A Spatio-Temporal Logic for the Specification and Refinement of Mobile Systems

Stephan Merz[1], Martin Wirsing[2], and Júlia Zappe[2]

[1] INRIA Lorraine, LORIA, Nancy
Stephan.Merz@loria.fr
[2] Institut für Informatik, Ludwig-Maximilians-Universität München
{wirsing,zappe}@informatik.uni-muenchen.de

Abstract. We define a variant of Lamport's Temporal Logic of Actions, extended by spatial modalities, that is intended for the specification of mobile systems with distributed state. We discuss notions of refinement appropriate for mobile systems, specifically concerning the topological structure of the system, and show how these can be represented in the logic via quantification and implication, ensuring transitivity and compositionality of refinements.

1 Background

The design of systems that make use of mobile code has recently found wide attention. Advances in network technology sometimes make it more attractive to transmit code for execution at remote sites than to rely on more conventional architectures such as client-server systems. It has quickly become apparent that the design of mobile systems requires specific abstractions that should be reflected in formal development methods and in underlying calculi and logics.

Milner's π calculus [9] has started a line of research that investigates foundational calculi for mobile systems, e.g. [4,5,10,13], emphasizing different aspects of mobility and offering different primitives to describe the interaction of mobile components. In particular, the Ambient Calculus due to Cardelli and Gordon introduced the notion of (nested and dynamically reconfigurable) named administrative domains that must be crossed by mobile code.

Some of these calculi have been complemented by logics that allow to express run-time properties of mobile systems [2,3,11]. These logics typically include both spatial and temporal modalities to reflect the topological system structure as well as its evolution over time. Formulas of these logics are evaluated over process terms via an *intensional* semantics [12] and closely reflect the syntactic structure of processes. This close correspondence makes it difficult, if not impossible, to refine the process terms during system development while preserving the formulas of the logic. In this regard, such logics are inadequate for use as *specification logics* that could underly a method of stepwise system development based on refinement.

In the present paper we follow a different approach and define a spatio-temporal specification logic whose semantics is based on a notion of system behaviors similar to standard (linear-time) temporal logics, and independent of any specific operational calculus. Our main goal in the design of the logic is to support refinement: properties established at a higher level of abstraction should be preserved in the implementation. In the context of reactive systems, this goal has successfully been achieved in Lamport's Temporal Logic of Actions [7]. We therefore follow the general philosophy of TLA, but add spatial modalities to express the topology of configurations. We also discuss, informally, concepts of refinement that arise in the development of mobile systems, and how to represent those in the logic. Besides classical temporal (or operation) refinement that is already supported in TLA since its formulas are invariant under stuttering equivalence, we identify two concepts that are more specific to mobile systems, namely *spatial extension* and *virtualisation of locations*. We show that these concepts can also be represented in our logic via implication and novel forms of quantification that express hiding of local state and of agent names, respectively.

The outline of the paper is as follows: Section 2 introduces Mobile TLA (MTLA) at the hand of a simple agent system. Section 3 gives a more formal account of the syntax and semantics of simple MTLA, the logic that we use to specify closed mobile systems. Section 4 studies refinement principles for mobile systems that motivate extensions of the logic by two forms of quantification. Finally, Sect. 5 summarizes our contributions and indicates future work.

2 Example: Joe's Shopping Agent

The specification of a mobile system describes relevant aspects of network topology as well as of the dynamic system behavior, including the movement and interaction of agents. We identify all network locations by unique (or physical) names. In informal discussions, we do not distinguish between names and the domains or agents they represent. Domains may be nested, giving rise to a tree structure of names, and mobility is formally represented by structural changes of these trees that result from agents moving across domain boundaries.

As our running example, we consider the specification of a simple shopping agent that scans a network in the search of the best offer for some item. We assume a finite, fixed set Net of names that represent (immobile) network locations that the agent may visit during its search; $joe \in Net$ denotes the agent's home location. The name $shopper \notin Net$ is used to denote the mobile shopping agent itself. Its local state is described by three state variables: the variable ctl indicates the control state of the shopping agent; it may assume the values "idle" and "shopping". When state equals "shopping", the variable $item$ indicates the good the shopper is searching for, and $found$ holds the set of offers that the agent has collected so far. For the locations $n \in Net$ we assume the state variable $offers$ to represent the catalogue of goods they offer, and the state variable id to denote a (logical) network name that is used by the agent to remember the origin of an offer.

A high-level MTLA specification appears in Fig. 1 as formula $Shopper$.[1] We now informally explain its meaning; the formal definition of MTLA is postponed to Sect. 3.

[1] We adopt Lamport's convention [6] of writing multi-line conjunctions and disjunctions as lists whose items are labelled with the respective connective, using indentation to suppress parentheses.

$$
\begin{array}{rcl}
\mathit{Init} & \equiv & \mathit{joe.shopper}\langle\mathbf{true}\rangle \wedge \mathit{shopper.ctl} = \text{"idle"} \\
\mathit{Prepare}(x) & \equiv & \wedge\ \mathit{shopper}\langle\mathbf{true}\rangle \wedge \circ\mathit{shopper}\langle\mathbf{true}\rangle \\
& & \wedge\ \mathit{shopper.ctl} = \text{"idle"} \\
& & \wedge\ \mathit{shopper.item}' = x \wedge \mathit{shopper.found}' = \emptyset \\
& & \wedge\ \mathit{shopper.ctl}' = \text{"shopping"} \\
\mathit{GetOffer} & \equiv & \wedge\ \mathit{shopper}\langle\mathbf{true}\rangle \wedge \circ\mathit{shopper}\langle\mathbf{true}\rangle \\
& & \wedge\ \mathit{shopper.ctl} = \text{"shopping"} \wedge \mathit{shopper.item} \in \mathit{offers} \\
& & \wedge\ \mathit{shopper.found}' = \mathit{shopper.found} \\
& & \qquad\qquad \oplus \{(\mathit{id}, \mathit{getOffer}(\mathit{offers}, \mathit{shopper.item}))\} \\
& & \wedge\ \textsc{unchanged}(\mathit{shopper.ctl}, \mathit{shopper.item}) \\
\mathit{PickOffer} & \equiv & \wedge\ \mathit{shopper}\langle\mathbf{true}\rangle \wedge \circ\mathit{shopper}\langle\mathbf{true}\rangle \\
& & \wedge\ \mathit{shopper.ctl} = \text{"shopping"} \wedge |\mathit{shopper.found}| \geq 3 \\
& & \wedge\ \mathit{bestOffer}' = \mathit{pickOffer}(\mathit{shopper.found}, \mathit{shopper.item}) \\
& & \wedge\ \mathit{shopper.ctl}' = \text{"idle"} \\
\mathit{Move}_{n,m} & \equiv & \wedge\ n.\mathit{shopper}\langle\mathbf{true}\rangle \wedge \mathit{shopper.ctl} = \text{"shopping"} \\
& & \wedge\ n.\mathit{shopper} \gg m.\mathit{shopper} \\
\mathit{vars} & \equiv & \mathit{shopper.ctl}, \mathit{shopper.item}, \mathit{shopper.found} \\
\mathit{JoeActions} & \equiv & (\exists x : \mathit{Prepare}(x)) \vee \mathit{PickOffer} \\
\mathit{Shopper} & \equiv & \wedge\ \mathit{Init} \\
& & \wedge\ \Box \bigwedge_{n,m \in \mathit{Net}} n\langle m[\mathbf{false}]\rangle \\
& & \wedge\ \Box\big[\mathit{joe}[\mathit{JoeActions}] \vee \bigvee_{n \in \mathit{Net}} n[\mathit{GetOffer}]\big]_{\mathit{vars}} \\
& & \wedge\ \bigwedge_{n \in \mathit{Net}} \Box\big[\bigvee_{m \in \mathit{Net}} \mathit{Move}_{n,m}\big]_{-n.\mathit{shopper}}
\end{array}
$$

Fig. 1. Specification of a simple shopping agent

The first conjunct *Init* of specification *Shopper* asserts the specifications's initial condition. It requires the shopping agent to be at its home location and its control state to be "idle". We write $n[F]$ to assert that F holds at location n, provided n exists. The formula $n\langle F\rangle$ also requires F to hold at n, but moreover asserts the existence of location n.

The second conjunct of *Shopper* describes part of the network topology: it requires the (immobile) locations $n \in \mathit{Net}$ to be present at all instances of time, and not to be nested inside one another.

The third conjunct of formula *Shopper* specifies the allowed changes to the shopping agent's local state. Similarly as in TLA, a formula $\Box[A]_v$ requires every transition that changes v to satisfy the transition formula A, which may refer to the post-state of the transition via either the next-state operator $\circ$ or primed state variables. Our example allows three kinds of transitions: at location *joe*, one of the *Prepare* or *PickOffer* actions may occur, whereas transitions described by *GetOffer* are allowed to occur at any location $n \in \mathit{Net}$ (even including the shopper's home location). For example, *GetOffer* requires the shopper to be and remain at the location of evaluation, in state "shopping", and the target item to be among the goods offered at the current location. The agent then inserts a pair $(\mathit{id}, \mathit{val})$ into the set *found* of collected offers where *id* is the logical identity of the current location and *val* is the offer as denoted by the (static) function *getOffer*. The other transition specifications can be interpreted similarly.

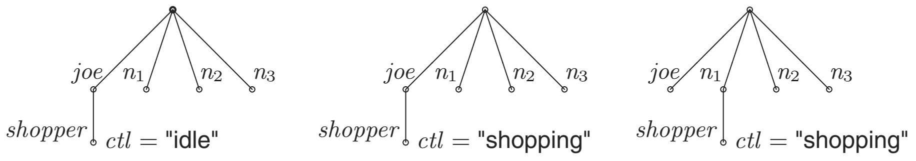

Fig. 2. Prefix of a run

The final conjunct of specification *Shopper* describes the shopper's movements about the network. A formula $\Box[A]_{-n}$ asserts that every transition such that $n\langle\mathbf{true}\rangle$ is true before and false after the transition has to satisfy A. In our example, we require action $Move_{n\text{`}m}$ to be performed, for some location $m \in Net$, whenever the shopper leaves domain $n$. The conjunct $n.shopper \gg m.shopper$ that appears in the description of $Move_{n\text{`}m}$ requires the subtree denoted by *shopper* located below the domain n to move to the domain m without changing the agent's local state.

A more complete specification of the shopping agent would also include fairness and liveness properties which we have omitted because they do not play an important role for the remainder of this paper.

Besides system specifications, MTLA formulas can also express system properties, including invariants. For example, the shopping agent described by the specification in Fig. 1 is always located beneath some network node, and therefore the implication

$$Shopper \Rightarrow \Box \bigvee_{n \in Net} n.shopper\langle\mathbf{true}\rangle$$

is valid. Similarly, the implication

$$Shopper \Rightarrow \Box(shopper.ctl = \text{"idle"} \Rightarrow joe.shopper\langle\mathbf{true}\rangle)$$

asserts that the shopping agent can be in its idle state only if it is at its home location.

3 Simple Mobile TLA

We describe the topological structure of a mobile system at any given instant as a finite tree t whose edges are labelled with unique ("physical") names n drawn from a denumerably infinite set N, as depicted in Fig. 2. Equivalently, instead of labelling edges we often consider the labels to be attached to the target nodes, such that every node of t except for the root carries a unique label. Moreover, with every node of the tree we associate a local state.

Technically, a *configuration* is a pair (t, λ): the tree t is given by a prefix-closed set $t \subseteq \mathsf{N}^*$ such that for any $\alpha n, \beta m \in t$ we have $n = m$ only if $\alpha = \beta$; the empty word ε denotes the root of the tree. For every node $\alpha \in t$, the local state $\lambda(\alpha)$ is a valuation of the state variables as explained below. A *run* of a system is represented as an ω-sequence $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\dots$ of configurations, cf. Fig. 2. For technical reasons we require that $t_i \neq \emptyset$ for all $i \in \mathbb{N}$. Transitions may change the local state at some nodes, but also modify the tree structure; structural changes represent the movement of agents across administrative domains or the creation or destruction of agents. For a run $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\dots$ and $i \in \mathbb{N}$, we denote by $\sigma|_i$ the suffix $(t_i, \lambda_i)(t_{i+1}, \lambda_{i+1}), \dots$.

A tree t induces a partial order on the names that it contains. More precisely, we write $m \preceq_t n$ for $m, n \in \mathsf{N}$ and say that m *is below* n in t iff $\alpha n \beta \in t$ for some $\alpha, \beta \in \mathsf{N}^*$ such that $\alpha n \beta$ ends in m. For a tree t and a name $n \in \mathsf{N}$, we write

$$t{\downarrow}n = \{\beta \in \mathsf{N}^* \mid \alpha n \beta \in t \text{ for some } \alpha \in \mathsf{N}^*\}$$

for the subtree of t rooted at the (unique) node labelled by n. If no such node exists, $t{\downarrow}n$ denotes the empty tree $\emptyset$. This notation is extended to sequences $\alpha \in \mathsf{N}^*$ by defining $t{\downarrow}\varepsilon = t$ and $t{\downarrow}\alpha n = (t{\downarrow}\alpha){\downarrow}n$. For a name n that occurs in a tree t, i.e. such that $t{\downarrow}n \neq \emptyset$, we write $\pi(t, n)$ for the unique path $\alpha n \in t$ ending in name n. By N^+ we denote the set $\mathsf{N} \cup \{\varepsilon\}$ (we assume that $\varepsilon \notin \mathsf{N}$). We extend some of our notation to N^+ by letting $\pi(t, \varepsilon) = \varepsilon$ if $t \neq \emptyset$ and defining $n \preceq_t \varepsilon$ (where $n \in \mathsf{N}^+$) iff $t{\downarrow}n \neq \emptyset$, and $\varepsilon \preceq_t n$ to hold for no $n \in \mathsf{N}$. We write $m \prec_t n$ if $m \preceq_t n$ and $m \neq n$.

The connectives of MTLA extend classical first-order logic by spatial and temporal modalities. We also add an operator to describe structural modifications of trees during transitions. Formally, we define (pure) formulas and terms as well as "impure" ones; the latter generalize the transition formulas of TLA [7]. We assume given a signature (consisting of function and predicate symbols) of first-order logic with equality and denumerable sets $\mathcal{V}_r$ and $\mathcal{V}_f$ of rigid and flexible individual variables. The semantics of MTLA assumes a first-order interpretation $\mathcal{I}$ (defining a non-empty universe $|\mathcal{I}|$ and interpretations of the function and predicate symbols). Terms and formulas are interpreted with respect to a run whose valuations interpret the flexible variables, an index $n \in \mathsf{N}^+$ that indicates the "location of evaluation", and a valuation of the rigid variables. In the following inductive definition, clauses 1–8 as well as 11, 12, and 14 are standard [7,8] whereas clauses 9, 10, 13, and 15 introduce the spatial extensions of MTLA.

Definition 1. *Assume a fixed first-order interpretation $\mathcal{I}$, a run $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\dots$ with $\lambda_i : t_i \to (\mathcal{V}_f \to |\mathcal{I}|)$, and a valuation $\xi : \mathcal{V}_r \to |\mathcal{I}|$. We define the terms and formulas of MTLA and their semantics, for arbitrary $n \in \mathsf{N}^+$.*

1. *Every variable $x \in \mathcal{V}_r \cup \mathcal{V}_f$ is a (pure) term. For a rigid variable $x \in \mathcal{V}_r$, we define $\sigma^{(n\cdot\square)}(x) = \xi(x)$. For a state variable $x \in \mathcal{V}_f$, we let $\sigma^{(n\cdot\square)}(x) = \lambda_0(\pi(t_0, n))(x)$ be the value assigned to x by the local interpretation associated with node n of the initial configuration in σ, provided that $t_0{\downarrow}n \neq \emptyset$; otherwise $\sigma^{(n\cdot\square)}(x)$ is an arbitrary but fixed element $a \in |\mathcal{I}|$.*
2. *If $t_1, \dots, t_k$ are (im)pure terms and f is a k-ary function symbol then $f(t_1, \dots, t_k)$ is again an (im)pure term whose interpretation is given by $\sigma^{(n\cdot\square)}(f(t_1, \dots, t_k)) = \mathcal{I}(f)(\sigma^{(n\cdot\square)}(t_1), \dots, \sigma^{(n\cdot\square)}(t_k))$.*
3. *If A is an (im)pure formula and $x \in \mathcal{V}_r$ then $\varepsilon x : A$ is an (im)pure term where $\sigma^{(n\cdot\square)}(\varepsilon x : A)$ is some value $a \in |\mathcal{I}|$ such that $\sigma\ n, \xi[x := a] \models A$ if some such value exists, otherwise $\sigma^{(n\cdot\square)}(\varepsilon x : A)$ is some arbitrary but fixed value $a \in |\mathcal{I}|$.*
4. *Every pure term or formula is also an impure term or formula.*
5. *If P is a k-ary predicate symbol and $t_1, \dots, t_k$ are (im)pure terms then $P(t_1, \dots, t_k)$ is an (im)pure formula. We define $\sigma\ n, \xi \models P(t_1, \dots, t_k)$ and say that $P(t_1, \dots, t_k)$ holds at location n in σ, iff $(\sigma^{(n\cdot\square)}(t_1), \dots, \sigma^{(n\cdot\square)}(t_k)) \in \mathcal{I}(P)$.*

6. **false** *is a pure formula that holds nowhere:* $\sigma\ n, \xi \not\models$ **false**.
7. *If* A, B *are (im)pure formulas then so is* $A \Rightarrow B$*. We define* $\sigma\ n, \xi \models A \Rightarrow B$ *iff* $\sigma\ n, \xi \not\models A$ *or* $\sigma\ n, \xi \models B$.
8. *If* A *is an (im)pure formula and* $x \in \mathcal{V}_r$ *then* $\exists x : A$ *is again an (im)pure formula, and* $\sigma\ n, \xi \models \exists x : A$ *iff* $\sigma\ n, \xi[x := a] \models A$ *for some* $a \in |\mathcal{I}|$.
9. *If* A *is an (im)pure formula and* $m \in \mathsf{N}$ *then* $m[A]$ *is again an (im)pure formula whose interpretation is given by* $\sigma\ n, \xi \models m[A]$ *iff* $m \prec_{t_0} n$ *implies* $\sigma\ m, \xi \models A$.
10. *If* A *is an (im)pure formula then* $\overline{\square}A$ *("everywhere* A*") is again an (im)pure formula, and* $\sigma\ n, \xi \models \overline{\square}A$ *iff* $\sigma\ m, \xi \models A$ *for all* $m \in \mathsf{N}^+$ *such that* $m \preceq_{t_0} n$.
11. *If* F *is a pure formula then* $\square F$ *("always* F*") is a pure formula with semantics* $\sigma\ n, \xi \models \square F$ *iff for all* $i \in \mathbb{N}$, $t_j{\downarrow}n = \emptyset$ *for some* $j \leq i$ *or* $\sigma|_i, n, \xi \models F$.
12. *If* F *is a pure formula then* $\circ F$ *("next-time* F*") is an impure formula, and we define* $\sigma\ n, \xi \models \circ F$ *iff* $t_1{\downarrow}n = \emptyset$ *or* $\sigma|_1, n, \xi \models F$.
13. *For* $m \in \mathsf{N}$ *and* $\alpha, \beta \in \mathsf{N}^*$, $\alpha\ m \gg \beta\ m$ *is an impure formula whose semantics is defined by* $\sigma\ n, \xi \models \alpha\ m \gg \beta\ m$ *iff both* $t_0{\downarrow}n\alpha m = t_1{\downarrow}n\beta m$ *and* $\lambda_0(\pi(t_0, m)\gamma) = \lambda_1(\pi(t_1, m)\gamma)$ *for all* $\gamma \in t_0{\downarrow}n\alpha m$.
14. *If* A *is an impure formula and* t *is a pure term then* $\square[A]_t$ *is a pure formula, and* $\sigma\ n, \xi \models \square[A]_t$ *iff for all* $i \in \mathbb{N}$, $t_j{\downarrow}n = \emptyset$ *for some* $j \leq i$ *or* $\sigma|_i^{(n\langle\square)}(t) = \sigma|_{i+1}^{(n\langle\square)}(t)$ *or* $\sigma|_i, n, \xi \models A$.
15. *If* A *is an impure formula and* S *is a non-temporal formula, i.e., built only using rules 1–10, then* $\square[A]_S$ *is a pure formula with semantics* $\sigma\ n, \xi \models \square[A]_S$ *iff for all* $i \in \mathbb{N}$, $t_j{\downarrow}n = \emptyset$ *for some* $j \leq i$ *or* $\sigma|_i, n, \xi \models S$ *iff* $\sigma|_{i+1}, n, \xi \models S$ *or* $\sigma|_i, n, \xi \models A$.

We say that F *holds of* σ*, written* $\sigma\xi \models F$ *iff* $\sigma\varepsilon, \xi \models F$*. Formula* F *is* valid*, written* $\models F$*, iff* $\sigma\xi \models F$ *holds for all runs* σ *and valuations* ξ.

Like TLA, MTLA is a linear-time temporal logic: formulas are interpreted over linear sequences of states. However, terms and formulas of MTLA are evaluated relative to a location, identified by a name n. Intuitively, the point of evaluation "follows" the movements of n in the tree. Because names may be created and deleted, the temporal operators of MTLA are effectively restricted to the possibly finite life-spans of a name, which ends when the name disappears from a tree. In particular, we consider a possible reappearance of a name in a later tree to represent a new domain that happens to reuse the same physical name.

The spatial modalities $m[\_]$ and $\overline{\square}$ shift the spatial focus of evaluation. A formula $m[F]$ is "weak" in the sense that it holds trivially if the name m does not occur below the current point of evaluation. Moreover, note that the operator $m[\_]$ "looks arbitrarily far inside" the tree; as we will argue in Sect. 4.2, this is important if we want to refine a single domain by a hierarchy of domains. The "everywhere" operator refers to all nodes of the subtree rooted at the current point of evaluation.

The distinction between pure and impure formulas in the definition of a variant of TLA was introduced in [8] where it was shown that such a mutually recursive definition (compared to a tier of temporal formulas on top of a tier of transition formulas as in TLA) makes the logic more expressive while simplifying its axiomatization. "Impureness" is introduced by the next-time operator $\circ$ of linear-time temporal logic or by the "move" operator $\gg$; impure formulas must be guarded by the $\Box[\_]_t$ or $\Box[\_]_S$ operators to produce a pure formula. This syntactic restriction ensures that all pure MTLA formulas are invariant under finite stuttering. Refinements are therefore allowed to introduce low-level steps that are invisible at the level of the original specification. Whereas formulas $\Box[A]_t$ specify the allowed changes of local states, formulas $\Box[A]_S$ are used to describe structural modifications of trees.

The formula $\alpha\ m \gg \beta\ m$ describes the movement of an agent m, including all enclosed sub-locations and their local states, from subdomain α to subdomain β, which would be impossible to specify using just formulas of the form $n[F]$ that refer only to single locations, not to entire subtrees. If the path αm does not exist below the current location, the definition of $\alpha\ m \gg \beta\ m$ requires m not to occur below β in the subtree after the transition.

When writing MTLA specifications we use many derived operators, beyond the standard abbreviations **true**, $\wedge$, $\vee$, and $\forall$. For a pure term t, we define the impure term $t' \equiv \varepsilon x : \circ(t = x)$ to denote the value of t at the next instant. Similarly, for an (im)pure term t and a name $n \in \mathsf{N}$, $n.t$ denotes the (im)pure term $\varepsilon x : n[x = t]$ that denotes the value of t at sublocation n, provided such a location exists. For pure terms $t_1, \ldots, t_n$ we write $\textsc{unchanged}(t_1, \ldots, t_n)$ to denote the impure formula $t_1' = t_1 \wedge \ldots \wedge t_n' = t_n$.

The formula $n\langle F\rangle \equiv \neg n[\neg F]$ is defined as the dual of $n[F]$; it requires the existence of a sublocation n such that F holds at n. To reduce the number of brackets, we sometimes write n (for a name $n \in \mathsf{N}$) instead of $n\langle\mathbf{true}\rangle$, asserting the existence of a location named n in the current tree, and write $n_1.\cdots.n_k[F]$ and $n_1.\cdots.n_k\langle F\rangle$ instead of $n_1[\cdots n_k[F]\cdots]$ and $n_1\langle\cdots n_k\langle F\rangle\cdots\rangle$.

The formula $\overline{\Diamond} P$ ("somewhere P") is defined as $\neg\overline{\Box}\neg P$ and holds of σ if P holds at some sublocation. Similarly, $\Diamond F$ ("eventually P") is defined as the dual of $\Box F$; it requires F to hold eventually (within the life span of the current name). We write $\Diamond\langle A\rangle_t$ for $\neg\Box[\neg A]_t$, and similarly for $\Diamond\langle A\rangle_S$; these formulas hold if eventually t (resp., S) change value during a transition satisfying A. The formulas $\Box[A]_{-S}$ and $\Box[A]_{+S}$ abbreviate $\Box[S \Rightarrow A]_S$ and $\Box[\neg S \Rightarrow A]_S$; these formulas assert that A holds whenever the spatial formula S becomes false (resp., true) during a transition. Finally, the formula $\Box[A]_{u_1,\ldots,u_n}$ (where the u_i may be pure terms or pure spatial formulas) denotes $\Box[A]_{u_1} \wedge \ldots \wedge \Box[A]_{u_n}$; it holds of σ provided every transition that changes some u_i satisfies A.

Although we will mostly argue semantically, we list a few axioms of MTLA. It is easy to see that implication distributes over all operators of "rectangular shape":

$$\models \boxtimes(A \Rightarrow B) \Rightarrow (\boxtimes A \Rightarrow \boxtimes B) \quad \text{for } \boxtimes \in \{n[\_], \overline{\Box}, \Box, \Box[\_]_u\}$$

The "everywhere" operator quantifies over all paths, so we have

$$\models \overline{\Box} F \Rightarrow F \qquad \text{and} \qquad \models \overline{\Box} F \Rightarrow n[F] \ \text{ for all } n \in \mathsf{N}$$

Finally, we have axioms that correspond to the assumed uniqueness of names and that express a form of "absorption" for the modalities $n[\_]$, which look arbitrarily deep into the tree. More precisely,

$$\models n[F] \Leftrightarrow \overline{\Box} n[F] \qquad \text{and} \qquad \models m.n\langle\mathbf{true}\rangle \Rightarrow (m.n[F] \Leftrightarrow n[F])$$

4 Refinement of Mobile Systems

The general idea behind refinement concepts is to allow a high-level description of a system to be replaced by a lower-level implementation while preserving the properties established at the higher level of abstraction. Concerning the refinement of mobile systems, we identify three basic principles of refinement that should be supported:

1. *Operation refinement* is a classical principle that is well-known from sequential and reactive systems. In particular, operations can be made more deterministic, and they can be decomposed into sequences of finer-grained actions.
2. *Spatial extension* can be used to decompose a single, high-level location n into a tree of sub-locations that collectively implement the behavior required of n, and whose root is again named n. In general, the local state originally associated with node n will be distributed among the locations of the implementation; it should then be hidden from the interface of the abstract specification.
3. *Virtualisation of locations* allows to replace a location of the abstract specification by a structurally different location hierarchy, with a different name. This form of refinement requires the name of the "virtualised" location to be hidden from the high-level interface.

A single refinement step may combine several of these principles. For example, we will see that a combination of operation refinement and virtualisation allows an atomic high-level move action to be implemented as a sequence of lower-level moves. We now consider each of the basic principles in more detail, motivating corresponding extensions of Simple MTLA, and illustrate them at the hand of our running example. Again, we follow the lead of TLA where refinement of a high-level specification Abs with internal variables aux by a low-level specification $Conc$ is expressed by validity of the implication

$$\models Conc \Rightarrow \exists\, aux : Abs$$

4.1 Operation Refinement

Figure 3 shows a specification of the shopping agent whose move actions have been restrained in two ways: first, moves from location n to another shop m are allowed only if the offers made at n (if any) have been entered in the agent's records and if m has not been visited before (i.e., its id does not appear in the agent's records). Second, moves to the home location are allowed only if the agent has recorded the offers made at n and if it has collected at least three offers. The restrained shopping agent's specification *RestrShopper* is identical to formula *Shopper* except for the fourth conjunct. It follows immediately from the definitions that both

$$MoveHome_n \Rightarrow Move_{n\text{`}joe} \quad \text{and} \quad VisitShop_{n\text{`}m} \Rightarrow Move_{n\text{`}m}$$

are valid. Propositional logic and the monotonicity of the operator $\Box[\_]_S$ imply

$$\models RestrShopper \Rightarrow Shopper$$

$$\begin{aligned}
\mathit{VisitShop}_{n,m} \equiv\ & \land\ n.\mathit{shopper}\langle\mathbf{true}\rangle \land \mathit{shopper}.\mathit{ctl} = \text{"shopping"}\\
& \land\ \mathit{shopper}.\mathit{item} \notin n.\mathit{offers} \lor n.\mathit{id} \in \mathrm{dom}(\mathit{shopper}.\mathit{found})\\
& \land\ m.\mathit{id} \notin \mathrm{dom}(\mathit{shopper}.\mathit{found})\\
& \land\ n.\mathit{shopper} \gg m.\mathit{shopper}\\
\mathit{MoveHome}_{n} \equiv\ & \land\ n.\mathit{shopper}\langle\mathbf{true}\rangle \land \mathit{shopper}.\mathit{ctl} = \text{"shopping"}\\
& \land\ \mathit{shopper}.\mathit{item} \notin n.\mathit{offers} \lor n.\mathit{id} \in \mathrm{dom}(\mathit{shopper}.\mathit{found})\\
& \land\ |\mathit{shopper}.\mathit{found}| \geq 3\\
& \land\ n.\mathit{shopper} \gg \mathit{joe}.\mathit{shopper}\\
\mathit{RestrShopper} \equiv\ & \land \ldots\\
& \land \textstyle\bigwedge_{n\in \mathit{Net}} \Box\big[\mathit{MoveHome}_n \lor \bigvee_{m\in \mathit{Net}\setminus\{\mathit{Joe}\}} \mathit{VisitShop}_{n,m}\big]_{-n.\mathit{shopper}}
\end{aligned}$$

Fig. 3. Restraining the movement of the shopping agent

reflecting the fact that the specification of Fig. 3 is a possible refinement of the original specification shown in Fig. 1.

As in TLA, operation refinement based on the decomposition of a high-level action into sequences of implementation actions is also formally expressed by validity of implication. This is a consequence of the invariance of MTLA formulas under stuttering. We now turn to refinement principles that are specific to mobile systems because they change the topological structure of configurations.

4.2 Spatial Extension without Distribution of State

During system development, one may choose to implement a single location of the high-level specification by a hierarchy of locations. Semantically, this is reflected in a situation as illustrated in Fig. 4 where a single location n_1 is refined into a tree with root n_1 and new sub-locations in, out, and dock. Sublocations of the original location may be assigned to different sub-trees of the implementation. However, the spatial relations between the locations that are visible in the high-level specification are preserved.

In the context of the "shopping agent" example, let us assume that every network node is equipped with a designated "dock" location to hold visiting agents, and that incoming and outgoing agents are placed into "in" and "out" communication buffers. Figure 5 contains a specification of a corresponding version of the (original) shopping agent example. Formula $\mathit{DockedMove}_{n‘m}$ describes a transition where the shopping

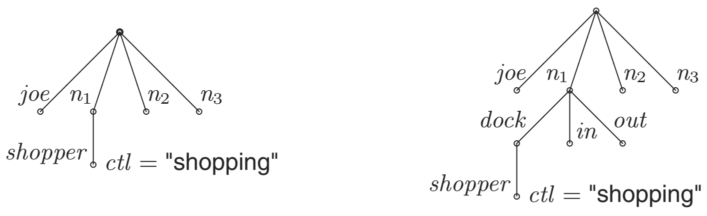

Fig. 4. Spatial extension of node n_1

$$\begin{array}{rcl}
\mathit{DockedInit} & \equiv & \mathit{joe}.\mathit{dock}_{\mathit{joe}}.\mathit{shopper}\langle\mathbf{true}\rangle \wedge \mathit{shopper}.\mathit{ctl} = \text{"idle"} \\
\mathit{SendShopper}_n & \equiv & \wedge\ n.\mathit{dock}_n.\mathit{shopper}\langle\mathbf{true}\rangle \wedge \mathit{shopper}.\mathit{ctl} = \text{"shopping"} \\
 & & \wedge\ n.\mathit{dock}_n.\mathit{shopper} \gg n.\mathit{out}_n.\mathit{shopper} \\
\mathit{DockedMove}_{n,m} & \equiv & \wedge\ n.\mathit{out}_n.\mathit{shopper}\langle\mathbf{true}\rangle \\
 & & \wedge\ n.\mathit{out}_n.\mathit{shopper} \gg m.\mathit{in}_m.\mathit{shopper} \\
\mathit{RcvShopper}_n & \equiv & \wedge\ n.\mathit{in}_n.\mathit{shopper}\langle\mathbf{true}\rangle \\
 & & \wedge\ n.\mathit{in}_n.\mathit{shopper} \gg n.\mathit{dock}_n.\mathit{shopper} \\
\mathit{DockedShopper} & \equiv & \wedge\ \mathit{DockedInit} \\
 & & \wedge\ \ldots \\
 & & \wedge\ \bigwedge_{n \in \mathit{Net}} \wedge\ \Box[\mathit{SendShopper}_n]_{-\mathit{dock}_n.\mathit{shopper}} \\
 & & \qquad\qquad \wedge\ \Box[\mathit{RcvShopper}_n]_{-\mathit{in}_n.\mathit{shopper}} \\
 & & \qquad\qquad \wedge\ \Box\big[\bigvee_{m \in \mathit{Net}} \mathit{DockedMove}_{n,m}\big]_{-\mathit{out}_n.\mathit{shopper}}
\end{array}$$

Fig. 5. Network nodes with agent docks

agent is taken out of the out_n buffer associated with location n and migrates to the in_m buffer of location m. The formulas $\mathit{SendShopper}_n$ and $\mathit{RcvShopper}_n$ specify the movements between the in_n and out_n buffers and the dock_n location where the agent is hosted during its visit. (A more complete elaboration of this refinement would strengthen the preconditions of the other transition to assert that the agent is actually "in dock" and would describe any necessary packing, unpacking, and security checks before actually placing the agent in the dock.)

Specification $\mathit{DockedShopper}$ is a refinement of $\mathit{Shopper}$, and in fact, the implication

$$\mathit{DockedShopper} \Rightarrow \mathit{Shopper}$$

is valid. The proof relies on the invariant that the shopper can only be placed in the "out" buffer when it is in state "shopping", formally expressed by the invariant

$$\mathit{DockedShopper} \Rightarrow \bigwedge_{n \in \mathit{Net}} \Box(\mathit{out}_n.\mathit{shopper}\langle\mathbf{true}\rangle \Rightarrow \mathit{shopper}.\mathit{ctl} = \text{"shopping"})$$

which is easily seen to follow from the definition of formula $\mathit{SendShopper}_n$. Observe also that the moves between the dock and the communication buffers correspond to invisible, "stuttering" transitions of the original specification because the shopping agent stays within the respective network domain. Therefore, these actions are allowed by formula $\mathit{Shopper}$.

For spatial extension to be an admissible refinement principle, it is important that formulas $n[F]$ be interpreted as referring not just to a sublocation n immediately beneath the current location but to locations arbitrarily deep in the subtree.

4.3 Spatial Extension with Distribution of State

In the case of specification $\mathit{DockedShopper}$, we were able to represent spatial extension simply by implication because the "dock", "in", and "out" sub-locations did not introduce any local state. In general, spatial extension will be accompanied with a distribution of the state variables associated with the high-level location n among the lower-level locations. Intuitively, such a distribution of local state is permissible provided that no other component attempts to directly access the local state at location n. In other words, state variables that are not part of the external "interface" of a specification may be distributed in the implementation. Because we do not wish to impose a fixed set of visibility rules, the specifier has to explicitly designate the interface in the specification by indicating which state variables should be hidden from the interface.

Logically, hiding local state components corresponds to existential quantification over flexible variables at certain locations. We therefore extend the syntax of MTLA formulas.

Definition 2. *The definition of MTLA formulas is extended by the following clause.*

16. If F is a pure formula, $m \in \mathsf{N}$ is a name, and $v \in \mathcal{V}_f$ is a flexible variable then $\boldsymbol{\exists}\, m.v : F$ is again a pure formula.

As in TLA [7], the semantics of quantification over flexible variables is somewhat complicated in order to preserve invariance under finite stuttering. We formally define *stuttering equivalence* as the smallest equivalence relation $\simeq$ on runs that identifies runs that differ by insertion or removal of duplicate configurations (t_i, λ_i):

$$\ldots(t_i, \lambda_i)(t_{i+1}, \lambda_{i+1})\ldots \;\simeq\; \ldots(t_i, \lambda_i)(t_i, \lambda_i)(t_{i+1}, \lambda_{i+1})\ldots$$

It is straightforward to show that pure formulas of Simple MTLA are invariant w.r.t. stuttering equivalence, that is, for any MTLA formula F we have $\sigma\xi \models F$ iff $\tau, \xi \models F$ whenever $\sigma \simeq \tau$.

We say that runs $\sigma = (s_0, \lambda_0)(s_1, \lambda_1)\ldots$ and $\tau = (t_0, \mu_0)(t_1, \mu_1)\ldots$ are *equal up to $v \in \mathcal{V}_f$ at $m \in \mathsf{N}$*, written $\sigma =_{m \triangleright v} \tau$ iff $s_i = t_i$ for all $i \in \mathbb{N}$ and $\lambda_i(\alpha)(x) = \mu_i(\alpha)(x)$ except possibly when $x \equiv v$ and $\alpha \equiv \beta\, m$ for some $\beta \in \mathsf{N}^*$. In other words, the tree structures of the configurations in σ and τ have to be identical, and the local valuations may differ at most in the valuation assigned to variable v at nodes labelled m.

Finally, we define *similarity up to v at m* as the smallest equivalence relation $\approx_{m \triangleright v}$ that contains both $\simeq$ and $=_{m \triangleright v}$. We define the semantics of existential quantification over flexible variables by

$$\sigma\; n, \xi \models \boldsymbol{\exists}\, m.v : F \quad \text{iff} \quad \tau, n, \xi \models F \;\text{ for some } \tau \approx_{m \triangleright v} \sigma$$

This definition clearly ensures that MTLA formulas of the form $\boldsymbol{\exists}\, m.v : F$ are again invariant w.r.t. stuttering equivalence.

As an example, we present another spatial extension of the shopping agent specification where we assume the databases containing the offers to reside in a sub-location db_n of each network node n. Its specification appears as formula *DBShopper* in Fig. 6. The formula *GetOffer* has been changed to access the database *offers* hosted at the sub-location db_n instead of directly at location n. The new specification is a refinement of the original one when the variable *offers* of each node $n \in Net$ is hidden from the interface; formally, the implication

$$\mathit{DBShopper} \Rightarrow \boldsymbol{\exists}\, n_1.\mathit{offers}, \ldots, n_k.\mathit{offers} : \mathit{Shopper}$$

$$\begin{array}{rcl}
\mathit{DBGetOffer}_n & \equiv & \wedge\ \mathit{shopper}\langle\mathbf{true}\rangle \wedge \circ\mathit{shopper}\langle\mathbf{true}\rangle \\
& & \wedge\ \mathit{shopper.ctl} = \text{"shopping"} \wedge \mathit{shopper.item} \in \mathit{db}_n.\mathit{offers} \\
& & \wedge\ \mathit{shopper.found}' = \mathit{shopper.found} \\
& & \qquad\qquad\qquad \oplus \{(\mathit{id} \mapsto \mathit{getOffer}(\mathit{db}_n.\mathit{offers}, \mathit{shopper.item}))\} \\
& & \wedge\ \text{UNCHANGED}(\mathit{shopper.ctl}, \mathit{shopper.item}) \\
\mathit{DBShopper} & \equiv & \wedge\ \mathit{Init} \\
& & \wedge\ \Box \bigwedge_{m,n \in Net} n\langle \mathit{db}_n\langle\mathbf{true}\rangle \wedge m[\mathbf{false}]\rangle \\
& & \wedge\ \Box\big[\mathit{joe}[\mathit{JoeActions}] \vee \bigvee_{n\in Net} n[\mathit{DBGetOffer}_n]\big]_{\mathit{vars}} \\
& & \wedge\ \bigwedge_{n \in Net} \Box\big[\bigvee_{m \in Net} \mathit{Move}_{n,m}\big]_{-n.\mathit{shopper}}
\end{array}$$

Fig. 6. Network nodes with sub-location hosting the database

is valid, assuming $Net = \{n_1, \ldots ,n_k\}$. Proofs of such refinements can be based on the following variant of the "refinement mapping" rule of TLA:

$$F[t/m.v] \Rightarrow \exists\, m.v : F \quad \text{provided all occurrences of } v \text{ in subformulas } \overline{\Box} G \text{ of } F \text{ are in the scope of some subformula } a[H] \text{ of } G$$

where t is a pure term and $F[t/m.v]$ denotes the formula F where all top-level occurrences of v in any subformula $m[A]$ (i.e., those occurrences that are not in the scope of any further spatial operator) are replaced by t. For example, refinement of *Shopper* by *DBShopper* can be shown by the above rule by observing the validity of

$$\mathit{DBShopper} \Rightarrow \mathit{Shopper}[\mathit{db}_{n_1}.\mathit{offers}/n_1.\mathit{offers}, \ldots ,\mathit{db}_{n_k}.\mathit{offers}/n_k.\mathit{offers}]$$

4.4 Virtualisation of Locations

As the final and most general of our refinement principles for mobile systems, we consider the replacement of some location n in the abstract specification by a structurally different hierarchy of locations such that the externally observable behavior of the system is preserved. Unlike in the previous case of structural refinement with possible distribution of local state, this new case allows for implementations that do not contain an agent named n, implying that n itself, and not just its state components, should be hidden from the interface of the high-level specification. This motivates a second extension of the syntax of MTLA.

Definition 3. *The definition of MTLA is extended by quantification over names.*

17. If F is a pure formula and $m \in \mathsf{N}$ is a name then $\exists\, m : F$ is again a pure formula.

Intuitively, a run σ satisfies a formula $\exists\, m : F$ if at every configuration a subtree may be identified that "plays the role of m" as described by formula F. Formally, we require the existence of some sequence of configurations that differ from those in σ by *inserting* a new node that behaves as required at appropriate places of every configuration, and explicitly ensure closure under stuttering equivalence.

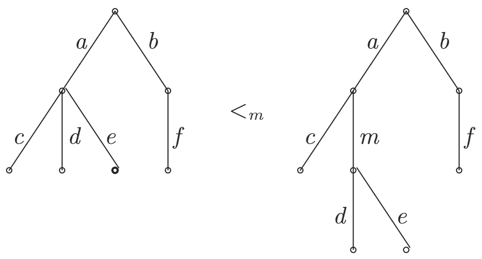

Fig. 7. Illustration of tree extension

For finite trees (with unique labellings) s and t and a name $m \in \mathsf{N}$ we define the relation $s <_m t$ to hold iff s results from t by removing the node labelled by m (if any); formally,

$$s <_m t \quad \text{iff} \quad \text{there exists } \alpha \in \mathsf{N}^* \text{ such that } s = \{\beta \in t : \alpha m \not\sqsubseteq \beta\} \cup \{\alpha\beta \ : \alpha m \beta \in t\}$$

(see Fig. 7 for an illustration of this definition, choosing $\alpha = a$). The relation $<_m$ is extended to configurations in the canonical way by requiring that the local state associated with any node in s be that of the corresponding node in t, and arbitrary at the new node m:

$$(s, \lambda) <_m (t, \mu) \quad \text{iff} \quad s <_m t \quad \text{and} \quad \text{for all } \gamma \in s,$$
$$\lambda(\gamma) = \begin{cases} \mu(\gamma) & \text{if } \gamma \in t \\ \mu(\alpha m \beta) & \text{if } \gamma = \alpha\beta \ \text{ where } \alpha m \beta \in t, \beta \neq \varepsilon \end{cases}$$

Finally, the relation $<_m$ is extended to entire runs by

$$(s_0, \lambda_0)(s_1, \lambda_1) \ldots <_m (t_0, \mu_0)(t_1, \mu_1) \ldots \quad \text{iff} \quad (s_i, \lambda_i) <_m (t_i, \mu_i) \text{ for all } i \in \mathbb{N}.$$

The semantics of quantification over names is now defined by

$$\sigma\ n, \xi \models \exists\, m : F \quad \text{iff} \quad \text{there exist runs } \rho, \tau \text{ such that } \sigma \simeq \rho, \rho <_l \tau, \text{and}$$
$$\tau, n, \xi \models F[l/m] \text{ for a name } l \text{ that does not occur in } \sigma \text{ or } F$$

We illustrate this refinement principle by an implementation that combines virtualisation and operation refinement to non-atomically move the shopping agent between network nodes via an intermediary location $\mathit{transit} \notin \mathit{Net}$. (A subsequent application of spatial extension would allow that location to be refined into sub-locations to model movement across several network hops.)

Figure 8 contains the specification of a shopping agent that, starting at any location $n \in \mathit{Net}$, first moves to the intermediary location $\mathit{transit}$ before moving on to some other location $m \in \mathit{Net}$. Observe that the implication

$$\mathit{SlowShopper} \Rightarrow \mathit{Shopper}$$

is not valid because $\mathit{SlowShopper}$ does not satisfy the invariant that the shopping agent is always located at some location $n \in \mathit{Net}$. However, we do have

$$\models \mathit{SlowShopper} \Rightarrow \exists\, \mathit{shopper} : \mathit{Shopper}$$

$$\begin{array}{rcl}
\mathit{StartMove}_n & \equiv & \land\ n.\mathit{shopper}\langle \textbf{true} \rangle \land \mathit{shopper.ctl} = \text{"shopping"} \\
 & & \land\ n.\mathit{shopper} \gg \mathit{transit.shopper} \\
\mathit{EndMove}_m & \equiv & \land\ \mathit{transit.shopper}\langle \textbf{true} \rangle \\
 & & \land\ \mathit{transit.shopper} \gg m.\mathit{shopper} \\
\mathit{SlowShopper} & \equiv & \land\ \mathit{Init} \\
 & & \land\ \Box \bigwedge_{m,n \in \mathit{Net} \cup \{\mathit{transit}\}} n\langle m[\textbf{false}]\rangle \\
 & & \land\ \Box\big[\mathit{joe}[\mathit{JoeActions}] \lor \bigvee_{n \in \mathit{Net}} n[\mathit{GetOffer}]\big]_{\mathit{vars}} \\
 & & \land\ \bigwedge_{n \in \mathit{Net}} \Box[\mathit{StartMove}_n]_{-n.\mathit{shopper}} \\
 & & \land\ \Box\big[\bigvee_{m \in \mathit{Net}} \mathit{EndMove}_m\big]_{-\mathit{transit.shopper}}
\end{array}$$

Fig. 8. Shopping agent with non-atomic moves

To see why that implication is valid, consider any run σ of $\mathit{SlowShopper}$. We have to extend the configurations of σ by a new location, say $\mathit{virtual}$, that indicates the current location of the $\mathit{shopper}$ agent of the original specification. Whenever the low-level $\mathit{shopper}$ agent is located at some node $n \in \mathit{Net}$, the same should be true of $\mathit{virtual}$. When $\mathit{shopper}$ is located at $\mathit{transit}$ in between transitions $\mathit{StartMove}_n$ and $\mathit{EndMove}_m$, the location $\mathit{virtual}$ should remain below location n, effectively delaying the high-level move action until the slow shopper arrives at its destination. At every configuration, the local state at location $\mathit{virtual}$ should be that of the slow shopper.

Proofs of refinements by virtualisation can be based on the rule

$$F[n/m] \Rightarrow \exists\, m : F \quad \text{where } n \text{ does not occur in } F$$

However, this "refinement mapping" rule would have to be complemented by a rule for introducing "spatial history variables" [1] in order to prove that the specification $\mathit{Shopper}$ is refined by $\mathit{SlowShopper}$, since the location of the shopper prior to the last $\mathit{StartMove}$ transition has to be remembered in order to compute the location of the witness $\mathit{virtual}$.

Refinement by virtualisation allows more radical refinements than that of $\mathit{Shopper}$ by $\mathit{SlowShopper}$. For example, the formula $\mathit{Shopper}$, which employs a mobile agent, could be refined by a client-server solution that replaces mobility by communication. The proof idea would then be to place the virtual shopping agent at the node from which an offer is received, and to enforce additional stuttering transitions to simulate the Move actions. On the other hand, an implementation might use a swarm of agents instead of a single one.

5 Conclusion

We have defined an extension MTLA of Lamport's TLA by spatial modalities. The logic is intended for the specification of systems that exhibit mobility of agents across hierarchical domains. We have also considered different principles of refinement of mobile systems, focussing on refinements that change the spatial structure of the original specification. We have demonstrated that all these principles can be represented in MTLA by implication, possibly after appropriate hiding of state components or entire agent hierarchies from the interface of the specification. In particular, transitivity and compositionality of refinement, expressed by the rules

$$\frac{S_1 \Rightarrow \exists\, x : S_2 \quad S_2 \Rightarrow \exists\, y : S_3}{S_1 \Rightarrow \exists\, x, y : S_3} \qquad\qquad \frac{S_1 \Rightarrow \exists\, x : S_2}{S_1 \wedge S_3 \Rightarrow (\exists\, x : S_2) \wedge S_3}$$

are immediate consequences of standard propositional and quantifier rules that hold for MTLA. This should make MTLA a sound basis for the stepwise and compositional development of mobile systems.

More generally, we believe that "extensional" semantics such as ours provide a useful complement to existing "intensional" formalisms for mobile systems. Obviously, our definitions will have to be complemented by deductive verification rules to allow formal, syntactic verification of the properties and refinements that we have claimed of our examples. We also want to study the decidability of the model checking problem for our logic, applied to finite-state systems.

References

1. Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 81(2):253–284, May 1991.
2. Luis Caires and Luca Cardelli. A spatial logic for concurrency (part I). In *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science, pages 1–37. Springer-Verlag, 2001. Revised version to appear in Information and Computation.
3. Luca Cardelli and Andrew Gordon. Anytime, anywhere. Modal logics for mobile ambients. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 365–377. ACM Press, 2000.
4. Luca Cardelli and Andrew Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.
5. Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the Join-calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, January 1996. ACM.
6. Leslie Lamport. How to write a long formula. Research Report 119, Digital Equipment Corporation, Systems Research Center, December 1993.
7. Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
8. Stephan Merz. A more complete TLA. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99: World Congress on Formal Methods*, volume 1709 of *Lecture Notes in Computer Science*, pages 1226–1244, Toulouse, September 1999. Springer-Verlag.
9. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.
10. R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Trans. on Software Engineering*, 24(5):315–330, 1998.
11. R. De Nicola and M. Loreti. A modal logic for Klaim. In T. Rus, editor, *Proc. Algebraic Methodology and Software Technology (AMAST 2000)*, volume 1816 of *Lecture Notes in Computer Science*, pages 339–354, Iowa, 2000. Springer-Verlag.
12. Davide Sangiorgi. Extensionality and intensionality of the ambient logic. In *Proc. of the 28th Intl. Conf. on Principles of Programming Languages (POPL'01)*, pages 4–17. ACM Press, 2001.
13. Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In *ICCL Workshop: Internet Programming Languages*, pages 47–77, 1998.