H. Bowman and J. Derrick (eds), Proc. 2nd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems, 1997.

A proof-theoretic approach to the design of object-based mobility

Carlos H. C. Duarte
Department of Computing, Imperial College
180 Queen's Gate, London, SW7 2BZ, United Kingdom
e-mail: `cd7@doc.ic.ac.uk`, tel: `+44 171 594 8341`, fax: `+44 171 581 8024`

Abstract. With the advent of technologies to realise parallel computing in mobile sometimes portable platforms, it is now possible to fulfil requirements related to the very dynamic and mutable user location. Designing the required applications calls for improved formal methods to treat mobility while assuring correctness. In this paper, we argue that mobile systems can be specified and verified in an effective modular manner using a logic which allows us to deal with object creation and reconfiguration. Capitalising on our previous work on the specification and verification of actor systems using a temporal logic of objects, here we show that our approach can be used to formally design location dependent applications.

Keywords: Actors, Specification, Verification, Mobile Systems, Location Management.

1 INTRODUCTION

We are currently facing a radical change in the way users interact with software systems and in the underlying distributed software architectures. Thanks to the advent of technologies like cellular phones, personal digital assistants and active badges, users are no longer required to go to specific access points to take advantage of some locally provided functionality. Such devices have become increasingly more personal and can be carried by their owners. In turn, software systems may now be used at any time and place, and can provide location dependent functionality such as ubiquitous message delivery, transportable user sessions and others [8]. What is essentially novel in this new operational environment is the very presence of mobility. The way to support the new requirements mobility poses is the management of location information.

The need to manage both location information and mobility brings with it new problems to be addressed in the design of parallel/distributed systems. For example, it is now important to develop quantitative models to predict network performance according to user mobility [10]. We are particularly concerned here with the more fundamental question on how to develop such systems so as to correctly provide the required functionality. As complexity substantially grows with the autonomy and heterogeneity presented by mobile devices and mobile applications become more and more open — characteristics that must be accounted for in some design step — introducing errors during the development process turns out to be easier and costly. The only way to assure software correctness is the development of a formal, qualitative model of the system, to be refined in a step-by-step process until an implementation is produced. As a result, it is possible to clarify any matter of concern through proof or refutation.

Unfortunately, most of the existing formal methods are not so good in designing mobile applications. Well established methods like VDM [9] and Z [17] do not address at all the inherent concurrency of mobile systems. In some other cases, concurrency is actually treated but the design process is organized in terms of notions like processes [13] or programs [3, 20], which certainly provide important insights on how an implementation should work but poorly support understanding and representing the problem domain in an organised manner. On the other hand, the use of object-based notions like attributes, actions and encapsulation as in [7] seems to bridge this gap. Even then, expressibility concerns arise since the basic notion of mobility has to be captured.

We have developed a logic to support the design of actor systems [4]. Actors are computational objects with encapsulated state which openly interact through asynchronous point-to-point message passing [1]. As a result of processing messages, new concurrent actors can be created and actor names can be communicated, supporting in this way dynamic reconfiguration of actor communities. Our logic uses temporal theories as object descriptions and theory morphisms as specification connectors particularising [7], although it presents an additional mode of interaction between objects to capture reliable asynchrony. These characteristics make it possible to produce specifications in isolation to be subsequently combined as well as to decompose proofs of global properties in lemmas about single objects to be verified in a localised manner. We claim here that these characteristics are sufficient to guarantee an effective and modular formal design of object-based mobile systems. We are not aware of any similar work in the literature.

In order to support our claim, we have chosen to study here the design of a location management architecture for networks of mobile users and devices. For simplicity, we ignore the important issues of dependability, authenticity and security [18] and concentrate just in the management of location information. In the next section, we introduce our actor-based design approach. Subsequently, we informally describe the requirements of location management applications and devote two sections to their design, namely to their specification and verification. We conclude the paper providing a brief evaluation of our achievements and suggesting further research.

2 A PROOF-THEORETIC APPROACH TO ACTOR DESIGN

We adopt full many-sorted first-order branching time logic with equality as the underlying foundation of our work. A good survey on the subject appears in [5]. Theory signatures and presentations are used to define respectively the language and description of object behaviours and to produce modularised designs. We identify therein the basic object-based notions of attribute, action and encapsulation, and connect these entities using signature and theory morphisms as proposed in [7]. The actor-based formalism is obtained as a particularization of this generic framework. In [4] we provide the rationale leading us to propose the formalism as such and the description of its proof-theory. Here we only present the relevant notions to mobile systems design.

Before introducing technical definitions, let us present in Figure 1 the specification of region tree nodes, instances of the spatial hierarchical data structures described in [16].

Actor REGIONTREENODE

data types $\mathsf{addr}, \mathsf{direc}, \mathsf{int}, \mathsf{bool}$ $(\mathrm{T}, \mathrm{F} : \mathsf{bool}; 0, 1, 3 : \mathsf{int}; + : \mathsf{int} \times \mathsf{int} \to \mathsf{int})$

attributes $me, to : \mathsf{addr}, reg[\mathsf{direc}] : \mathsf{addr}, void : \mathsf{bool}, ans : \mathsf{int}$

actions $cnt(\mathsf{addr}, \mathsf{addr})$: **local birth**;

$node(\mathsf{addr}, \mathsf{addr})$: **local + extrn birth**;

$inc, updt(\mathsf{addr}, \mathsf{addr}, \mathsf{addr}, \mathsf{addr})$: **local computation**;

$ack(\mathsf{addr}, \mathsf{addr}, \mathsf{addr}, \mathsf{addr})$: **extrn message**;

$split(\mathsf{addr}), in(\mathsf{addr}, \mathsf{addr}), rep(\mathsf{addr}, \mathsf{addr}, \mathsf{bool})$: **local + extrn message**

axioms $k, n, p, q, r, s, t, u, x, y, z : \mathsf{addr}, d : \mathsf{direc}, v : \mathsf{int}, b : \mathsf{bool}$

$node(n, p) \vee cnt(n, p) \to me = n \wedge to = p \wedge void = \mathrm{T} \wedge \forall d \cdot reg[d] = n \wedge ans = 0$ (1.2)

$updt(\vec{n}) \wedge me = p \wedge to = q \wedge ans = v \to \mathbf{X}(me = p \wedge to = q \wedge ans = v \wedge void = \mathrm{F} \wedge \vec{reg} = \vec{n})$ (1.3)

$inc \wedge me = n \wedge to = p \wedge \vec{reg} = \vec{q} \wedge ans = v \to \mathbf{X}(me = n \wedge to = p \wedge \vec{reg} = \vec{q} \wedge ans = v + 1)$ (1.4)

$split(n) \wedge me = p \to \mathbf{X}(\exists \vec{q} \cdot \mathbf{new}(node, q_i, q_i, p) \wedge updt(\vec{q}) \wedge \mathbf{send}(ack, n, \vec{q}))$ (1.5)

$(in(n, p) \wedge r = n \vee rep(r, s, \mathrm{T}) \wedge to = p) \wedge me = n \to \mathbf{X}(\mathbf{send}(rep, p, r, n, \mathrm{T}))$ (1.6)

$in(n, p) \wedge me = q \wedge q \neq n \wedge void = \mathrm{F} \wedge \vec{reg} = \vec{r} \to \mathbf{X}(\exists! \, s \cdot \mathbf{new}(cnt, s, q, p) \wedge \mathbf{send}(in, r_i, n, s))$ (1.7)

$in(n, p) \wedge me = q \wedge q \neq n \wedge void = \mathrm{T} \to \mathbf{X}(\mathbf{send}(rep, p, n, q, \mathrm{F}))$ (1.8)

$rep(n, p, \mathrm{F}) \wedge to = r \wedge me = q \to \mathbf{X}(ans = 3 \wedge \mathbf{send}(rep, r, n, q, \mathrm{F}) \vee inc)$ (1.9)

$\exists \vec{n}, \vec{p} \cdot \mathbf{new}(node, n_i, p_i, q) \vee updt(\vec{n}) \vee \mathbf{send}(ack, r, \vec{n}) \leftarrow split(r) \wedge me = q$ (1.10)

$inc \leftarrow \exists n, p \cdot rep(n, p, \mathrm{F}) \wedge ans \neq 3$ (1.11)

$\mathbf{send}(rep, n, p, q, \mathrm{T}) \leftarrow (in(q, n) \wedge p = q \vee \exists s \cdot rep(p, s, \mathrm{T}) \wedge to = n) \wedge me = q$ (1.12)

$\mathbf{send}(rep, n, p, q, \mathrm{F}) \leftarrow (in(p, n) \wedge p \neq q \wedge void = \mathrm{T} \vee \exists s \cdot rep(p, s, \mathrm{F}) \wedge to = n \wedge ans = 3) \wedge me = q$ (1.13)

$\exists n \cdot \mathbf{new}(cnt, n, q, r) \vee \mathbf{send}(in, p_i, s, n) \leftarrow in(s, r) \wedge me = q \wedge q \neq s \wedge void = \mathrm{F} \wedge \vec{reg} = \vec{p}$ (1.14)

$node(k, n) \vee cnt(k, n) \to \mathbf{G}(\mathbf{E}(\mathbf{deliv}(split(p))) \wedge \mathbf{E}(\mathbf{deliv}(in(q, r))) \wedge \mathbf{E}(\mathbf{deliv}(rep(s, t, b))))$ (1.15)

$node(k, n) \vee cnt(k, n) \to \mathbf{XG}(inc \vee updt(p, q, r, s) \vee \mathbf{E}(split(t)) \wedge \mathbf{E}(in(u, x)) \wedge \mathbf{E}(rep(y, z, b)))$ (1.16)

End

Figure 1 Specification of region trees.

These will be used in designing location management later on. At the top of each theory presentation, we can see sequences of sorts plus the associated constants and operations (data types), attribute and action symbols, denoting respectively fixed meaning data objects, the local state and the messages and computations dealt with by these actors. For instance, a region can receive a request to divide itself (*split*) in sub-regions (*reg*) organised according to the four directions of the compass points (direc). Eventually, continuations (*cnt*) are created in order to answer inclusion queries (*in*). These symbols belong the language of region tree nodes and are generically formalised as follows:

Definition 1 (Actor Signature) An actor signature Δ is a triple of disjoint and finite families $(\Sigma, \mathcal{A}, \Gamma)$ where:

- $\Sigma = (S, \Omega)$ is an universe signature, i.e., S is a set of sort symbols and Ω is an $S^* \times S$-indexed family of operation symbols. We require that the sort of mail addresses $\mathsf{addr} \in S$;
- $\mathcal{A}$ is an $S^* \times S$-indexed family of attribute symbols;
- $\Gamma = (\Gamma_e, \Gamma_l, \Gamma_c)$, S^*-indexed sets of action symbols with $(\Gamma_e \cup \Gamma_l) \cap \Gamma_c$ empty. Γ_c is a set of local computation symbols. Γ_e and Γ_l represent respectively sets of events to be requested from the environment and provided locally. These two sets consist in collections of message and birth computation symbols, e.g. $\Gamma_l - \Gamma_{l_b}$ and Γ_{l_b} respectively.

For ϵ as the empty sequence, we write an $\epsilon \times s$-indexed family of symbols as if s were its index. Given a set X, we denote the sub-set of X symbols of sort $\langle s_1, \ldots, s_n \rangle \times s$ as $X_{\langle s_1, \ldots, s_n \rangle, s}$. We shall also operate with subscripts ($\Gamma_{e_b \cap l_b}$) to denote operations on sub-sets of Γ ($\Gamma_{e_b} \cap \Gamma_{l_b}$).

Axioms defined out of terms and formulae are also present in the previous specification. An example is that asserting both the invariance of the attributes holding the mail addresses of the node and its parent (terms in 1.2) and the creation of sub-regions in the instant following an update (*updt*). Likewise, we use a formula to say that after the birth of a node requests for *split*, *in* or *rep* may always be delivered (1.14), although they may only be consumed if the node is not busy sequentially performing local computations (1.15). To define these notions, we assume that an infinite family of rigid variables and its classification Ξ according to the sorts of a signature Δ are given:

Definition 2 (Terms) The S-indexed set of terms $T_\Delta(\Xi)$ is defined as follows, provided that $q \in \Xi_s \cup \Omega_s \cup \mathcal{A}_s$, $p \in \Omega_{\langle s_1,\ldots,s_n \rangle, s}$, $f \in \mathcal{A}_{\langle s_1,\ldots,s_n \rangle, s}$ and $t_i \in T_\Delta(\Xi)_{s_i}$:

$$t ::= q \mid p(t_1, \ldots, t_n) \mid f(t_1, \ldots, t_n)$$

Definition 3 (Formulae) The set $F_\Delta(\Xi)$ of formulae is defined by the mutual recursion below, provided that $c \in \Gamma_{\langle s_1,\ldots,s_n \rangle}$, $t_i \in T_\Delta(\Xi)_{s_i}$, $y \in \Xi_s$ and $g_i \in F_\Delta(\Xi)$:

$$\begin{aligned} g &::= \mathbf{beg} \mid c(t_1, \ldots t_n) \mid t_1 =_s t_2 \mid \mathbf{E}g' \mid g_1 \to g_2 \mid \neg g_1 \mid \exists y \cdot g_1 \\ g' &::= g \mid \mathbf{X}g'_1 \mid g'_1 \mathbf{U} g'_2 \mid g'_1 \to g'_2 \mid \neg g'_1 \mid \exists y \cdot g'_1 \end{aligned}$$

Terms consist in variables, nulary function and attribute symbols; or function and attribute symbols applied to terms. We usually write a sequence of similar terms $t_1, \ldots, t_n$ as $\vec{t}$. Formulae stand for the initial instant (**beg**); action occurrences; term equality; the occurrence of a property in some possible behaviour (**E**), in the next instant (**X**) or until another property holds (**U**); or formulae aggregation using first-order logic connectives. These are the original CTL* constructs [5] enriched to express object-based notions.

The reader may wonder why **new**, **deliv** and **send** do not appear in our definitions above. Actually, they stand for the abbreviation of logical actions as defined in [4]. Much in the way that ASCCS is a subset of SCCS [13], our calculus — which captures synchronous object creation and reliable asynchronous message passing — can be seen as a particularization of the synchronous object calculus of [7]. The aforementioned connectives are definable therein and have the following informal meaning:

For	of type	formula	reads
$c, n, \vec{v_c}$	$\Gamma_{e_b \cup l_b}, \mathsf{addr}, T_\Delta(\Xi)$	$\mathbf{new}(c, n, \vec{v_c})$	creation of an actor with a given name
$c, n, \vec{v_c}$	$\Gamma_{(e-e_b)\cup(l-l_b)}, \mathsf{addr}, T_\Delta(\Xi)$	$\mathbf{send}(c, n, \vec{v_c})$	dispatch of a message to a specific actor
$c, \vec{v_c}$	$\Gamma_{l-l_b}, T_\Delta(\Xi)$	$\mathbf{deliv}(c, \vec{v_c})$	delivery of a message

As for the actor primitive **become**, which allows actors to have mutable state space, there is no treatment here. If it does not receive a higher-order interpretation, this primitive is definable within the core actor theory as noticed in [1]. In designing mobile systems, such a higher-orderness is not required.

Specifications characterise communities of actors with similar behaviour. What make these actors different from each other are their distinct names (of sort addr, prefixed to terms when talking about global properties) and their potentially distinct interactions with the environment. Specifications are defined as theory presentations comprising a signature and a set of axioms explicitly provided by the specifier:

Definition 4 (Actor Specification) An actor specification is a pair $\Phi = (\Delta, \Psi)$ where Δ is an actor signature and Ψ is a finite set of formulae over Δ (the specification axioms).

To each specification Φ is assigned a set of additional logical axioms called Ax_Φ. These axioms are provided in schematic form in the Appendix and constrain the behaviour of the environment relative to the specified actors. Such axioms become necessary, together with the deductive system of the branching time temporal logic of objects and the additional inference rules of the Appendix, in the verification of actor properties.

For notational convenience, formulae containing derived first-order logic connectives and inequalities stand for the usual translations. Moreover, free variables in axioms are considered to be universally quantified. Other admissible connectives are defined as:

FOR	OF TYPE	FORMULA	READS	REPRESENTS
g	$F_\Delta(\Xi)$	$\mathbf{A}g$	in any behaviour	$\neg\mathbf{E}\neg g$
g	$F_\Delta(\Xi)$	$\mathbf{F}g$	eventually in the future	$(g \to g)\mathbf{U}g$
g	$F_\Delta(\Xi)$	$\mathbf{G}g$	always in the future	$\neg\mathbf{F}\neg g$
g_1, g_2	$F_\Delta(\Xi)$	$g_1\mathbf{W}g_2$	weak until	$\mathbf{G}g_1 \vee g_1\mathbf{U}g_2$
g_1, g_2, p	$F_\Delta(\Xi)$	$g_1\overset{i}{\leftarrow}_p g_2$	initially precedes	$p \to (\neg g_1)\mathbf{W}(g_2 \wedge \neg g_1)$
g_1, g_2, p	$F_\Delta(\Xi)$	$g_1 \leftarrow_p g_2$	precedes	$g_1\overset{i}{\leftarrow}_p g_2 \wedge g_1 \to \mathbf{X}(g_1\overset{i}{\leftarrow}_{(p\to p)} g_2)$

The unary connectives above are non-strict (they include the present) and usually appear in branching-time logics. Conversely, the precedence connectives are strict and forbid the simultaneous occurrence of some properties. In specifications, where p usually stands for the occurrence of the initial instant (which we write as **beg**), their index is omitted. All these temporal connectives are used, e.g., to state and reason about causality relations: that a query *in* for the inclusion of a node n in a region, when consumed by any non-empty node distinct from n, in the next instant results not only in the dispatch of many similar queries to the respective sub-regions but also in the creation of a continuation actor to process their answers (1.6), something that cannot happen otherwise (1.13).

Given some independently specified actor communities, we may want to interconnect them to define communities of heterogeneous cooperating actors. This can be done by providing language translations between their theory presentations obeying what follows:

Definition 5 (Signature Morphisms) Given two actor signatures $\Delta_1 = (\Sigma_1, \mathcal{A}_1, \Gamma_1)$ and $\Delta_2 = (\Sigma_2, \mathcal{A}_2, \Gamma_2)$, a signature morphism $\tau : \Delta_1 \to \Delta_2$ consists of:

- a morphism of algebraic structures $\tau_\upsilon : \Sigma_1 \to \Sigma_2$ such that $\tau_\upsilon(\mathsf{addr}_1) = \mathsf{addr}_2$;
- for each $f \in \mathcal{A}1_{\langle s_1,\ldots,s_n\rangle,s}$, an attribute symbol $\tau_\alpha(f) : \tau_\upsilon(s_1) \times \ldots \times \tau_\upsilon(s_n) \to \tau_\upsilon(s)$ in $\mathcal{A}_2$;
- for each $c \in \Gamma1_{\langle s_1,\ldots,s_n\rangle}$, an action symbol $\tau_\gamma(c) : \tau_\upsilon(s_1) \times \ldots \times \tau_\upsilon(s_n)$ in Γ_2 such that $\tau_\gamma(\Gamma_{e1}) \subseteq \Gamma_{e2}$, $\tau_\gamma(\Gamma_{l_1}) \subseteq \Gamma_{l_2}$, $\tau_\gamma(\Gamma_{c1}) \subseteq \Gamma_{c2}$, where $\tau_\gamma(\Gamma_{e_{b_1}}) \subseteq \Gamma_{e_{b_2} \cup l_{b_2}}$ and $\tau_\gamma(\Gamma_{e1-e_{b_1}}) \subseteq \Gamma_{(e_2-e_{b_2})\cup(l_2-l_{b_2})}$, $\tau_\gamma(\Gamma_{l_{b_1}}) \subseteq \Gamma_{l_{b_2}}$ and $\tau_\gamma(\Gamma_{l_1-l_{b_1}}) \subseteq \Gamma_{l_2-l_{b_2}}$ so that $\tau_\gamma(\Gamma_{e_1\cap l_1}) \subseteq \Gamma_{e_2\cap l_2}$.

It is straightforward to define inductively the translation of symbols, classifications, terms, formulae and sets thereof under τ.

From the first item, we can see that interconnected actor communities are named using the same sort, namely addr. In addition, the third item says that event symbols representing requests of a community to its environment can be associated with events internally provided by a distinct community (eg. $\tau_\gamma(\Gamma_{e_{b_1}}) \subseteq \Gamma_{e_{b_2}\cap l_{b_2}}$), meaning that messages can be dispatched to and actors created within one community from another.

Technically, the translation of connected theory presentations induced by signature morphisms does not capture the expected interconnection of actor behavior in a precise way. This is due to the existing logical axioms of the original theories, which are not translated by such morphisms. We are obliged to use an alternative notion:

Definition 6 (Theory or Specification Morphisms) Given two actor specifications $\Phi_1 = (\Delta_1, \Psi_1)$ and $\Phi_2 = (\Delta_2, \Psi_2)$, a specification morphism $\tau : \Phi_1 \to \Phi_2$ is a signature morphism such that $\vdash_{\Phi_2} \tau(g)$ for every $g \in \Psi_1 \cup Ax_{\Phi_1}$.

We should stress that connecting specifications using theory morphisms is analogous to providing links between identifiers in distinct actor programs [1]. As we shall see later, to verify actor properties, we also need to identify which actors are assumed to exist in the environment and which are able to receive messages from the outside, as in [2].

3 INFORMAL REQUIREMENTS OF LOCATION MANAGEMENT

A central problem in designing and implementing software systems for networks of mobile users and devices is how to manage object location. An extensive description of the problem can be found in the literature (cf. [8, 12, 18]). In this section, we provide an informal list of requirements strictly imposed by mobility. In the next section, we enumerate some design decisions based on this list and propose a formal specification for the corresponding mobile architecture.

We can classify the requirements of location management in three families, the first concerning the nature of location information and located objects, the second about the process of acquiring location information and the third on how to deal with it. In what follows, we ignore real time issues and provide a partial list of functional requirements:

1. A *location information* must be *dynamic*, in the sense that, at each time, it may be a distinct instance of a class of objects.

2. A *location information* must be *mutable*, in the sense that, at each time, it may be an instance of a distinct class of objects.

3. *Located objects* may be *users* or *devices*, at least.

4. *Location information acquisition* must be *unintrusive*, which means that the acquisition process cannot intrude user behaviour nor require user intervention.

5. *Location information acquisition* must offer support to *multiple location observations*, which means that simultaneous observations producing distinct location information for the same object may occur.

6. *Location information management* must support *indeterminacy*, which means that location information for some objects may not be available at some instant.

7. *Location information management* must offer support to *object naming*, which is the assignment of meaningless unique names to located objects.

The first two items should not be confused. While it is obvious that mobile object locations may change as time passes, meaning that they are dynamic, it is not so obvious that they should also be mutable. The reason for this is that a location service may provide information with distinct accuracies or that multiple services may be used [12].

Actor Sensor

data types addr, int $(0, 1, \text{MAX} : \mathsf{int}; + : \mathsf{int} \times \mathsf{int} \to \mathsf{int})$

attributes $srv, obj, loc, id : \mathsf{addr}; tick : \mathsf{int}$

actions $sens(\mathsf{addr}, \mathsf{addr}, \mathsf{addr}, \mathsf{addr})$: **local** + **extrn birth**;

$reloc(\mathsf{addr}), set(\mathsf{int}), obs$: **local computation**;

$detect(\mathsf{addr}, \mathsf{addr}), unreach(\mathsf{addr}, \mathsf{addr})$: **extrn message**

axioms $n, p, q, r : \mathsf{addr}, v : \mathsf{int}$

$$sens(n,p,q,r) \to srv = n \wedge obj = p \wedge loc = q \wedge id = r \wedge tick = 0 \tag{2.2}$$

$$reloc(n) \wedge srv = p \wedge obj = q \wedge id = r \wedge tick = v \to \mathbf{X}(srv = p \wedge obj = q \wedge id = r \wedge tick = v) \tag{2.3}$$

$$reloc(n) \to \mathbf{X}(loc = n) \tag{2.4}$$

$$set(v) \wedge srv = n \wedge obj = p \wedge loc = q \wedge id = r \to \mathbf{X}(srv = n \wedge obj = p \wedge loc = q \wedge id = r) \tag{2.5}$$

$$set(v) \to \mathbf{X}(tick = v) \tag{2.6}$$

$$obs \wedge srv = n \wedge loc = p \wedge (obj = q \vee id = q) \to \mathbf{X}(set(0) \wedge \mathbf{send}(detect, n, q, p)) \tag{2.7}$$

$$sens(n,p,q,r) \to \mathbf{G}(\neg obs \wedge tick = \text{MAX} \leftrightarrow \mathbf{send}(unreach, srv, obj, loc)) \tag{2.8}$$

$$sens(n,p,q,r) \to \mathbf{G}(\neg obs \wedge tick = \text{MAX} \to set(0)) \tag{2.9}$$

$$sens(n,p,q,r) \to \mathbf{G}(\neg set(0) \wedge \neg obs \leftrightarrow set(tick+1) \wedge \mathbf{send}(detect, srv, id, loc)) \tag{2.10}$$

$$\mathbf{send}(detect, n, p, q) \to src = n \wedge loc = q \wedge (id = p \wedge set(v) \vee obj = p \wedge set(0)) \tag{2.11}$$

$$set(0) \leftarrow obs \vee tick + 1 = \text{MAX} \tag{2.12}$$

End

Figure 2 Specification of sensors.

4 LOCATION MANAGEMENT DESIGN IN A FORMAL SETTING

Based on the previous requirements list, we make our first design decision following [8] by using references to objects denoting spatial regions instead of dealing with location information directly. In this way, each located object acquires a new attribute (*loc*), which is annotated with the mail address of an object representing a location space region. If we use region trees as described in Section 2 for this purpose, we treat both the dynamic and mutable character of location information with this decision: as an attribute, location information can always be changed; as a reference, it does not constrain the shape and size of location observations. We make, however, the simplifying assumption that spatial regions are disjoint squares, due to the structure of such trees.

In order to treat the requirements concerning location information acquisition and management, we adopt the specification of sensors in Figure 2. Each sensor is created with knowledge of a location service mail address (*srv*) and is responsible for producing sequential observations (*obs*) of a named user (*obj*) in the specific region. Sensors are mobile as well and detect themselves in the monitored region (2.7 and 2.9). We omit their straightforward generalisation to deal with the observation of several distinct objects.

Each sensor keeps an internal clock which is reset — *set*(0) — after MAX cycles or when the user is observed. Axiom 2.11 guarantees that resets do not happen unless this condition is fulfilled. Indeterminacy is treated by this clocking mechanism, which signs to the location service that the user is unreachable (*unreach*) whenever an observation does not happen before the deadline MAX (2.7). A *detect* message with the user location is sent to the service otherwise (2.6). Multiple observations are obtained by having many sensors dealing with the same located object. Unintrusivity is also treated as there is no causal connection between the production of observations and user behaviour.

Actor MOBILEAGENT
  data types addr, bool (T, F : bool)
  attributes me, id, loc, to : addr, $fwdg$: bool
  actions ag(addr, addr, addr) : **local** + **extrn birth**;
      $redir$(addr) : **local computation**;
      sub(addr, addr) : **extrn message**;
      fwd(addr), mv(addr, addr), cp(addr, addr, addr) : **local** + **extrn message**
  axioms $n, p, q, r, s, t, u, x, y, z$: addr

$$ag(n,p,q) \to me = n \wedge id = p \wedge loc = q \wedge fwdg = \text{F} \wedge to = n \tag{3.2}$$
$$redir(n) \wedge me = p \wedge id = q \wedge loc = r \to \mathbf{X}(me = p \wedge id = q \wedge loc = r \wedge fwdg = \text{T} \wedge to = n) \tag{3.3}$$
$$fwd(n) \to \mathbf{X}(redir(n)) \tag{3.4}$$
$$mv(n,p) \wedge fwdg = \text{F} \wedge me = q \wedge id = r \to \mathbf{X}(redir(q) \wedge \mathbf{send}(cp, n, p, q, r)) \tag{3.5}$$
$$mv(n,p) \wedge fwdg = \text{T} \wedge to = q \to \mathbf{X}(\mathbf{send}(mv, q, n, p)) \tag{3.6}$$
$$cp(n,p,q) \wedge loc = r \to \mathbf{X}(\exists!\, s \cdot \mathbf{new}(ag, s, s, q, r) \wedge \mathbf{send}(fwd, p, s) \wedge \mathbf{send}(sub, n, s, r)) \tag{3.7}$$
$$redir(n) \leftarrow fwd(n) \vee \exists p, q \cdot (mv(p,q) \wedge me = n \wedge fwdg = \text{F}) \tag{3.8}$$
$$\exists n, p \cdot \mathbf{new}(ag, n, p, q, r) \vee \mathbf{send}(fwd, s, n) \vee \mathbf{send}(sub, t, n) \leftarrow cp(t, s, q) \wedge loc = r \tag{3.9}$$
$$\mathbf{send}(mv, n, p, q) \leftarrow mv(p,q) \wedge to = n \wedge fwdg = \text{T} \tag{3.10}$$
$$\mathbf{send}(cp, n, p, q, r) \leftarrow mv(n,p) \wedge me = q \wedge id = r \wedge fwdg = \text{F} \tag{3.11}$$
$$ag(n,p,q) \to \mathbf{G}(\mathbf{E}(\mathbf{deliv}(fwd, s)) \wedge \mathbf{E}(\mathbf{deliv}(mv, t, u)) \wedge \mathbf{E}(\mathbf{deliv}(cp, x, y, z))) \tag{3.12}$$
$$ag(n,p,q) \to \mathbf{XG}(\neg redir(r) \to \mathbf{E}(fwd(s)) \wedge \mathbf{E}(mv(t,u)) \wedge \mathbf{E}(cp(x,y,z))) \tag{3.13}$$
End

Figure 3 Simplified specification of mobile agents.

If we realise the sensors of Figure 2 as optical devices connected to the architecture through radio frequency links, for instance, software mobility arises only when located object agents are considered. Such agents are meant to follow located objects through the architecture providing location dependent functionality such as ubiquitous message delivery and transportable user sessions [18]. Although we leave this additional functionality unspecified here, we present a specification of mobile agents in Figure 3.

We choose to capture mobility as localised agent replication. A mobile agent may receive a request to move to the location of another agent (mv), presumably located closer to the object the former represents. If an agent is currently moving to a new location, such requests will be delayed by self-forwarding until the agent finishes to move (3.5). In order to move, the original agent issues a request for the correctly located agent to create a local copy (cp) of its own (3.4), supplying in the message any required information for the copy (here in particular its name). After consuming this kind of message, an agent creates the desired replica and notifies both the original agent and the requesting service that the located object representative can be substituted (3.6).

In order to ensure coordination between sensors and agents, a location service must guarantee that the asynchronous messages they exchange are correctly addressed and ordered. The situation is better explained by the diagram in Figure 4. Once a located object is detected in a new region (at), the location service has to find a mobile agent in that region to request the creation of a replica of the moving agent there. The location space is recurrently queried (in) until such an agent is found. Then, the service requests the agent of the relocated object to move the place of the correctly located agent ($move$). At the end, the service is notified (sub) so that the old agent can be discarded while

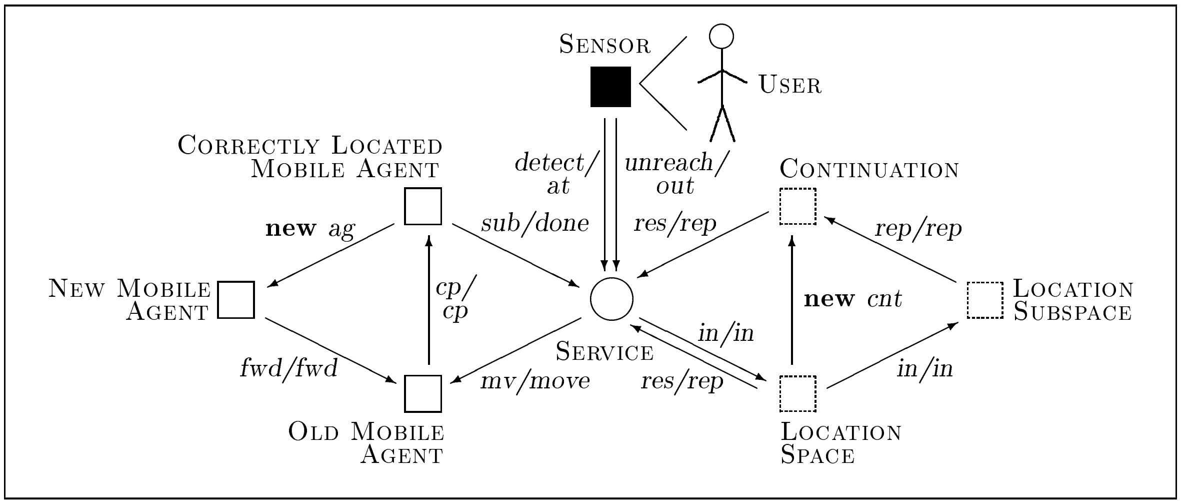

Figure 4 Internal event flow of the Location Manamegent Architecture.

forwarding all possibly dispatched additional move requests to the new agent (*fwd*).

Since the location service has to associate located object names (*id*) with mobile agents, has to keep track of their location (*loc*) and has to put agents in contact to support mobility, we consider that servers providing compartmentalised bits of this functionality for each object are organised in circular lists, adopting the specification in Figure 5. Each server also records if there is no location information available for the object (*nl*). This knowledge is used to postpone answering location queries (?) until the object location becomes known (4.7 and 4.8).

Every request to the location service circulates around the linked list until the correct recipient is found. In case an observation from a sensor arrives carrying a new object location (*at*), a request for the rest of the list to find some agent placed therein is issued aiming to support moving to that location (4.9). For each located object registered in the service, the location space will be queried in a two step process: a continuation actor to process the answer of the query will be created (4.13), and this new actor will either request the original agent to move (4.14), if the current agent is located accordingly, or forward the query to the next list element (4.15).

Although illustrative, the informal description of the relationship between each pair of specifications should not substitute their formal composition, which is still missing here. The diagram in Figure 4 gives a good clue on what remains to be defined: the “physical communication channels”, which are formally defined using specification morphisms. For every pair of specifications, each of them represented by a distinguished geometric figure, that diagram shows how to relate their message symbols. For instance, the messages *mv* and *sub* of agents should be respectively associated with *move* and *done* of servers. Notice that relating external to local symbols yields the only possible direction of the message flow. Also notice in our example that we cannot produce a direct translation either from the theory of agents into that of servers or in the opposite direction. Therefore, to interconnect these entities we need to define mediating theory presentations to serve as

Actor SERVER
  data types addr, bool (T, F : bool)
  attributes me, nxt, loc, id, ag : addr, nl : bool
  actions srv(addr, addr, addr, addr, addr) : **local** + **extrn birth**;
    ch(addr, addr, addr, bool) : **local computation**;
    in(addr, addr), $move$(addr), @(addr, addr), ack(addr) : **extrn message**;
    ins(addr, addr, addr, addr), $done$(addr, addr), res(addr, addr, bool) : **local** + **extrn message**;
    at(addr, addr), out(addr, addr) : **local** + **extrn message**;
    $mvrq$(addr, addr, addr), ?(addr, addr) : **local** + **extrn message**
  axioms n, p, q, r, s, t, u, x : addr, b : bool

$$srv(n,p,q,r,s) \rightarrow me = n \wedge nxt = p \wedge id = q \wedge loc = r \wedge ag = s \wedge nl = \mathrm{F} \quad (4.2)$$
$$ch(n,p,q,b) \wedge me = r \wedge id = s \wedge nl = b \rightarrow \mathbf{X}(me = r \wedge id = s \wedge nl = b) \quad (4.3)$$
$$ch(n,p,q,b) \rightarrow \mathbf{X}(nxt = n \wedge loc = p \wedge ag = qnl = b) \quad (4.4)$$
$$ins(n,p,q,r) \wedge nxt = s \wedge loc = t \wedge ag = u \wedge nl = b \rightarrow \mathbf{X}(\exists x \cdot \mathbf{new}(srv, x, x, s, n, p, q) \wedge ch(x, t, u, b)) \quad (4.5)$$
$$ins(n,p,q,r) \wedge nxt = s \rightarrow \mathbf{X}(\exists t \cdot \mathbf{new}(srv, t, t, s, n, p, q) \wedge \mathbf{send}(ack, r, t)) \quad (4.6)$$
$$done(n,p) \wedge nxt = q \rightarrow \mathbf{X}(ch(q, p, n, \mathrm{F})) \quad (4.7)$$
$$?(n,p) \wedge id = n \wedge me = q \wedge loc = r \rightarrow \mathbf{X}(nl = \mathrm{F} \wedge \mathbf{send}(@, p, n, r) \vee \mathbf{send}(?, q, n, p)) \quad (4.8)$$
$$?(n,p) \wedge id \neq n \wedge nxt = q \rightarrow \mathbf{X}(\mathbf{send}(?, q, n, p)) \quad (4.9)$$
$$at(n,p) \wedge id = n \wedge me = q \wedge nxt = r \wedge ag = s \wedge (loc \neq p \vee nl = \mathrm{T}) \rightarrow \mathbf{X}(\mathbf{send}(mvrq, r, s, p, q)) \quad (4.10)$$
$$at(n,p) \wedge id \neq n \wedge nxt = q \rightarrow \mathbf{X}(\mathbf{send}(at, q, n, p)) \quad (4.11)$$
$$out(n,p) \wedge id = n \wedge nxt = q \wedge loc = r \wedge ag = s \rightarrow \mathbf{X}(ch(q, r, s, \mathrm{T})) \quad (4.12)$$
$$out(n,p) \wedge id \neq n \wedge nxt = q \rightarrow \mathbf{X}(\mathbf{send}(out, q, n, p)) \quad (4.13)$$
$$mvrq(n,p,q) \wedge nxt = r \wedge loc = s \wedge ag = t \rightarrow \mathbf{X}(\exists! u \cdot \mathbf{new}(srv, u, q, r, p, n, t) \wedge \mathbf{send}(in, s, p, u)) \quad (4.14)$$
$$res(n,p,\mathrm{T}) \wedge me = q \wedge id = r \wedge ag = s \rightarrow \mathbf{X}(\mathbf{send}(move, r, s, q)) \quad (4.15)$$
$$res(n,p,\mathrm{F}) \wedge me = r \wedge nxt = s \wedge loc = t \wedge id = u \rightarrow \mathbf{X}(\mathbf{send}(mvrq, s, u, t, r)) \quad (4.16)$$

⋮
and the usual axioms to guarantee absence of unsolicited responses and enableness
⋮
End

Figure 5 Specification of location service nodes.

connectors. Their nature is illustrated by the diagram in Figure 6.

To define the mobile architecture in a formal manner, we call each mediating specification in Figure (6.a) a CONNECTOR. Each of them contains two external message symbols only (without axioms as well) and hence translations including their contents after necessary renamings into the connected theories trivially exist. Taking connectors, connected theories and the morphisms betweem them, the composite theory presentations are defined up to isomorphism by categorical constructions called pushouts (amalgamed sums), which always exist and are finite for any finite number of actor specifications [4]. Defined in this way, each COMPONENT in the figure contains all the renamed symbols and axioms of the connected theories, but the symbols identified by the connectors are equalised. That is why, e.g., a message *move* from servers can be understood as *mv* when it is dispatched to an agent, no matter its name in the composite component. The detailed definition of connectors and their morphisms appears in Figure (6.b).

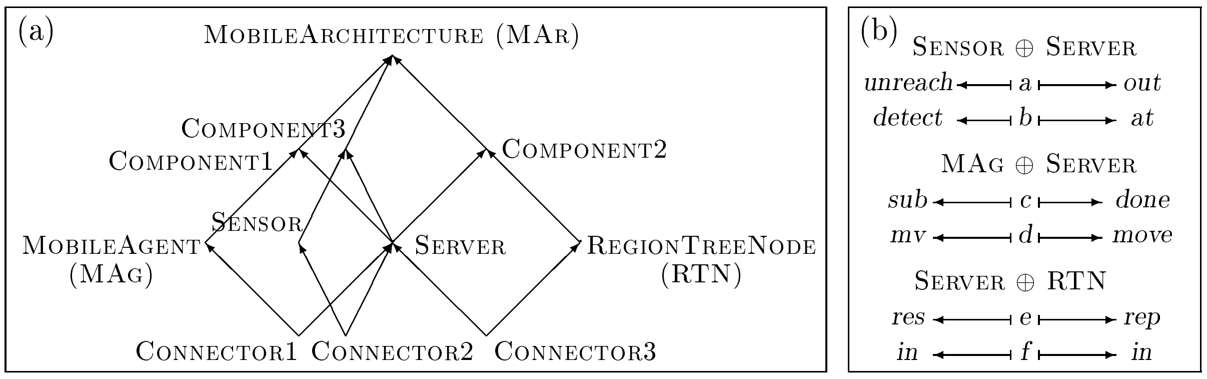

Figure 6 Composition of the architecture: Shared actors (a) and action symbols (b).

5 VERIFYING LOCATION MANAGEMENT PROPERTIES

As we have already mentioned, the existence of morphisms to connect separated actor specifications does not guarantee that interaction between actors in the respective communities can occur. Here, an analogy with telecommunication systems is useful to illustrate the situation: even if the physical cables to connect the private equipment of a customer to the network exist, it cannot receive phone calls unless assigned to an appropriate number. Therefore, "logical channels" are also needed in dealing with any kind of global property. These logical channels are captured here as an assumption on the configuration of the heterogeneous actor community.

Since the location space is a relatively separated component of our architecture, let us use it to exemplify how a property can be verified. Assume that the environment always creates region tree nodes configured correctly, providing the right name for the node and its parent (which is the node itself in the case of the root) in the creation:

$$r.\mathbf{new}(reg, s, t, u) \to s = t \land (u = r \lor u = s) \tag{1}$$

The main functionality provided by the location space is the support to queries. Hence, every node s should eventually answer queries addressed to it:

$$r.\mathbf{send}(in, s, t, u) \to \mathbf{F}(\exists x, v \cdot x.\mathbf{send}(rep, r, s, t, v)) \tag{2}$$

The usual procedure for proving interaction properties of actor communities is: (a) find a local invariant of the recipient; (b) show that the invariant guarantees that this actor will eventually become enabled for delivery/consumption of the message; (c) prove that the dispatch eventually leads to the message delivery and this guarantees the consumption, which in turn produces the outcome of the interaction. The reader should notice that these steps correspond to the application of the rules *COM* followed by *RESP*, both described in the Appendix, which capture the fairness requirements of reliable message passing and finite consumption delay present in the Actors model [1].

Returning to the location space example, it is not difficult to see that the invariant of each region tree node is:

$$Inv \equiv (\forall d \cdot reg[d] = me \lor void = \mathrm{F}) \land (0 \leq ans \leq 3) \tag{3}$$

To prove this, first observe that Inv is a precondition for the occurrence of birth actions of RTN, *node* and *cnt*, in Axiom 1.1. Moreover, each local computation preserves this property, meaning that occurrences of *updt* do not change the logical value of the first conjunct while keeping ans unmodified, according to 1.2, and *inc* is only allowed to happen if the value of ans remains in the interval [0,3], from Axioms 1.3 and 1.10. An application of rule $SAFE$, also described in the Appendix, shows that Inv always holds.

Now, suppose that s, the recipient, is an empty node. Let p as in rule $RESP$ be $Inv \vee \mathbf{init}$. Since Inv is invariant, its disjunction with **init** is preserved by every local computation of the actor and therefore the first premise of that rule holds. In passing, notice that *updt* and *inc* cannot continuously happen since they are causally connected to the occurrence of *split* or *rep* and the axiom scheme **10.** in the Appendix says that such actions can only happen one at a time. From Axiom 1.15, it is clear that p guarantees the eventual enableness for query consumption, thus the third premise in the aforementioned rule also holds. Making q in the same rule be the reply, we can see from Axioms 1.6 and 1.8 that it will eventually happen. A similar but simpler rationale can be used to show through the application of COM that once the query is dispatched, it will eventually be delivered. Chaining these results, we conclude a step in the verification of (2).

To complete the verification of (2), consider now those cases where the cell has already been split, and so answering a query may require creating auxiliary continuation actors. Verifying such cases of chained interaction requires the following derived rule [5]:

$$\boxed{[WELL] \quad \dfrac{1. \quad P(x) \to \mathbf{F}(Q \vee \exists y \cdot (y \prec x \wedge P(y)))}{(\exists x \cdot P(x)) \to \mathbf{F}Q}} \qquad \text{where } \prec \text{ is a well-founded relation}$$

We usually take $P(x)$ as a message dispatch by an actor with mail address x and Q as the outcome of its consumption. To prove the non-trivial case of (2), we need to apply such rule twice, one to show that the query eventually reaches the correct region or that all the location space terminal nodes are queried without matching the parameter of the message, and the other to show that the reply will proceed through the created continuations until it reaches the original actor. For the second case, e.g., the required well-founded relation can ben taken as $R_p(x, y) \overset{\text{def}}{=} (y.to = x) \wedge (y \neq p)$. The anti-reflexivity of R_p derives from the use of Axiom 1.6 in rule $EXIT$, which shows that a node is prevented from being a continuation of itself, and since *to* is invariant, R_p indeed defines a well-founded relation. Therefore, we can apply $WELL$ and complete the proof.

Although we have omitted this detail in the informal proof above, the assumption (1) is fundamental to ensure that the answer to the query will be addressed to the correct actor, be it dispatched by the recipient itself or by a continuation actor. It is also important to mention that, formally, queries are not expected to come from within the community of region tree nodes, but from that of sensors. Due to the morphisms connecting these communities to each other, the complex properties of the system can be decomposed and translated into lemmas to be proved in a localised manner, using the axioms of just one theory presentation.

In a more realistic context, a reasonable assumption for our architecture would consider for instance that for every region there is a named server associated to an agent and dissociated from any sensor, which could be regarded as the meta-level actors required to exist in order to support resource management activities [19], in our case the access to some location. The formal verification of properties of our architecture considering these assumptions appears elsewhere. Of course, (2) could be (re)used there and other properties would be verified almost in the same manner.

6 FINAL REMARKS

In this paper, we have presented an approach to the design of object-based mobile systems using a temporal logic specially tailored to the Actors model [1]. In [4] we investigated the requirements the model poses to the definition a logic and proposed a proof-theory which constrains the synchronous value-passing calculus of [7] to deal only with synchronous object creation and asynchronous message passing for named reconfigurable objects. It turns out that this logic can be used without any modification to capture mobility. Basically, our approach consists in annotating located objects with an additional attribute, containing a reference to location objects as in [8]. The advantage of approaching mobility in this extra-logical manner is that specifications can be defined in isolation to be subsequently combined and global proofs can be decomposed in lemmas to be locally verified much in the way we design any system using the same logic.

A few related work can be gathered in the literature. In [20] additional notation is suggested to treat mobility using the programming logic of UNITY [3]. During the refinement process, specifications are augmented with logical variables to handle time and action, which are built in here, and with concrete locations. As shown in the literature, there are many benefits in using referential location information instead [8, 12]. In addition, if mobility arises in a set of requirements, that approach would not be so effective: initial specifications are required before any mobility aspect can be considered.

In the process calculi literature, mobility has received a lot of attention, motivating the evolution of the static process configurations of SCCS [13] to the dynamic ones of the π-calculus [14]. We have shown here that many applications of the π-calculus can be treated using our logic. For instance, we can simulate recursion creating continuation actors and exchanging asynchronous messages; dynamic data structures can be represented as objects and so on. More importantly, mobility receives a rather different treatment as process in the π-calculus are modelled as terms and here objects are represented by theory presentations. This is due to different design decisions: while it is relatively easier to define notions of simulation and reduction for processes, it is possible to specify and reason about objects as first-class entities using the same more expressive language, which we feel more appropriate to represent the real world. The same may only be achieved by adjoining a modal or temporal logic to process calculi. It would be interesting to compare our logic to those presented in [15] in terms of expressive power.

As a case study, we presented here the design of a location management architecture for networks of mobile users and devices. We are currently investigating the occurrence of failures in this framework in order to access if our logic is still convenient for specifying fault-tolerant behaviour. An alternative direction for further work is to refine the proposed specifications to obtain a concrete implementation. Refinement theories for logics of actions and objects already exist [6, 11] and could be adapted to our case. A challenge in such an effort would be to achieve a compositional development process, where the refinement of components and interconnections in isolation always yield an implementation for the whole system.

Acknowledgements: The author would like to thank two anonymous referees for their valuable suggestions to improve an earlier version of this paper. This work has been financially supported by CNPq, the Brazilian National Research Council.

REFERENCES

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[2] G. Agha, I. A. Mason, S. Smith, and C. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, 1997.

[3] K. M. Chandy and J. Misra. *Parallel Program Design, A Foundation.* Addison-Wesley, 1988.

[4] C. H. C. Duarte. Towards a proof-theoretic foundation for actor specification and verification. In P.-Y. Schobbens and A. Cesta, editors, *Proc. 4th Workshop on Formal Models of Agents (ModelAge'97)*, pages 115–128, January 1997. Certosa di Pontignano, Italy.

[5] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 996–1072. Elsevier, 1990.

[6] J. Fiadeiro and T. Maibaum. Sometimes "tomorrow" is "sometime": Action refinement in a temporal logic of objects. In D. Gabbay and H. Ohlbach, editors, *Temporal Logic*, volume 827 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1994.

[7] J. Fiadeiro, C. Sernadas, T. Maibaum, and G. Saake. Proof-theoretic semantics of object-oriented specification constructs. In R. A. Meersman, W. Kent, and S. Khosla, editors, *Proc. IFIP WG 2.6 Working Conference on Object-Oriented Databases: Analysis, Design and Construction*, pages 243–284. North Holland, 1991.

[8] A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network*, 8(1):62–70, January 1994.

[9] C. B. Jones. *Systematic Software Development Using VDM.* Prentice Hall, 2nd edition, 1990.

[10] D. Lam, J. Jannink, D. C. Cox, and J. Widom. Modelling location management in personal communication systems. In *Proc. of International Conference on Universal Personal Communications (ICUPC'96).* IEEE Press, 1996.

[11] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[12] U. Leonhardt and J. Magee. Towards a general location service for mobile environments. In *Proc. 3rd International Workshop on Service in Distributed and Networked Environments (SDNE'96)*, pages 43–50. IEEE Press, June 1996.

[13] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[14] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, September 1992.

[15] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.

[16] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 2(2):187–260, June 1984.

[17] J. M. Spivey. *The Z Notation: A Reference Manual.* International Series in Computer Science. Prentice-Hall, 1989.

[18] M. Spreitzer and M. Theimer. Architectural considerations for scalable, secure, mobile computing with location information. In *Proc. 14th International Conference on Distributed Computing Systems*, pages 29–38. IEEE Computer Society Press, June 1994.

[19] N. Venkatasubramanian and C. Talcott. A meta architecture for distributed resource management. In *Proc. Hawaii International Conference on System Sciences*, pages 124–133. IEEE Computer Society Press, January 1993.

[20] C. D. Wilcox and G.-C. Roman. Reasoning about places, times and actions in the presence of mobility. *IEEE Transactions on Software Engineering*, 22(4):225–247, April 1996.

APPENDIX

We provide here the set of axiom schemes and inference rules which make our logic suitable for specifying and reasoning about actor communities. These are meant to particularize the axiomatisation of the underlying many sorted first order branching time temporal logic with equality adopted. As in [7] we also require that variables are rigid, attributes have a functional time-dependent interpretation and actions denote atomic events. The reader is referred to [4] for further details.

Definition 7 (Axiom Schemes) Given an actor specification $\Phi = ((\Sigma,\ \mathcal{A},\ \Gamma),\ \Psi)$, the following are logical axiom schemes for Φ-actors:

1. $\bigvee_{c \in \Gamma_c} \exists \vec{v_c} \cdot n.c(\vec{v_c}) \vee \bigwedge_{f \in \mathcal{A}} \forall \vec{v_f}, k \cdot n.f(\vec{v_f}) = k \rightarrow \mathbf{X}(n.f(\vec{v_f} = k))$
2. $\bigwedge_{c \in \Gamma_{(e-e_b)\cup(l-l_b)}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow \mathbf{G}(\neg n_1.\mathbf{init}) \vee \bigwedge_{n \in \vec{v_c}_{\mathsf{addr}} \cup \{n_2\}} n_1.Wait(n, \neg\mathbf{send}(c, n_2, \vec{v_c}))$
3. $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow (\neg n.\mathbf{deliv}(c, \vec{v_c}))\mathbf{W}(n.\mathbf{init})$
4. $\bigwedge_{c \in \Gamma_{(l-l_b)\cup c}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow (\neg n.c(\vec{v_c}))\mathbf{W}(n.\mathbf{init})$
5. $\bigwedge_{c \in \Gamma_{e_b \cup l_b}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow \mathbf{G}(\neg n_1.\mathbf{init}) \vee \bigwedge_{n \in \vec{v_c}_{\mathsf{addr}} \cup \{n_2\}} n_1.Wait(n, \neg\mathbf{new}(c, n_2, \vec{v_c}))$
6. $\bigwedge_{c \in \Gamma_{l_b}} \mathbf{beg} \rightarrow \mathbf{G}(\exists n_1, n_2, \vec{v_c} \cdot \mathbf{E}(n_1.\mathbf{new}(c, n_2, \vec{v_c})))$
7. $\bigwedge_{c \in \Gamma_{l_b}} \forall \vec{v_c} \cdot (\exists n_1 \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}) \rightarrow \mathbf{X}(n_2.c(\vec{v_c}))) \wedge (n_2.c(\vec{v_c}) \overset{i}{\leftarrow}_{\mathbf{beg}} \exists n_1 \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}))$
8. $\bigwedge_{\substack{c,d \in \Gamma_{l_b} \\ d \neq c}} \forall \vec{v_c} \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}) \rightarrow \nexists n_3, \vec{v_c}', \vec{v_d} \cdot ((n_3 \neq n_1 \vee \vec{v_c}' \neq \vec{v_c}) \wedge n_3.\mathbf{new}(c, n_2, \vec{v_c}')) \vee n_3.\mathbf{new}(d, n_2, \vec{v_d})$
9. $\bigwedge_{\substack{c,d \in \Gamma_{l-l_b} \\ d \neq c}} \forall \vec{v_c} \cdot n.\mathbf{deliv}(c, \vec{v_c}) \rightarrow \nexists \vec{v_c}', \vec{v_d} \cdot (\vec{v_c}' \neq \vec{v_c} \wedge n.\mathbf{deliv}(c, \vec{v_c}')) \vee n.\mathbf{deliv}(d, \vec{v_d})$
10. $\bigwedge_{\substack{c,d \in \Gamma_{(l-l_b)\cup c} \\ c \neq d}} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \rightarrow \nexists \vec{v_c}', \vec{v_d} \cdot (\vec{v_c}' \neq \vec{v_c} \wedge n.c(\vec{v_c}')) \vee n.d(\vec{v_d})$

where:

$$Wait(n, g) \equiv (g)\mathbf{W}(\mathbf{init}) \wedge (g)\mathbf{W}(Acq(n))$$
$$Acq(n) \equiv \bigvee_{d \in \Gamma_{l-l_b}} \exists \vec{v_d} \cdot (\mathbf{deliv}(d, \vec{v_d}) \wedge n \in \vec{v_d}) \vee \bigvee_{d \in \Gamma_{l_b}} \exists \vec{v_d} \cdot (d(\vec{v_d}) \wedge n \in \vec{v_d}) \vee \bigvee_{d \in \Gamma_{e_b}} \exists \vec{v_d} \cdot \mathbf{new}(d, n, \vec{v_d})$$

The first scheme says that each actor has encapsulated state; only its local computations can change attribute values. The next four schemes say that either an actor is not created within a community or dispatch, delivery and consumption of messages plus local computations and requests for creation do happen before its birth. Notice that the second and fifth schemes are more liberal if the actor is never created but are more restrictive otherwise requiring actor names to become known first due to delivery of a message, the birth of the source or the creation of the target before they could be used in the task. The sixth scheme says that it is always possible to create some new actors and the seventh states that requests for creation and actual births are causally connected. The last three schemes constrain concurrency, i.e. that actors cannot be simultaneously created with the same name; that messages cannot be delivered in parallel to the same actor; and that messages and local computations cannot be processed at the same time; the last axiom being supplied only to simplify specification and reasoning.

Definition 8 (Rules of Inference) Given an actor specification $\Phi = ((\Sigma,\ \mathcal{A},\ \Gamma),\ \Psi)$, the following are inference rules for deriving properties of existing Φ-actors, where each p, p' and q is an arbitrary formula over a single actor and n, n' and m are terms of sort addr:

$$\boxed{[EXIST] \quad d \in \Gamma_{l_b} \quad \frac{\begin{array}{ll} 1. & p' \to \exists \vec{v_d} \cdot n'.\mathbf{new}(d,m,\vec{v_d}) \\ 2. & p \to q \vee \bigvee_{c\in\Gamma_{l_b}} \exists \vec{v_c} \cdot n.\mathbf{new}(c,m,\vec{v_c}) \end{array}}{p' \to \mathbf{XG}(p \to q)}}$$

$$\boxed{[SAFE] \quad \frac{\begin{array}{ll} 1. & \bigwedge_{c\in\Gamma_{l_b}} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \to q \\ 2. & \bigwedge_{c\in\Gamma_c} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \wedge q \to \mathbf{X}q \end{array}}{\mathbf{G}q}}
\qquad
\boxed{[INV] \quad \frac{\begin{array}{ll} 1. & \bigwedge_{c\in\Gamma_c} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \wedge q \to \mathbf{X}q \end{array}}{q \to \mathbf{G}q}}$$

$$\boxed{[RESP] \quad d \in \Gamma_{l-l_b} \quad \frac{\begin{array}{ll} 1. & \bigwedge_{c\in\Gamma_c} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \wedge p \to \mathbf{X}(p \vee n.d(\vec{v_d})) \\ 2. & n.d(\vec{v_d}) \to \mathbf{F}q \\ 3. & p \to \mathbf{FE}(n.d(\vec{v_d})) \end{array}}{n.\mathbf{deliv}(d,\vec{v_d}) \to \mathbf{X}(\mathbf{F}p \to \mathbf{F}q)}}$$

$$\boxed{[COM] \quad d \in \Gamma_{l-l_b} \quad \frac{\begin{array}{ll} 1. & \bigwedge_{c\in\Gamma_c} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \wedge p \to \mathbf{X}(p \vee n.\mathbf{deliv}(d,\vec{v_d})) \\ 2. & n.\mathbf{deliv}(d,\vec{v_d}) \to \mathbf{F}q \\ 3. & p \to \mathbf{FE}(n.\mathbf{deliv}(d,\vec{v_d})) \end{array}}{n'.\mathbf{send}(d,n,\vec{v_d}) \to \mathbf{X}(\mathbf{F}p \to \mathbf{F}q)}}$$

The rule $EXIST$, based on the fact that an actor name cannot be reused once it is given to some actor, guarantees a local safety property from the configuration of the actors in the environment. $SAFE$ and INV are the usual rules for verifying local safety and invariance properties. Rules COM and $RESP$ capture the fairness requirements for actors and are to verify that a delivery or a message consumption eventually happen due to an interaction if the actor ever becomes enabled for a similar task in the future.