characterized by the κ model simulation. Additional evidence for this claim can be found in [18].

5. CONCLUSION

We have presented in this paper the κ model, a general model for distributed and mobile computation, based on a notion of domain.

The work undertaken here is far from complete, however. First, two important requirements are currently not covered by our model : the ability to define domains with overlapping contents, and the ability to create new domains by combining existing ones (see [18] for a discussion). To meet these requirements, one would require complementing the κ model with general rules for the combination and the composition of domains.

A second direction for further research resides in the definition of an effective programming model based on the κ model. We are currently investigating the extension of the blue calculus introduced in [3] with constructions based on domains. Armed with such a domain-based process calculus, we would then be in a position to further develop the theory of distributed domains, relying on standard process calculus tools such as bisimulations and type systems.

A third line of investigation lies in the systematic comparison of models and calculi for distributed programming. Apart from the Mobile Ambients calculus and the DJoin calculus, we have successfully simulated the π_{1l}-calculus [1], in the κ model (see [18]). We believe the Seal-calculus, as defined in [22], the Dπ-calculus, as defined in [8], or the Safe Ambients calculus introduced in [11] can be simulated in much the same way. Equipped with a full-fledge domain-based process calculus, we could certainly get stronger results than those reported here, e.g. along the lines of fully abstract simulations up to given execution contexts.

The κ model, as defined in this paper, is certainly too powerful for programming purposes. Studying how different distributed process calculi can be simulated with it, can provide us with useful hints towards the definition of a domain-based process calculus which can subsume them and accommodate them as sub-calculi.

References

[1] R. Amadio : "An asynchronous model of locality, failure, and process mobility" – Research Report RR-3109, INRIA, Sophia-Antipolis, France, February 1997.

[2] G. Berry, G. Boudol : "The chemical abstract machine" – Theoretical Computer Science, vol. 96, 1992.

[3] G. Boudol : "The π-Calculus in Direct Style" – Higher-Order and Symbolic Computation, vol. 11, 1998.

[4] L. Cardelli, A. Gordon : "Mobile Ambients" – Foundations of Software Science and Computational Structures, Maurice Nivat (Ed.), Lecture Notes in Computer Science, Vol. 1378, Springer, 1998.

[5] C. Fournet, G. Gonthier: "The reflexive chemical abstract machine and the join-calculus" – In proceedings 23rd ACM Symposium on Principles of Programming Languages (POPL), January 1996.

[6] C. Fournet, G. Gonthier, J.J. Levy, L. Maranget, D. Remy: "A calculus of mobile agents" – in Proceedings CONCUR '96, LNCS 1119, Springer Verlag, 1996.

[7] C. Fournet, J.J. Levy, A. Schmitt: "A distributed implementation of ambients" – INRIA Research Report, August 1999.

[8] M. Hennessy, J. Riely : "Resource access control in systems of mobile agents" – Technical Report 2/98, School of Cognitive and Computer Sciences, University of Sussex, UK.

[9] ITU-T Recommendation X.902 | ISO/IEC International Standard 10746-2: "ODP Reference Model: Foundations" – November 1995.

[10] ITU-T Recommendation X.903 | ISO/IEC International Standard 10746-3: "ODP Reference Model: Architecture" – November 1995.

[11] F. Levi, D. Sangiorgi: "Controlling Interference in Ambients" – in Proceedings 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000), Boston, MA, USA, Jan. 2000.

[12] J. McAffer: "Meta-Level Architecture Support for Distributed Objects" – Proceedings of Reflection 96, G. Kiczales (ed), San Francisco, USA, April 1996.

[13] R. Milner : "Communicating and mobile systems : the π-calculus" – Cambridge University Press, 1999.

[14] E. Najm, J.B. Stefani: "A formal semantics for the ODP computational model" – Computer Networks and ISDN Systems 27, pp.1305-1329, 1995.

[15] E. Najm, J.B. Stefani: "Computational Models for Open Distributed Systems", – 2nd IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS '97), Canterbury, UK, July 1997.

[16] Object Management Group: "The Common Object request Broker: Architecture and Specification" – CORBA v2.0, July 1995.

[17] Object Management Group: "CORBA Components" – OMG document orbos/99-02-01, March 1999.

[18] J.B. Stefani, F. Germain: "Eléments d'un modèle de traitement pour Marvel" – RNRT project Marvel, Deliverable D2.0, January 2000.

[19] Sun Microsystems: "Java Remote Method Invocation Specification" – Technical Report, Sun Microsystems, Mountain View CA, USA, May 1997.

[20] Sun Microsystems: "Enterprise Java Beans" – Specification v1.0, March 1998.

[21] Sun Microsystems: "Jini Architecture Specification" – Specification v1.0, January 1999.

[22] J. Vitek, G. Castagna : "Towards a calculus of secure mobile computations" – Workshop on Internet Programming Languages, Chicago, Illinois, USA, 1998.

[23] T. Watanabe, A. Yonezawa : "Reflection in an object-oriented concurrent language" – in Proceedings OOPSLA '88, San Diego, California, USA, 1988.