Specification and Refinement of Mobile Systems in MTLA and Mobile UML [⋆]

Alexander Knapp [a] Stephan Merz [b,*] Martin Wirsing [a]
Júlia Zappe [a,b]

[a] *Institut für Informatik, Ludwig-Maximilians-Universität, Munich, Germany*
[b] *INRIA Lorraine & LORIA, Nancy, France*

Abstract

We define the spatio-temporal logic MTLA as an extension of Lamport's Temporal Logic of Actions TLA for the specification, verification, and formal development of systems that rely on mobile code. The formalism is validated by an encoding of models written in the Mobile UML notation. We identify refinement principles for mobile systems and justify refinements of Mobile UML state machines with the help of the MTLA semantics.

Key words: mobile systems, temporal logic, spatial logic, specification, verification, refinement, UML, system development

1 Introduction

Advances in networking technology have enabled novel paradigms of design for software-intensive systems, based on the transmission of mobile code for execution at remote sites rather than the more conventional, communication-based architectures such as client-server systems. It has quickly become apparent that the design of mobile systems requires specific abstractions that should be supported by formal methods of system development and their underlying

⋆ We gratefully acknowledge support by the EC 5th Framework project AGILE: Architectures for Mobility (IST-2001-32747) and by a grant from CCUFB. The stay of Júlia Zappe at LORIA has been partly financed by a grant from Région Lorraine.
* Corresponding author.
Email addresses: `Alexander.Knapp@ifi.lmu.de` (Alexander Knapp), `Stephan.Merz@loria.fr` (Stephan Merz), `Martin.Wirsing@ifi.lmu.de` (Martin Wirsing), `Julia.Zappe@ifi.lmu.de` (Júlia Zappe).

calculi and logics. Milner's π-calculus [1] has been the first to address mobility of names, later foundational calculi for mobile systems [2,3,4,5], have emphasized different aspects of mobility and have defined 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 delimit code mobility.

Some of these calculi have been complemented by logics in which run-time properties of mobile systems can be expressed [6,7,8]. Formulas of these logics are evaluated over process terms by means of an *intensional* semantics [9] and closely reflect the syntactic structure of processes. Although well suited for the verification of properties of processes, these logics are not intended to support notions of refinement that are flexible enough to allow for nontrivial development steps while preserving properties that have been established at abstract levels of description.

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 runs similar to standard (linear-time) temporal logics, independently of any specific operational calculus, and that can support appropriate notions of refinement. We base our logic on Lamport's Temporal Logic of Actions TLA [10], which provides a formal basis for the refinement of reactive systems, but add spatial modalities for describing the topology of system configurations.

The development of systems based on mobile code has also inspired research into appropriate semi-formal notations to software design. In particular, Baumeister et al. [11,12,13] have identified a set of UML [14] stereotypes that apply to class, interaction, state machine, and activity diagrams describing mobile systems. We validate the expressiveness of MTLA by encoding in it a restricted class of mobile UML state machines. We then discuss concepts of refinement that apply to the development of mobile systems. Besides standard operation refinement, we identify two refinement principles that are more specific to mobile systems, namely *spatial extension* and *virtualisation of locations*. These principles support modifications of the spatial structure of mobile systems; they require to clearly delimit the externally visible interface of a specification. We justify refinement principles for mobile UML state machines based on their MTLA semantics, establishing proof obligations that can be read off the UML model without a need for explicit reasoning in MTLA.

The remainder of this article is structured as follows: Sect. 2 introduces the logic MTLA. The encoding of mobile UML state machines is presented in Sect. 3. Section 4 describes our refinement principles and applies them to mobile UML state machines. To illustrate our concepts we use a running example of a mobile shopping agent that roams a network in search for offers for a given item and, upon return to its home site, presents the offers it has collected.

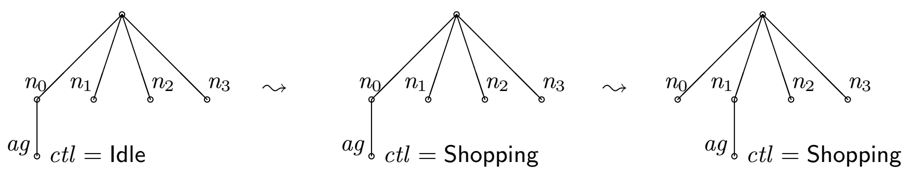


Figure 1. Prefix of a run.

2 The Logic MTLA

2.1 Configurations and runs

We represent the spatial structure of a mobile system at any given instant as a finite tree t whose nodes are labeled with unique ("physical") names n drawn from a denumerably infinite set N, as illustrated in Fig. 1. Reflecting the intended interpretation, we interchangeably refer to the elements of N as "names" or "locations". The root node, which represents the top-level (or "system") domain, is labeled with the implicit label $\varepsilon \notin \mathsf{N}$. Every node of the tree (including the root node) is endowed with a local state, represented as a valuation of a set $\mathcal{V}_f$ of flexible (or "state") variables.

Technically, a tree t is presented as a strict partial order $(\mathsf{N}_t, \prec_t)$ over a finite set $\mathsf{N}_t \subset \mathsf{N}$. We write N_t^ε for the set $\mathsf{N}_t \cup \{\varepsilon\}$, and extend the tree order $\prec_t$ to N_t^ε by letting $n \prec_t \varepsilon$ for all $n \in \mathsf{N}_t$. We write $m \preceq_t n$ if $m \prec_t n$ or $m = n$. For a tree $t = (\mathsf{N}_t, \prec_t)$ and a name $n \in \mathsf{N}_t^\varepsilon$, we write $t{\downarrow}n$ for the subtree of t rooted at the (unique) node labelled by n. For $n \notin \mathsf{N}_t^\varepsilon$, we let $t{\downarrow}n$ denote the empty tree.

A *configuration* is a pair (t, λ): for every node $n \in \mathsf{N}_t^\varepsilon$, the local state assigns a value $\lambda(n, v)$ to each variable $v \in \mathcal{V}_f$. A *run* of a system, illustrated in Fig. 1, is represented as an ω-sequence $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ of configurations. Transitions may change the local state at some nodes, but also modify the tree structure; such structural changes represent the movement of locations or the creation or destruction of locations. For a run $\sigma = (t_0, \lambda_0)(t_1, \lambda_1)\ldots$ and $i \in \mathbb{N}$, we denote by $\sigma|_i$ the suffix $(t_i, \lambda_i)(t_{i+1}, \lambda_{i+1}), \ldots$.

2.2 Simple MTLA

The connectives of MTLA extend classical logic by spatial and temporal modalities. We assume given a signature of first-order logic with equality, and denumerable sets $\mathcal{V}_r$ and $\mathcal{V}_f$ of rigid and flexible individual variables. We want to ensure that formulas of MTLA are invariant under "stuttering" as in Lamport's TLA [10]. Following [15], we introduce auxiliary classes of "impure" terms and formulas that may contain top-level occurrences of the next-time operator $\circ$. Impure formulas may contain temporal operators and thus generalize the action formulas of TLA. In the grammar below, $x \in \mathcal{V}_r$ and $v \in \mathcal{V}_f$ represent rigid and flexible variables, f and P are (k-ary) function and predicate symbols, $m \in \mathsf{N}$ is a name, t and u (possibly with subscripts) are pure and impure terms, and F and A are pure and impure formulas, while S represents pure spatial formulas: pure formulas that do not contain any temporal operators ($\circ$, $\Box$, and its variants). The syntax is ambiguous because we use the same connectives for pure and impure formulas; this ambiguity is harmless as we will define the semantics of these connectives in the same way for pure and impure formulas.

$$
\begin{aligned}
t &::= x \mid v \mid f(t_1, \ldots, t_k) \mid \iota x : F \\
u &::= t \mid f(u_1, \ldots, u_k) \mid \iota x : A \\
F &::= P(t_1, \ldots, t_k) \mid \textbf{false} \mid F_1 \Rightarrow F_2 \mid \exists x : F \mid m[F] \mid \Box F \mid \Box[A]_t \mid \Box[A]_S \\
A &::= F \mid P(u_1, \ldots, u_k) \mid A_1 \Rightarrow A_2 \mid \exists x : A \mid m[A] \mid \circ F
\end{aligned}
$$

Our term formation rules include the definite description operator $\iota x : F$ ("the x such that F" [16]). The only spatial modalities of simple MTLA are of the form $m[\cdot]$ ("$\cdot$ at m") for names m, and they can be applied to pure and impure formulas. The temporal formulas $\Box F$ ("always F") and $\Box[A]_t$ ("always square A sub t") are as in TLA. The latter formula asserts that all transitions that change the value of the (pure) term t satisfy the impure formula A. Similarly, $\Box[A]_S$ asserts that every transition that changes the truth value of S satisfies A. Observe that "impurity" is introduced by the next-time operator $\circ$, and that impure formulas are "purified" by applications of $\Box[A]_t$ or $\Box[A]_S$.

The semantics of MTLA is based on a first-order interpretation $\mathcal{I}$ that provides a non-empty universe $|\mathcal{I}|$, a "null" value $d_{\mathcal{I}} \in |\mathcal{I}|$, and interpretations $\mathcal{I}(f)$ and $\mathcal{I}(P)$ of the function and predicate symbols f and P of the signature where "=" is interpreted as equality on $|\mathcal{I}|$. Terms and formulas are evaluated over a run $\sigma = ((\mathsf{N}_0, \prec_0), \lambda_0)((\mathsf{N}_1, \prec_1), \lambda_1)\ldots$ whose valuations λ_i interpret the flexible variables, at a position $n \in \mathsf{N}_i^{\epsilon}$ that indicates the "location of evaluation", and with respect to a valuation ξ of the rigid variables. We write $\sigma^{(n,\xi)}(t)$ for the value denoted by the term t and write $\sigma, n, \xi \models A$ if formula A holds true for σ, n, and ξ. The formal inductive definition appears in Fig. 2; as every pure term and formula is also an impure one, we only give one clause for the operators that apply to both classes of formulas.

The spatial modalities $m[\cdot]$ shift the spatial focus of evaluation; they are weak in the sense that the formula is trivially true if m does not occur below the current location. More importantly, they refer to nodes at arbitrary nesting depth, and not just the immediate subnodes of the current location. The semantics of the temporal operators is standard but takes into account that

$$\begin{aligned}
\sigma^{(n,\xi)}(x) &= \xi(x) \quad \text{for } x \in \mathcal{V}_r \\
\sigma^{(n,\xi)}(v) &= \left\{\begin{array}{ll} \lambda_0(n,v) & \text{if } n \in \mathsf{N}_0^\varepsilon \\ d_{\mathcal{I}} & \text{otherwise} \end{array}\right\} \quad \text{for } x \in \mathcal{V}_f \\
\sigma^{(n,\xi)}(f(t_1,\ldots,t_k)) &= \mathcal{I}(f)(\sigma^{(n,\xi)}(t_1),\ldots,\sigma^{(n,\xi)}(t_k)) \\
\sigma^{(n,\xi)}(\iota x : A) &= \begin{cases} d \in |\mathcal{I}| & \text{if } \sigma,n,\xi[x:=d] \models A \text{ and} \\ & \quad \sigma,n,\xi[x:=e] \not\models A \text{ for all } e \in |\mathcal{I}| \setminus \{d\} \\ d_{\mathcal{I}} & \text{otherwise} \end{cases}
\end{aligned}$$

$\sigma,n,\xi \models P(t_1,\ldots,t_k)$ iff $(\sigma^{(n,\xi)}(t_1),\ldots,\sigma^{(n,\xi)}(t_k)) \in \mathcal{I}(P)$

$\sigma,n,\xi \not\models$ **false**

$\sigma,n,\xi \models A \Rightarrow B$ iff $\sigma,n,\xi \not\models A$ or $\sigma,n,\xi \models B$

$\sigma,n,\xi \models \exists x : A$ iff $\sigma,n,\xi[x:=d] \models A$ for some $d \in |\mathcal{I}|$

$\sigma,n,\xi \models m[A]$ iff $m \prec_0 n$ implies $\sigma,m,\xi \models A$

$\sigma,n,\xi \models \Box F$ iff for all $i \in \mathbb{N}$, $n \notin \mathsf{N}_j^\varepsilon$ for some $j \leq i$ or $\sigma|_i,n,\xi \models F$

$\sigma,n,\xi \models \circ F$ iff $n \notin \mathsf{N}_1^\varepsilon$ or $\sigma|_1,n,\xi \models F$

$\sigma,n,\xi \models \Box[A]_t$ iff for all $i \in \mathbb{N}$, $n \notin \mathsf{N}_j^\varepsilon$ for some $j \leq i$
or $\sigma|_i^{(n,\xi)}(t) = \sigma|_{i+1}^{(n,\xi)}(t)$ or $\sigma|_i,n,\xi \models A$

$\sigma,n,\xi \models \Box[A]_S$ iff for all $i \in \mathbb{N}$, $n \notin \mathsf{N}_j^\varepsilon$ for some $j \leq i$
or $(\sigma|_i,n,\xi \models S$ iff $\sigma|_{i+1},n,\xi \models S)$ or $\sigma|_i,n,\xi \models A$

Figure 2. Semantics of simple MTLA.

life spans of a name may be finite: for example, the $\Box$ operators only extend for as long as the current name of evaluation is valid. In particular, we consider a later reappearance of a name n to be unrelated to any earlier occurrences of n. We say that F holds of σ and ξ, written $\sigma,\xi \models F$ iff F holds at the root location, i.e. $\sigma,\varepsilon,\xi \models F$. Formula F is *valid*, written $\models F$, iff $\sigma,\xi \models F$ holds for all runs σ and valuations ξ.

When writing MTLA formulas, we use many derived operators, beyond the standard abbreviations **true**, $\wedge$, $\vee$, $\Leftrightarrow$, and $\forall$. For a pure term t, we define the impure term $t' \equiv \iota x : \circ(t = x)$ to denote the value of t at the next instant; t' denotes the null value if the current name of evaluation is invalid for the next state. Similarly, for an (im)pure term t and a name $n \in \mathsf{N}$, $n.t$ denotes the (im)pure term $\iota x : n[x = t]$ that denotes the value of t at sublocation n or the null value if no such location exists. For pure terms $t_1,\ldots,t_n$ we write UNCHANGED$(t_1,\ldots,t_n)$ to denote the impure formula $t_1' = t_1 \wedge \ldots \wedge t_n' = t_n$. We write $[A]_t$ for $A \vee t' = t$ and $[A]_S$ for $A \vee (\circ S \Leftrightarrow S)$.

The formula $n\langle A\rangle \equiv \neg n[\neg A]$ is defined as the dual of $n[A]$; it requires the existence of a sublocation n such that A holds at n. To reduce the number of brackets, we write $n_1.\cdots.n_k[F]$ instead of $n_1[\cdots n_k[F]\cdots]$, and similarly define $n_1.\cdots.n_k\langle F\rangle$. We also sometimes write $n_1.\cdots.n_k$ instead of $n_1.\cdots.n_k\langle\mathbf{true}\rangle$, asserting the existence of nested locations $n_1,\ldots,n_k$ in the current tree.

The formula $\Diamond F$ ("eventually F") is defined as $\neg\Box\neg F$; it requires that F holds 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) changes value during a transition satisfying A. The formulas $\Box[A]_{-S}$ and $\Box[A]_{+S}$ abbreviate $\Box[S \Rightarrow A]_S$ and $\Box[\neg S \Rightarrow A]_S$; they assert that A holds whenever the spatial formula S becomes false (resp., true) during a transition. Finally, the formula $\Box[A]_{a_1,\ldots,a_n}$ (where the a_i may be pure terms or pure spatial formulas) denotes $\Box[A]_{a_1} \wedge \ldots \wedge \Box[A]_{a_n}$; it holds of σ provided every transition that changes some a_i satisfies A.

2.3 Example

A first specification of the shopping agent example, written in simple MTLA, appears as formula *SimpleShopper* in Fig. 3. We adopt Lamport's convention [17] of writing multi-line conjunctions and disjunctions as lists whose items are labeled with the respective connective, relying on indentation to suppress parentheses.

We assume a fixed, finite set *Loc* of (immobile) network locations; $home \in Loc$ denotes the home location of the shopping agent. The name $ag \notin Loc$ refers to the shopping agent itself. The specification *SimpleShopper* consists of four conjuncts: formula *Network* requires all locations $n \in Loc$ to be always present without being nested. The second conjunct *Init* expresses the initial condition: the agent ag should be beneath its home location *home*, and the control state should be "idle". The third conjunct specifies the possible state transitions that affect the local state of the agent. At the home location, the shopper can be seeded for a tour through the network or it can present its results. At any location (even including the home location), it can collect offers for the item it is looking for from the database *supply* of that location. Finally, the fourth conjunct describes the possible moves of an agent: whenever the formula $n.ag\langle\mathbf{true}\rangle$ becomes false for some $n \in Loc$, the agent must be in "shopping" state, and it must move to some other location $m \in Loc$ without changing its local variables. Formula *SimpleShopper* describes the safety part of the specification. Liveness and fairness properties can also be expressed in MTLA, just as in TLA, but we concentrate on safety properties in this article.

Because MTLA is a logic, it can also be used to formulate correctness proper-

$$
\begin{array}{rcl}
\mathit{Network} & \equiv & \Box \bigwedge_{n,m \in \mathit{Loc}} n\langle m[\mathbf{false}]\rangle \\
\mathit{Init} & \equiv & \mathit{home}.\mathit{ag}\langle \mathit{ctl} = \text{“idle”}\rangle \\
\mathit{Look}(x) & \equiv & \wedge\ \mathit{ag}\langle\mathbf{true}\rangle \wedge \circ\mathit{ag}\langle\mathbf{true}\rangle \\
 & & \wedge\ \mathit{ag}.\mathit{ctl} = \text{“idle”} \wedge \mathit{ag}.\mathit{ctl}' = \text{“shopping”} \\
 & & \wedge\ \mathit{ag}.\mathit{lookFor}' = x \wedge \mathit{ag}.\mathit{offers}' = \emptyset \\
\mathit{Offer} & \equiv & \wedge\ \mathit{ag}\langle\mathbf{true}\rangle \wedge \circ\mathit{ag}\langle\mathbf{true}\rangle \\
 & & \wedge\ \mathit{ag}.\mathit{ctl} = \text{“shopping”} \wedge \mathit{ag}.\mathit{lookFor} \in \mathit{items}(\mathit{supply}) \\
 & & \wedge\ \mathit{ag}.\mathit{offers}' = \mathit{ag}.\mathit{offers} \cup \mathit{getOffers}(\mathit{supply}, \mathit{ag}.\mathit{lookFor}) \\
 & & \wedge\ \text{UNCHANGED}(\mathit{ag}.\mathit{ctl}, \mathit{ag}.\mathit{lookFor}) \\
\mathit{Present} & \equiv & \wedge\ \mathit{ag}\langle\mathbf{true}\rangle \wedge \circ\mathit{ag}\langle\mathbf{true}\rangle \\
 & & \wedge\ \mathit{ag}.\mathit{ctl} = \text{“shopping”} \\
 & & \wedge\ \mathit{ag}.\mathit{ctl}' = \text{“idle”} \wedge \mathit{found}' = \mathit{ag}.\mathit{offers} \\
\mathit{Move}_{n,m} & \equiv & \wedge\ n.\mathit{ag}\langle\mathbf{true}\rangle \wedge \mathit{ag}.\mathit{ctl} = \text{“shopping”} \wedge \circ m.\mathit{ag}\langle\mathbf{true}\rangle \\
 & & \wedge\ \text{UNCHANGED}(\mathit{ag}.\mathit{ctl}, \mathit{ag}.\mathit{lookFor}, \mathit{ag}.\mathit{offers}) \\
\mathit{vars} & \equiv & \mathit{ag}.\mathit{ctl}, \mathit{ag}.\mathit{lookFor}, \mathit{ag}.\mathit{offers} \\
\mathit{HomeActions} & \equiv & (\exists x : \mathit{Look}(x)) \vee \mathit{Present} \\
\mathit{SimpleShopper} & \equiv & \wedge\ \mathit{Network} \\
 & & \wedge\ \mathit{Init} \\
 & & \wedge\ \Box\Big[\mathit{home}[\mathit{HomeActions}] \vee \bigvee_{n \in \mathit{Loc}} n[\mathit{Offer}]\Big]_{\mathit{vars}} \\
 & & \wedge\ \bigwedge_{n \in \mathit{Loc}} \Box\Big[\bigvee_{m \in \mathit{Loc}} \mathit{Move}_{n,m}\Big]_{-n.\mathit{ag}}
\end{array}
$$

Figure 3. Specification of a simple shopping agent.

ties. For example, the following invariants hold of specification *SimpleShopper*:

$$\Box \bigvee_{n \in \mathit{Loc}} n.\mathit{ag}\langle\mathbf{true}\rangle \tag{1}$$

$$\Box(\mathit{ag}.\mathit{ctl} = \text{“idle”} \Rightarrow \mathit{home}.\mathit{ag}\langle\mathbf{true}\rangle) \tag{2}$$

Invariant (1) asserts that the shopping agent never leaves the network: it is always situated beneath some location $n \in \mathit{Loc}$. The second invariant states that the control state of the agent can be “idle” only when the agent is at its home location. In general, as in TLA, specification *Spec* satisfies a property *Prop* if and only if the implication $\mathit{Spec} \Rightarrow \mathit{Prop}$ is valid. System properties can thus be verified deductively: a complete axiomatization of propositional MTLA appears in [18]. For the verification of finite-state systems, one can alternatively use model checking procedures that are obtained as natural extensions of those for TLA and other linear-time temporal logics.

2.4 Freeze and Move

Formulas of simple MTLA only restrict the behavior of names and of local state variables that occur explicitly. It can sometimes be useful to constrain parts of the spatial configuration whose names are not (yet) known. For example, the action $Move_{n,m}$ of Fig. 3 describes that location ag moves from n to m, but it does not assert anything of the locations that may be nested beneath ag. A stronger specification would assert that the entire subtree rooted at ag moves from n to m; in fact, such primitives are common in calculi for mobile processes such as the Ambient calculus [2]. An important difference between these specifications arises when considering refinement (addressed in Sect. 4): the *SimpleShopper* specification allows for refinements where the main agent ag moves while leaving some sub-agent behind, whereas a specification based on the stronger move action requires all sub-agents to move along with ag.

We first extend simple MTLA by impure formulas $\mathbf{freeze}_m$, for $m \in \mathsf{N}$, whose semantics is given by

$$\sigma, n, \xi \models \mathbf{freeze}_m \quad \text{iff} \quad t_0{\downarrow}n.m = t_1{\downarrow}n.m$$

In other words, $\mathbf{freeze}_m$ holds if either the name m does not occur below n in the first two configurations of σ, or if it occurs below n in both configurations and the subtrees rooted at m have the same shape. For a name $m \in \mathsf{N}$ and sequences $\alpha, \beta \in \mathsf{N}^*$, we then define the derived impure formula

$$\alpha.m \gg \beta.m \quad \equiv \quad \alpha.m\langle\mathbf{true}\rangle \wedge \circ\beta.m\langle\mathbf{true}\rangle \wedge \mathbf{freeze}_m$$

asserting that the subtree rooted at location m, initially located below α, moves below β while preserving its spatial structure.

2.5 Hiding of State Variables

Designating the externally visible interface of components is an important part of system design: it is achieved by hiding the private state variables of a component. We represent hiding by existential quantification over flexible variables, at designated locations, and extend the syntax of MTLA formulas, stipulating that $\boldsymbol{\exists}\, m.v : F$ is a pure formula whenever F is a pure formula, $m \in \mathsf{N}^\varepsilon$ is a name, and $v \in \mathcal{V}_f$ is a flexible variable (we write $\boldsymbol{\exists}\, v : F$ for $\boldsymbol{\exists}\, \varepsilon.v : F$).

As in TLA [10], the semantics of quantification over flexible variables is defined in such a way that it preserves invariance of formulas under stuttering, and this is a crucial prerequisite in order to have refinements preserve properties. We formally define *stuttering equivalence* $\simeq$ as the smallest equivalence relation on runs that identifies runs that differ by insertion or removal of duplicate configurations (t_i, λ_i):

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

Extending the corresponding proof for TLA [19], it is straightforward to show that formulas of MTLA as defined in Sect. 2 are invariant w.r.t. stuttering equivalence: for any pure MTLA formula F and any behaviors σ and τ such that $\sigma \simeq \tau$ we have $\sigma, \xi \models F$ if and only if $\tau, \xi \models F$.

We say that two 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* $m \in \mathsf{N}^\varepsilon$, written $\sigma =_{m.v} \tau$ iff $s_i = t_i$ for all $i \in \mathbb{N}$ and $\lambda_i(n, x) = \mu_i(n, x)$ for all $n \in \mathsf{N}^\varepsilon$ and $x \in \mathcal{V}_f$ except if $n = m$ and $x = v$. In other words, the tree structures of the configurations in σ and τ must 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 $\approx_{m.v}$ that contains both $\simeq$ and $=_{m.v}$. We define the semantics of existential quantification over flexible variables by

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

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

Formulas $\boldsymbol{\exists}\, m.v : F$ can be proven using the following variant of the familiar quantifier introduction axiom

$$(\boldsymbol{\exists}\text{-I}) \quad F[t/m.v] \Rightarrow \boldsymbol{\exists}\, m.v : F$$

where t is a pure term and $F[t/m.v]$ denotes the formula obtained from F by substituting all top-level occurrences of v in subformulas $m[A]$ (i.e., those occurrences that are not in the scope of any further spatial modality) by t. More formally, this "localized substitution" is defined inductively with the crucial clauses being

$$(n[F])[t/m.v] \equiv \begin{cases} n\big[F\{t/v\}\big] & \text{if } m = n \\ n\big[F[t/m.v]\big] & \text{otherwise} \end{cases} \qquad (n[F])\{t/v\} \equiv n[F]$$

where $F\{t/v\}$ is an auxiliary substitution operation that replaces t for those occurrences of v that are not in the scope of a spatial modality.

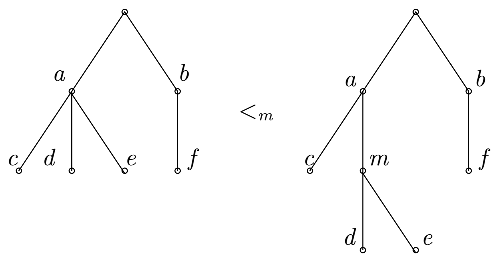


Figure 4. Illustration of tree extension.

2.6 *Hiding of Names*

Beyond the hiding of state variables at components, we introduce the concept of hiding a location (representing an entire component). This can be useful in the specification of a mobile system in order to prevent the environment from relying on the presence of certain sublocations that might not be present in a later implementation. Analogously to the quantification over flexible variables introduced in Sect. 2.5, we introduce an existential quantifier over names: $\boldsymbol{\exists}\, m : F$ is a pure formula if F is a pure formula and $m \in \mathsf{N}$ is a name.

Intuitively, $\boldsymbol{\exists}\, m : F$ holds of a run σ if there exists some run τ that satisfies F and that differs from σ by extending the trees by name m at every configuration. The formal definition is somewhat more complicated because the name used for the extension of the trees should be fresh, and this may require renaming of the bound variable in F. For finite trees $s = (\mathsf{N}_s, \prec_s)$ and $t = (\mathsf{N}_t, \prec_t)$ and a name m we define the relation $s <_m t$ to hold iff s results from t by removing the node labelled by m (if any):

$$s <_m t \quad \text{iff} \quad \mathsf{N}_s = \mathsf{N}_t \setminus \{m\} \text{ and } (a \prec_s b \text{ iff } a \prec_t b \text{ for all } a, b \in \mathsf{N}_s)$$

(see Fig. 4 for an illustration of this definition). 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:

$$(s, \lambda) <_m (t, \mu) \quad \text{iff} \quad s <_m t \text{ and } \lambda(a, v) = \mu(a, v) \text{ for all } a \in \mathsf{N}_s \text{ and } v \in \mathcal{V}_f.$$

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

$$\begin{aligned} \sigma, n, \xi \models \boldsymbol{\exists}\, m : F \quad \text{iff} \quad & \text{there exist runs } \rho, \tau \text{ such that } \sigma \simeq \rho,\ \rho <_l \tau, \text{ and} \\ & \tau, n, \xi \models F[l/m] \text{ for a name } l \text{ that occurs neither in } F \text{ nor in } \sigma. \end{aligned}$$

The existence of a fresh name l is easily ensured by extending the set of names that may occur in τ. Again, this quantifier observes standard proof rules. In particular, we have the introduction axioms

$$\begin{array}{lll} (\exists\text{-ref}) & F[n/m, t_1/m.a_1, \ldots, t_k/m.a_k] \Rightarrow \exists\, m : F & \\ (\exists\text{-sub}) & n\langle \mathbf{true} \rangle \Rightarrow \exists\, m : n.m\langle \mathbf{true} \rangle & (m \neq n) \end{array}$$

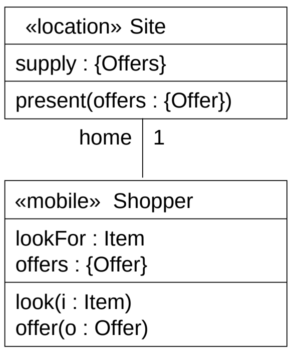
The axiom ($\exists$-ref) is the expected counterpart to the axiom ($\exists$-I) considered in Sect. 2.5. Because the name quantifier hides the location as well as its local state, the axiom calls for an instantiation of the hidden name m as well as of the local variables $a_1, \ldots, a_k$ associated with m. Again, this axiom by itself is incomplete, and in particular the axiom ($\exists$-sub) allows us to introduce a new sublocation n of an existing location m.

3 Formalization of Mobile UML

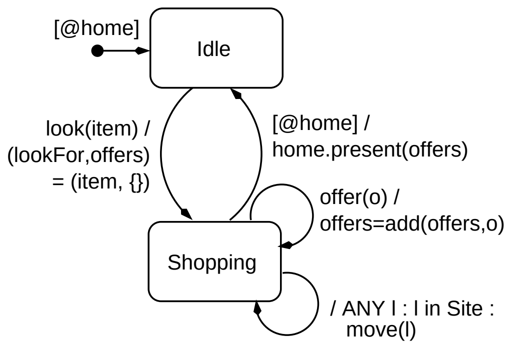
UML has become the de-facto standard notation for object-oriented software development. We validate the expressiveness of MTLA by encoding systems of interacting, mobile UML state machines. The encoding is mechanizable and can be used as a basis for proving properties about systems specified in Mobile UML. Our main interest, however, is in justifying correctness-preserving refinement transformations of mobile UML state machines.

Mobile UML [11,12,13] extends the UML [14] by concepts for modeling mobile computation. The extension is described in terms of the UML itself, using stereotypes and tagged values as meta-modeling tools. Most importantly, instances of classes distinguished by the stereotype «location» denote *locations* where other objects may reside. Mobile objects are instances of classes with the stereotype «mobile» and may change their locations over life-time; cf. Fig. 5(a). As for any UML instance, the behaviour of mobile UML objects can be described by UML state machines, which present an object-oriented variant of Statecharts as defined by Harel [20].

For the purposes of this article, we consider a restricted class of state machines (for a more complete semantical treatment, see, e.g., [21]), but extended by the special move action that causes the object to move to the given target location; see Fig. 5(b). In particular, we exclude hierarchical state machines and pseudo-states, with the exception of a single initial state per state machine. We take into account only events triggered by asynchronous signals (excluding call, time, and change events) and ignore deferred events. Although our encoding could be extended to encompass the full range of features of UML state machines, the simplifications we impose let us concentrate on the problems of mobility and refinement that are our primary concern.



(a) Class diagram.


[@home]
Idle
look(item) / (lookFor,offers) = (item, {})
[@home] / home.present(offers)
offer(o) / offers=add(offers,o)
Shopping
/ ANY l : l in Site : move(l)

(b) State machine for the shopper.

Figure 5. High-level model for the shopper.

3.1 Mobile UML State Machines

A mobile UML state machine consists of a finite set of states and a finite set of transitions between states. A single state is marked as the initial state, depicted by a filled circle; all other states are depicted by rounded rectangles and labelled by a name. The transitions carry labels of the form $\mathit{trig}[\mathit{grd}]/\mathit{act}$, any and all of which can be absent. The trigger *trig* denotes a signal reception of the form $\mathsf{op}(\mathit{par})$ where op is the name of an operation declared in the class and *par* is a list of parameters. The guard *grd* is a Boolean expression over the attributes of the class, the variable self that denotes the object's identity, and the parameters that appear in the trigger clause. In addition, we allow for guards $e_1 \prec e_2$ that refer to the hierarchy of objects; such a clause is true if (the object denoted by) e_1 is currently located beneath e_2.[1] The most common form is $\mathsf{self} \prec e$, requiring the current object to be located below e, which we abbreviate to $@e$. The action *act* denotes the response of an object, beyond the state transition. For simplicity, we assume that all actions are of the form $\mathsf{ANY}\ x : P : \mathit{upd}, \mathit{send}, \mathit{move}$ where each of the constituents may be absent. Herein, P is a predicate over location objects, and $\mathsf{ANY}\ x : P$ functions as a binder that chooses some location object x satisfying P which can be used in the remainder of the action. The *upd* part is a simultaneous assignment $(a_1, \ldots, a_k) = (e_1, \ldots, e_k)$ of expressions e_i to attributes a_i. The *send* part is of the form $e.\mathsf{op}(\mathit{par})$ and denotes the emission of a signal op with parameters *par* to receiver object e. Finally, the *move* part consists of a single $\mathsf{move}(e)$ action that indicates that the object should move to the location object whose identity is denoted by e. We require that all free variables in the action are among the attributes of the class, the parameters introduced by the trigger, the location x bound by ANY, and self.

UML state machines of mobile objects are executed in an environment con-

[1] Again, e_1 and e_2 are expressed in terms of attributes and parameters, in accordance with the encapsulation principle central to object-oriented design.

sisting of a single network for exchanging messages between objects and an event queue for each object. Each object starts in the initial state of its state machine. In any state, if either the trigger of an outgoing transition is at the head of the event queue associated with the object or if an outgoing transition shows no triggering signal, this outgoing transition may be taken, provided its guard is satisfied. If a transition is taken, its action is executed and the state of the object becomes the target state of the transition. On executing the action of the transition, the signals raised by the send part of the action are put into the network. The network delivers its contents to the event queues of the receiving objects.

Figure 5(b) shows a UML state machine for the high-level shopping agent, based on the class diagram of Fig. 5(a). The behaviour of this simple shopping agent intuitively corresponds to the MTLA specification presented in Fig. 3.

Our interpretation of transitions deviates in certain ways from the UML standard. First, the UML standard prioritizes triggerless transitions (so-called "completion transitions") over transitions that require an explicit triggering event. In contrast, we consider that completion transitions may be delayed; this less deterministic interpretation is more appropriate for descriptions at higher levels of abstraction. As a second, minor deviation, we allow guards to appear in transitions leaving a state machine's initial state.

3.2 MTLA *Semantics of State Machines*

For the MTLA encoding, every object in a system of Mobile UML state machines is represented by an MTLA location whose local state includes a unique, unmodifiable identifier *self* containing the object's identity. We denote by *Obj* the set of all MTLA locations that represent objects of a given object system. The subset *Loc* denotes the set of MTLA locations that represent UML ≪location≫ objects (including ≪mobile≫ ≪location≫s), and the formalization of a system of state machines at a given level of abstraction is with respect to these sets *Obj* and *Loc*. An object configuration is represented as a tree of names as described in Sect. 2.

The local state at each node represents the attributes of the corresponding object, including *self*. In addition, we use the attributes *ctl* to hold the current control state of the object (i.e., the active state of the corresponding state machine) and *evts* to represent the list of events that are waiting to be processed by the object. Objects interact asynchronously by sending and receiving messages. In the MTLA formalization, the communication network is represented explicitly by an attribute *msgs* located at the root node of the configuration tree.

Every transition of an object is translated into an MTLA action formula that takes a parameter o denoting the location corresponding to the object. In the following, if φ is an MTLA expression (a term or a formula), we write φ^x and φ_o, respectively, for the expressions obtained by replacing x by $x.\mathit{self}$ and by replacing all attributes a of o by $o.a$.

The action formula representing a transition is a conjunction built from the translations of its trigger, guard, and action components. The automaton transition from states src to dest is reflected by a conjunct $o.\mathit{ctl} = \mathsf{src} \wedge o.\mathit{ctl}' = \mathsf{dest}$.

A trigger $\mathsf{op}(\mathit{par})$ contributes to the definition of the action formula in two ways: first, the parameters par are added to the formal parameters of the action definition. Second, we add the conjunct

$$\neg \mathit{empty}(o.\mathit{evts}) \wedge \mathit{head}(o.\mathit{evts}) = \langle \mathsf{op}, \mathit{par} \rangle \wedge o.\mathit{evts}' = \mathit{tail}(o.\mathit{evts})$$

asserting that the transition can only be taken if the trigger is actually present in the event queue and that it is removed from the queue upon execution of the transition. For transitions without an explicit trigger we add the conjunct $\text{UNCHANGED}(o.\mathit{evts})$ to indicate that the event queue is unmodified.

A Boolean guard g over the object's attributes is represented by a formula g_o, indicating that g is true at location o. A constraint $e_1 \prec e_2$ on the hierarchy of objects is represented by a conjunct of the form

$$\bigvee_{o_1, o_2 \in \mathit{Obj}} o_1.\mathit{self} = (e_1)_o \wedge o_2.\mathit{self} = (e_2)_o \wedge o_2.o_1 \langle \mathbf{true} \rangle$$

An action $\mathsf{ANY}\ x : P : \mathit{upd}, \mathit{send}, \mathit{move}$ is translated to an MTLA formula $\bigvee_{x \in \mathit{Loc}} P^x_o \wedge \mathit{acts}^x$ where acts^x is a conjunction of formulas representing the upd, send, and move constituents of the action. In more detail, a multiple assignment to attributes is represented by a formula

$$o.a_1' = (e_1)^x_o \wedge \ldots \wedge o.a_k' = (e_k)^x_o \wedge \text{UNCHANGED}(o.a_{k+1}, \ldots, o.a_n)$$

where $a_{k+1}, \ldots, a_n$ are the attributes of o that are not modified by the assignment and where x is the variable bound by ANY. Sending a message $e.\mathsf{op}(\mathit{par})$ is modeled by a conjunct $\mathit{msgs}' = \mathit{msgs} \cup \{\langle e^x_o, \mathsf{op}, \mathit{par}^x_o \rangle\}$ asserting that the message is added to the set of undelivered messages. For actions that do not send a message we instead add the conjunct $\mathit{msgs}' = \mathit{msgs}$. If the action contains a clause $\mathsf{move}(e)$, we add a conjunct

$$\bigvee_{l \in \mathit{Loc}} l.\mathit{self} = e^x_o \wedge \varepsilon.o \gg l.o$$

that asserts that o will move to (the location with identity) e^x_o. Otherwise we add the conjunct $\bigwedge_{l \in \mathit{Loc}} [\mathbf{false}]_{l.o}$ to indicate that the object does not enter or

$$
\begin{array}{lll}
\mathit{Init}(ag) & \equiv & ag.\mathit{ctl} = \mathsf{Idle} \wedge \bigvee_{l \in \mathit{Loc}} (l.ag\langle \mathbf{true} \rangle \wedge ag.\mathit{home} = l.\mathit{self}) \\
\mathit{Stationary}(ag) & \equiv & \bigwedge_{l \in \mathit{Loc}} [\mathbf{false}]_{l.ag} \\
\mathit{Deq}(ag, \mathit{msg}) & \equiv & \neg \mathit{empty}(ag.\mathit{evts}) \wedge \mathit{head}(ag.\mathit{evts}) = \mathit{msg} \wedge ag.\mathit{evts}' = \mathit{tail}(ag.\mathit{evts}) \\
\mathit{Look}(ag, \mathit{item}) & \equiv & \wedge\ ag.\mathit{ctl} = \mathsf{Idle} \wedge ag.\mathit{ctl}' = \mathsf{Shopping} \wedge \mathit{Deq}(ag, \langle \mathsf{look}, \mathit{item} \rangle) \\
 & & \wedge\ ag.\mathit{lookFor}' = \mathit{item} \wedge ag.\mathit{offers}' = \{\} \wedge \textsc{unchanged}(ag.\mathit{home}) \\
 & & \wedge\ \mathit{msgs}' = \mathit{msgs} \wedge \mathit{Stationary}(ag) \\
\mathit{Offer}(ag, o) & \equiv & \wedge\ ag.\mathit{ctl} = \mathsf{Shopping} \wedge ag.\mathit{ctl}' = \mathsf{Shopping} \\
 & & \wedge\ \mathit{Deq}(ag, \langle \mathsf{offer}, o \rangle) \wedge ag.\mathit{offers}' = \mathit{add}(ag.\mathit{offers}, o) \\
 & & \wedge\ \textsc{unchanged}(ag.\mathit{lookFor}, ag.\mathit{home}) \\
 & & \wedge\ \mathit{msgs}' = \mathit{msgs} \wedge \mathit{Stationary}(ag) \\
\mathit{Present}(ag) & \equiv & \wedge\ \bigvee_{l \in \mathit{Loc}} ag.\mathit{home} = l.\mathit{self} \wedge l.ag\langle \mathbf{true} \rangle \\
 & & \wedge\ ag.\mathit{ctl} = \mathsf{Shopping} \wedge ag.\mathit{ctl}' = \mathsf{Idle} \\
 & & \wedge\ \textsc{unchanged}(ag.\mathit{lookFor}, ag.\mathit{offers}, ag.\mathit{home}, ag.\mathit{evts}) \\
 & & \wedge\ \mathit{msgs}' = \mathit{msgs} \cup \{\langle ag.\mathit{home}, \mathsf{present}, ag.\mathit{offers} \rangle\} \\
 & & \wedge\ \mathit{Stationary}(ag) \\
\mathit{Move}(ag) & \equiv & \bigvee_{l \in \mathit{Loc}} \wedge\ l.\mathit{self} \in \mathit{Loc} \\
 & & \qquad\quad \wedge\ ag.\mathit{ctl} = \mathsf{Shopping} \wedge ag.\mathit{ctl}' = \mathsf{Shopping} \\
 & & \qquad\quad \wedge\ \textsc{unchanged}(ag.\mathit{lookFor}, ag.\mathit{offers}, ag.\mathit{home}, ag.\mathit{evts}) \\
 & & \qquad\quad \wedge\ \mathit{msgs}' = \mathit{msgs} \wedge \varepsilon.ag \gg l.ag \\
\mathit{RcvEvt}(ag, e) & \equiv & \wedge\ \langle ag.\mathit{self}, e \rangle \in \mathit{msgs} \wedge \mathit{msgs}' = \mathit{msgs} \setminus \langle ag.\mathit{self}, e \rangle \\
 & & \wedge\ ag.\mathit{evts}' = \mathit{append}(ag.\mathit{evts}, e) \\
 & & \wedge\ \textsc{unchanged}(ag.\mathit{ctl}, ag.\mathit{lookFor}, ag.\mathit{offers}, ag.\mathit{home}) \\
 & & \wedge\ \mathit{Stationary}(ag) \\
\mathit{DiscEvt}(ag) & \equiv & \wedge\ \neg \mathit{empty}(ag.\mathit{evts}) \wedge ag.\mathit{evts}' = \mathit{tail}(ag.\mathit{evts}) \\
 & & \wedge\ \neg \exists i : \mathit{head}(ag.\mathit{evts}) = \langle \mathsf{look}, i \rangle \vee ag.\mathit{ctl} \neq \mathsf{Idle} \\
 & & \wedge\ \neg \exists o : \mathit{head}(ag.\mathit{evts}) = \langle \mathsf{offer}, o \rangle \vee ag.\mathit{ctl} \neq \mathsf{Shopping} \\
 & & \wedge\ \textsc{unchanged}(ag.\mathit{ctl}, ag.\mathit{lookFor}, ag.\mathit{offers}, ag.\mathit{home}) \\
 & & \wedge\ \mathit{msgs}' = \mathit{msgs} \wedge \mathit{Stationary}(ag) \\
\mathit{Next}(ag) & \equiv & \vee\ (\exists i : \mathit{Look}(ag, i)) \vee (\exists o : \mathit{Offer}(ag, o)) \vee \mathit{Present}(ag) \\
 & & \vee\ \mathit{Move}(ag) \vee (\exists e : \mathit{RcvEvt}(ag, e)) \vee \mathit{DiscEvt}(ag) \\
\mathit{attr}(ag) & \equiv & \langle ag.\mathit{ctl}, ag.\mathit{lookFor}, ag.\mathit{offers}, ag.\mathit{home}, ag.\mathit{evts} \rangle \\
\mathit{IShopper}(ag) & \equiv & \wedge\ \mathit{Init}(ag) \wedge ag.\mathit{evts} = \langle\rangle \wedge \Box[\mathit{Next}(ag)]_{\mathit{attr}(ag)} \wedge \Box[\mathbf{false}]_{ag.\mathit{self}} \\
 & & \wedge\ \bigwedge_{l \in \mathit{Loc}} \Box[\mathit{Next}(ag)]_{l.ag} \wedge \bigwedge_{l \in \mathit{Loc}} \Box ag.l[\mathbf{false}]
\end{array}
$$

Figure 6. MTLA specification of the shopper behavior (see Fig. 5)

leave any location in *Loc*.

To model the reception of new events, we add an action $\mathit{RcvEvt}(o, e)$ that removes an event e addressed to o from the network and appends it to the queue *evts* of unprocessed events while leaving all other attributes unchanged. We also add an action $\mathit{DiscEvt}(o)$ that discards events that do not have asso-

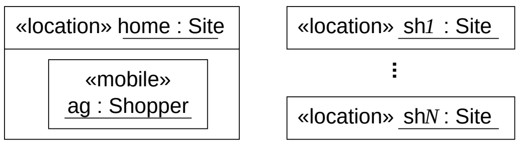


Figure 7. Object diagram for the shopper example.

ciated transitions from the current control state. The entire next-state relation $Next(o)$ of object o is represented as a disjunction of all actions defined from the transitions and the implicit actions $RcvEvt$ and $DiscEvt$, existentially quantifying over all parameters that have been introduced in the translation.

A state predicate $Init(o)$ defining the initial conditions of object o is similarly obtained from the transition(s) leaving the initial state of the state machine. Finally, the overall specification of the behavior of an object o of class C is given by the MTLA formulas

$$\begin{aligned} IC(o) \equiv & \wedge\ Init(o) \wedge o.evts = \langle\rangle \wedge \Box[Next(o)]_{attr(o)} \wedge \Box[\mathbf{false}]_{o.self} \quad (3)\\ & \wedge \textstyle\bigwedge_{l \in Loc} \Box[Next(o)]_{l.o}\ (\wedge \textstyle\bigwedge_{l \in Loc} o.l[\mathbf{false}])\\ C(o) \equiv & \ \exists\ o.ctl, o.evts : IC(o) \quad (4)\end{aligned}$$

The "internal" specification $IC(o)$ asserts that the initial state must satisfy the initial condition, that all modifications of attributes of o (including ctl and $evts$) and all moves of o relative to locations in Loc are accounted for by the next-state relation, and that the object identity is immutable. If C is not a ≪location≫ class, an additional conjunct states that no location object should be nested beneath o. For example, the formula $IShopper(ag)$ shown in Fig. 6 defines the behavior of an object ag of class Shopper introduced in Fig. 5. The "external" specification $C(o)$ is obtained from $IC(o)$ by hiding the implicit attributes $o.ctl$ and $o.evts$.

The specification of a finite system of objects consists of the conjunction of the specifications of the individual objects. Moreover, we add conjuncts that describe the hierarchy of locations and objects and that constrain the network. For our shopper example, we might assume a typical system configuration being given by the object diagram in Fig. 7. This configuration can be translated into the formula

$$\begin{aligned} Sys \equiv \exists\ msgs : & \wedge \textstyle\bigwedge_{i=1}^{N} \wedge\ sh_i\langle self = \mathsf{shop\text{-}}i \wedge home[\mathbf{false}] \wedge \textstyle\bigwedge_{j=1}^{N} sh_j[\mathbf{false}]\rangle\\ & \qquad \wedge\ Site(sh_i)\\ & \wedge\ home\langle self = \mathsf{home} \wedge \textstyle\bigwedge_{i=1}^{N} sh_i[\mathbf{false}]\rangle \wedge Site(home)\\ & \wedge\ home.ag\langle self = \mathsf{ag}\rangle \wedge Shopper(ag)\\ & \wedge \textstyle\bigwedge_{l \in Loc} \Box[\mathbf{false}]_{l.sh_1,\ldots,l.sh_N,l.home}\\ & \wedge\ msgs = \langle\rangle \wedge \Box\Big[\textstyle\bigvee_{o \in Obj} Next(o)\Big]_{msgs}\end{aligned}$$

The formula in the scope of the existential quantifier asserts that the initial configuration contains the $N+1$ sites $sh_1, \ldots, sh_N$ and *home*, and a shopping agent *ag*. (We omit the state machine for the sites and its formalization by the MTLA formula *Site*.) Moreover, *home* and the shops $sh_1, \ldots, sh_N$ are immobile and unnested locations, whereas *ag* is initially situated beneath *home*. The last conjunct asserts that messages are only sent and received according to the specifications of the participating objects. The external specification is obtained by hiding the set of messages in transit, which is implicit at the UML level.

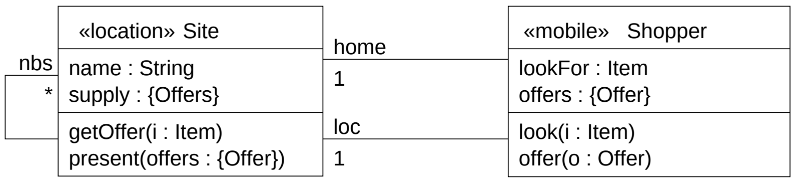
For this example, *Obj* is the set $\{sh_1, \ldots, sh_N, home, ag\}$ and $Loc = Obj \setminus \{ag\}$. Moreover, we define a set *Site* containing the identities of the elements of *Loc*, i.e. $Site = \{\mathsf{shop\text{-}1}, \ldots, \mathsf{shop\text{-}}N, \mathsf{home}\}$.

The MTLA formula obtained from the UML model is somewhat lengthier than the specification *SimpleShopper* of Sect. 2.3, mostly because we have to specify asynchronous message passing for UML models. However, the invariants (1) and (2) considered for *SimpleShopper* still hold of the encoding of the UML system.

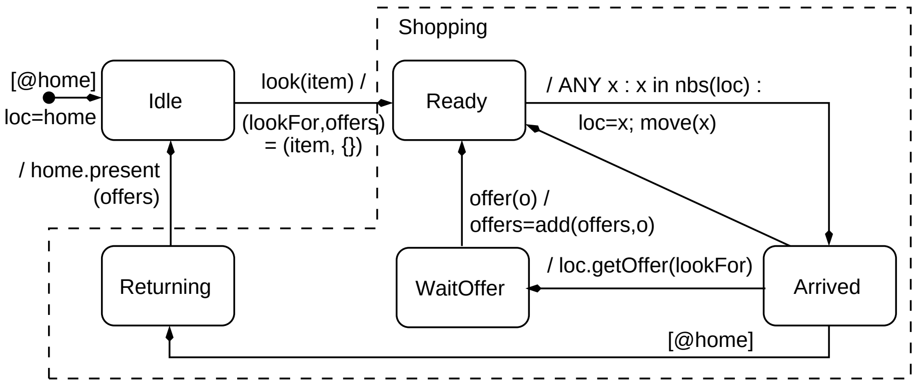
4 Refinement of Mobile UML State Machines

Refinement concepts are the crucial ingredients of any formal method of system development [22,23]. High-level specifications are gradually enriched, preserving already verified properties, until an implementation can be easily read off or automatically generated. Advocated by researchers in academia for several decades, refinement has been extended for object orientation [24] and is increasingly finding its way into industrial development projects [25,26]. We identify and study three principles of refinement for mobile systems:

(1) *Operation refinement* is a principle that is well-known from sequential and reactive systems. It can be used to reduce the non-determinism of operations; actions that are atomic at the abstract level can also 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. It may be necessary to hide local state associated with node n if it is to be distributed among the sub-locations of the implementation.
(3) *Virtualisation of locations* allows to implement one or several locations 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) Class diagram.



(b) State machine.

Figure 8. Refined shopper.

A single refinement step may combine several of these principles. For example, we will see that an atomic move action at a high level of description can be decomposed into a sequence of lower-level moves using a combination of operation refinement and virtualisation. We consider each of the basic principles in more detail 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 h by a low-level specification $Conc$ is expressed by validity of the implication $Conc \Rightarrow \exists\, h : Abs$. In connection with Mobile UML, our main interest is in justifying correctness-preserving refinements of mobile UML state machines without resorting to reasoning about behaviors within MTLA.

4.1 Operation Refinement

Early system models may afford a high degree of non-determinism, which is reduced during system design. For example, consider the state machine for the shopping agent shown in Fig. 8, which imposes a number of constraints with respect to the state machine shown in Fig. 5. After arriving at a new shop location (whose identity is recorded in the additional attribute loc), the agent may now either query for offers by sending a new message getOffer or it may immediately move on to another neighbor location. In the former case, the agent waits until the offers are received, adds them to its local memory, and then moves on. When the agent arrives at its home location, it may quit the cycle, presenting the collected offers and returning to the Idle state.

Intuitively, the state machine of Fig. 8 is a refinement of the one shown in Fig. 5 because the states of the refined state machine can be mapped to those of the high-level state machine such that every transition of the lower-level machine either is explicitly allowed or is invisible at the higher level. In particular, the states Ready, Arrived, WaitOffer, and Returning can all be mapped to the high-level state Shopping, as indicated by the dashed line enclosing these states. Assuming that the set $\mathsf{nbs}(s)$ contains only identities in Site, for all $s \in \mathit{Site}$, each transition of the refined model either corresponds to a transition of the abstract model or to a stuttering transition. For example, the transition from Arrived to WaitOffer is invisible at the level of abstraction of the model shown in Fig. 5. Because MTLA specifications are stuttering invariant, we expect the formalization of a system containing the refined shopper to imply the formula Sys of Sect. 3.2, which describes the original system.

We now formalize this intuition by defining what it means for a state machine R to refine another state machine M for some class C. We have to be more precise about the context in which M and R are supposed to be embedded. The machines are specified with respect to two class diagrams C^R and C^M that describe the attribute and method signatures Σ^R and Σ^M, which include all method names that appear in transition labels (either received or sent), and we assume that Σ^R extends Σ^M. Similarly, we assume that the sets Obj^R and Loc^R of MTLA names for the objects and the Location objects at the level of the refinement are supersets of the corresponding sets Obj^M and Loc^M at the abstract level. Finally, the refinement may be subject to global hypotheses about the refined system, such as the hierarchy of names, that are formally asserted by an MTLA state predicate H. Thus, we say that the state machine R for class C^R refines the state machine M for class C^M under hypothesis H if for all system specifications Sys^M and Sys^R where Sys^R results from Sys^M by replacing all occurrences of $C^M(o)$ by $C^R(o)$ and by conjoining some formulas such that Sys^R implies $\Box H$, the implication $\mathit{Sys}^R \Rightarrow \mathit{Sys}^M$ is valid.

In order to prove that R refines M, we relate the machines by a mapping η that associates with every state s of R a pair $\eta(s) = (\mathit{Inv}(s), \mathit{Abs}(s))$ where $\mathit{Inv}(s)$ is a set of MTLA state predicates, possibly containing spatial operators, and where $\mathit{Abs}(s)$ is a state of M. With such a mapping we associate certain proof obligations: the invariants must be inductive for R, and the (MTLA formalizations of the) transitions of the machine R must imply some transition allowed at the corresponding state of M, or leave unchanged the state of M.

Theorem 1 *Assume that M and R are two state machines for classes C^M and C^R such that the attribute and method signature Σ^R of C^R extends the signature Σ^M of C^M, and that η is a mapping associating with every state s of R a set $\mathit{Inv}(s)$ of* MTLA *state predicates and a state $\mathit{Abs}(s)$ of M. If all of the following conditions hold (with free variable o) then R refines M under* *hypothesis H. We write $\overline{\varphi}$ for*

$$\varphi[Abs(o.ctl)/o.ctl, o.evts\lceil_{\Sigma^M}/o.evts, msgs\lceil_{\Sigma^M}/msgs]$$

where $e\lceil_\Sigma$ denotes the collection (set or sequence) of those elements of e whose first component is in Σ.

(1) $Abs(s_0^R) = s_0^M$ where s_0^M and s_0^R denote the initial states of M and R. Moreover,

$$\models H \wedge Init^R(o) \Rightarrow o[Inv(s_0^R)] \wedge \overline{Init^M}(o)$$

holds for the initial conditions $Init^R$ and $Init^M$ of M and R.

(2) For every transition of R with source and target states s and t formalized by the MTLA *action formula $A(o, par)$:*

$$\models H \wedge H' \wedge o[Inv(s)] \wedge A(o, par) \Rightarrow o[Inv(t)']$$

(3) For every state s of R and every outgoing transition of s formalized by formula $A(o, par)$, let $Abs(s)$ denote the corresponding state of M, let $B_1(o, par_1)$, ..., $B_m(o, par_m)$ be the MTLA *formulas for the outgoing transitions of $Abs(s)$, let $attr^M(o)$ be the tuple of attributes defined for M and Loc^M the set of locations for M. Then:*

$$\begin{aligned}\models H \wedge H' \wedge o[Inv(s)] \wedge A(o, par) \ \Rightarrow\\ &\vee \textstyle\bigvee_{i=1}^m (\exists par_i : \overline{B_i}(o, par_i))\\ &\vee \text{UNCHANGED}(\overline{attr^M}(o), msgs\lceil_{\Sigma^M}) \wedge \textstyle\bigwedge_{l \in Loc_M} [\mathbf{false}]_{l.o}\end{aligned}$$

Proof. Assume that the low- and high-level systems are specified by the MTLA formulas Sys^R and Sys^M. By $ISys^R$ and $ISys^M$, we will denote the corresponding formulas without the quantification over $msgs$ and the implicit attributes $o_i.ctl$ and $o_i.evts$, for all objects o_i. We will prove that

$$\models ISys^R \Rightarrow \overline{\overline{ISys^M}} \tag{5}$$

where we let $\overline{\overline{F}}$ denote the formula

$$F[Abs(o.ctl)/o.ctl, msgs\lceil_{\Sigma^M}/msgs, o_i.evts\lceil_{\Sigma^M}/o_i]$$

— in particular, all event queues, and not just the queue of the refined object o, are restricted to messages in the high-level signature Σ^M.

From (5), we obtain the assertion $\models Sys^R \Rightarrow Sys^M$ by the standard introduction and elimination rules for existential quantification.

For the proof of (5), recall that we assume Sys^R to be obtained from Sys^M by adding (some global assumptions that imply) $\Box H$ and by replacing the specification $C^M(o)$ by $C^R(o)$. Thus, all conjuncts in $ISys^M$ that refer to the global system, and in particular the object hierarchy, are implied by ISys^R, and we only have to consider the specifications of the objects that appear in ISys^M. Formula (5) is shown by a proof of step simulation, considering the actions possible at the implementation level.

First consider any object o_i different from the refined object o. By definition,

$$\models \mathit{ISys}^R \Rightarrow \mathit{Init}^M(o_i) \land o_i.\mathit{evts} = \langle\rangle$$

and therefore it follows that

$$\models \mathit{ISys}^R \Rightarrow \mathit{Init}^M(o_i) \land o_i.\mathit{evts}\upharpoonright_{\Sigma^M} = \langle\rangle$$

Similarly, $\mathit{Next}^R(o_i)$ and $\mathit{Next}^M(o_i)$ as well as $\mathit{attr}^R(o_i)$ and $\mathit{attr}^M(o_i)$ are identical. In particular, all moves (with respect to location in Loc^M) at the implementation level must also be allowed at the abstract level. To see that

$$\models [\mathit{Next}^M(o_i)]_{\mathit{attr}^M(o_i),\mathit{msgs}} \Rightarrow [\overline{\overline{\mathit{Next}^M}}(o_i)]_{\overline{\overline{\mathit{attr}^M(o_i),\mathit{msgs}}}}$$

observe that the definition of every action $A(o_i, \mathit{par})$ other than $\mathit{RcvEvt}(o_i, e)$ or $\mathit{DiscEvt}(o_i)$ is such that

$$\models A(o_i, \mathit{par}) \Rightarrow \overline{\overline{A}}(o_i, \mathit{par})$$

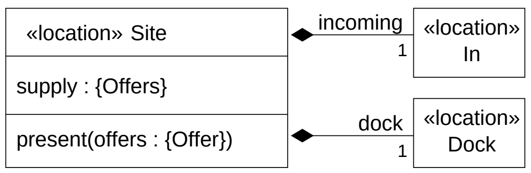
because any message consumed or sent by A is contained in Σ^M, implying that $o_i.\mathit{evts}$ and msgs on the one side and $o_i.\mathit{evts}\upharpoonright_{\Sigma^M}$ and $\mathit{msgs}\upharpoonright_{\Sigma^M}$ on the other side are modified in the same way. On the other hand, executions of $\mathit{RcvEvt}(o_i, e)$ or $\mathit{DiscEvt}(o_i)$ for an event not in Σ^M are mapped to stuttering transitions with respect to $o_i.\mathit{evts}\upharpoonright_{\Sigma^M}$ and $\mathit{msgs}\upharpoonright_{\Sigma^M}$.

Now consider the refined object o itself. Condition (1) ensures that the initial condition $\overline{\overline{\mathit{Init}^M}}(o) \land o.\mathit{evts}\upharpoonright_{\Sigma^M} = \langle\rangle$ holds for any run satisfying ISys^R. Moreover, conditions (1) and (2) inductively establish that $o[\mathit{Inv}(\mathit{ctl})]$ holds along the entire run. Therefore, condition (3) shows that every move of o in a run described by ISys^R either maps to a move allowed by $\overline{\overline{\mathit{ISys}^M}}$ or does not affect the projection of the state visible at the abstract level, i.e.

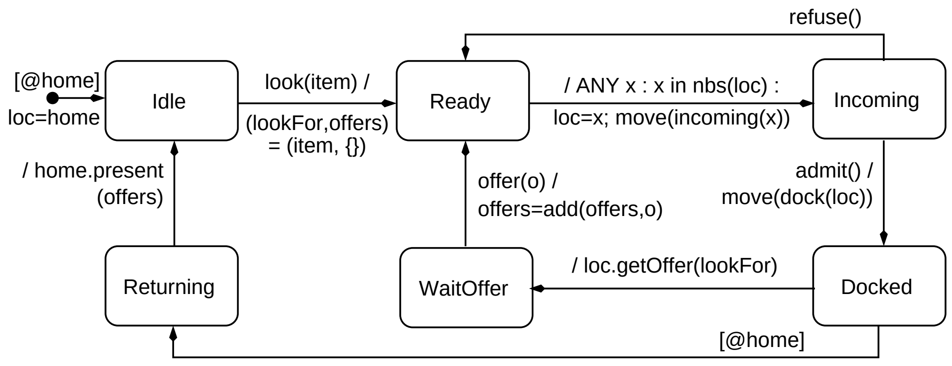
$$\models \mathit{ISys}^R \Rightarrow \Box[\overline{\overline{\mathit{Next}^M}}(o)]_{\overline{\overline{\mathit{attr}}}(o)} \land \Box[\overline{\overline{\mathit{Next}^M}}(o)]_{\overline{\overline{\mathit{msgs}}}} \land \bigwedge_{l \in \mathit{Loc}^M} \Box[\overline{\overline{\mathit{Next}^M}}(o)]_{l.o}$$

This completes the proof of (5), and thus of the theorem. Q.E.D.

Theorem 1 ensures that R can replace M in any system that satisfies hypotheses H. In particular, all properties (MTLA formulas) that have been established for the high-level system are guaranteed to be preserved by the refined model. The proof obligations of Thm. 1 are induced by the elements of the UML state machine rather than by their MTLA encoding, and they



(a) Class diagram.



(b) State machine.

Figure 9. A "docked" shopping agent.

can easily be rewritten in a different logic, such as OCL, because they are non-temporal.

In order to prove that the state machine of Fig. 8 refines that of Fig. 5, with respect to hypothesis $H \equiv \forall s \in Site : \mathsf{nbs}(s) \in Site$, we must define the mapping η. We have already indicated the definition of the state abstraction mapping Abs. For the mapping Inv, we associate (the MTLA encoding of) $@home$ with state Returning and $ag.loc \in Site$ with all other states. It is then easy to verify the conditions of Thm. 1. In particular, the transitions leaving state Arrived do not modify the shopping agent's attributes, and they do not send messages contained in the original signature. They are therefore allowed by condition (3) of Thm. 1. Note that the hypothesis H captures the refinements to the location structure.

4.2 Spatial Extension without Distribution of State

A system developer may choose to refine single locations of the high-level model into hierarchies of locations. However, the spatial relations between the locations that existed at the abstract level of description should be preserved. Consider the model shown in Fig. 9. It is based on the idea that prior to interacting with an object, incoming agents are first placed in a special sublocation for security checking. Instead of a simple, atomic move from one shop to another as in Figs. 5 and 8, this version moves the shopping agent first to the "incoming" sublocation of the target location. If the agent is accepted by the host, as modeled by the reception of an admit signal, it transfers to the "dock" sublocation where the real processing takes place. Otherwise, the host will send a refuse signal, and the shopping agent moves on to another neighbor host. As shown in the class diagram, we assume that every site location contains sublocations In and Dock that are represented by the MTLA sublocations l_in and l_dock.

Theorem 1 can again be used to show that the "docked" shopper of Fig. 9 is a further refinement of the state machine shown in Fig. 8 with respect to the hypothesis

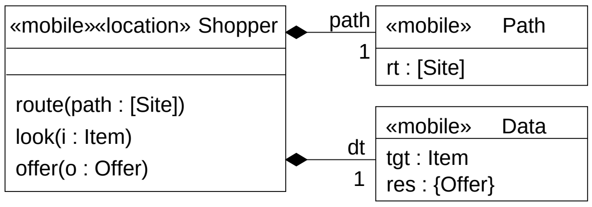
$$H \equiv \bigwedge_{l \in Loc^R} \begin{array}{l} \wedge\ l.l\_in\langle\mathbf{true}\rangle \wedge l.l\_dock\langle\mathbf{true}\rangle \\ \wedge\ \mathsf{incoming}(l.self) = l\_in.self \wedge \mathsf{dock}(l.self) = l\_dock.self \end{array}$$

that asserts the relationship between sites and their sublocations and the connection with the associations of the class diagram in Fig. 9(a). The states Incoming and Docked of the state machine of the "docked" shopper are mapped to the single high-level state Arrived, and the invariant mapping associates (the MTLA encoding of) @loc with the location Incoming and $ag.loc \in Site$ with all states. Indeed, the move action labeling the transition from Ready to Incoming will be formalized by an MTLA action formula $\bigvee_{l \in Loc^R} \varepsilon.ag \gg l\_in.ag$, which, by virtue of the hypothesis H, implies the corresponding formula $\bigvee_{l \in Loc^M} \varepsilon.ag \gg l.ag$ formalizing the move between the high-level states Ready and Arrived. Similarly, H and the invariant establish that the move between the Incoming and Docked states maps to a stuttering action: first, the local attributes and the message queue are left unchanged. Moreover, the invariant associated with state Incoming asserts that the agent is located beneath the site (with identity) loc. Therefore, a move to the "dock" sublocation of that same site is invisible with respect to the locations in Loc^M: the action implies $[\mathbf{false}]_{l.ag}$, for all $l \in Loc^M$.

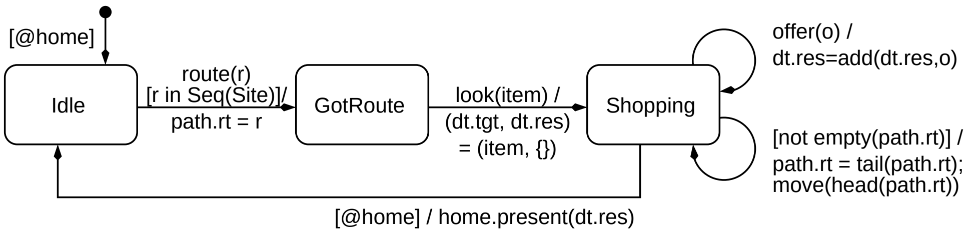
For these kinds of refinement to be admissible, it is essential that the spatial operators of MTLA refer to locations at an arbitrary depth instead of just the children of a node and that it is therefore impossible to specify the precise location of the agent. In fact, we consider the concept of "immediate sublocation" to be as dependent on the current level of abstraction as the notion of "immediate successor state", and MTLA allows to express neither.

4.3 Spatial Extension with Distribution of State

In the case of the docked shopper, the refinement could simply be represented by implication because the local attributes associated with the objects of the abstract model, such as the set offers are equally present at the refined level. In other cases, refinements of the spatial hierarchy will be accompanied by a distribution of the high-level attributes over the hierarchy of sublocations



(a) Class diagram.



(b) State machine.

Figure 10. Spatial refinement of the shopper.

of the refined model. For a simple example, departing again from the high-level shopper of Fig. 5, consider the UML model shown in Fig. 10. As shown in the class diagram, we assume that the shopping agent contains two sub-agents path that determines the path to follow through the network and dt that collects the data, and we have replaced the attributes lookFor and offers of the high-level shopper by attributes tgt and res assigned to the dt sub-agent.[2] The transition from Idle to GotRoute determines the route of the agent. It is guarded by the condition $r \in \mathit{Seq}(\mathit{Site})$, asserting that r is a list of (identities of) network sites.

Spatial distribution of attributes is similar to the concept of data refinement in standard refinement-based formalisms. Intuitively, the refinement of Fig. 10 is admissible provided that the public interface is preserved. We will therefore assume that the attributes item and offers have been marked as private in the class diagram for the abstract shopper, ensuring that no other object relies on their presence.

Formally, the visibility (either "private" or "public") of attributes is taken into account in the MTLA formalization of state machines by redefining the external specification of the behavior of an object o of class C with private attributes $a_1, \ldots, a_k$ to be the MTLA formula

$$C(o) \quad \equiv \quad \exists\, o.a_1, \ldots, o.a_k, o.\mathit{ctl}, o.\mathit{evts} : IC(o) \tag{6}$$

where $IC(o)$ is defined as before by formula (3). Since the specification of an object system is based on the external object specification, private attributes

[2] The renaming of the attributes is not necessary, but will make it clear in the following to which model we are referring.

are invisible at the system level, and the definition of refinement modulo a hypothesis remains as before.

The verification of refinement relies on conditions generalizing those of Thm. 1, provided that the private attributes of the high-level object can be computed from those of the implementation via a refinement mapping [27]. The relation between the two diagrams R and M is therefore given by the mapping η as before, complemented by terms $t_1, \dots, t_k$ that represent the values of the private high-level attributes $a_1, \dots, a_k$. These terms have then to be substituted for the attributes in the formulas concerning the high-level state machine M.

Theorem 2 *Extending the context of Theorem 1 by terms $t_1, \dots, t_k$, we now write $\overline{\varphi}$ for*

$$\varphi[Abs(o.ctl)/o.ctl, o.evts\upharpoonright_{\Sigma^M}/o.evts, msgs\upharpoonright_{\Sigma^M}/msgs, t_1/o.a_1, \dots, t_k/o.a_k]$$

If the set of public attributes of R is a superset of those of M then R refines M under hypothesis H up to hiding of attributes $o.a_1, \dots o.a_k$ if the conditions of Theorem 1 hold for this new interpretation of substitution.

The proof of Thm. 2 is a straightforward extension of that of Thm. 1. The assumption that every public attribute of M is also a public attribute of R is necessary to apply the elimination rule for existential quantification. As in Thm. 1, the proof obligations can be read off from the UML diagram. We note, however, that both theorems only give sufficient conditions for refinement, but that they are incomplete and will in general have to be complemented by definitions of auxiliary variables [27].

For the example shown in Fig. 10, the hypothesis is

$$H \equiv ag.path\langle\mathbf{true}\rangle \wedge ag.dt\langle\mathbf{true}\rangle$$

which, as before, reflects the additional structure given by the class diagram in Fig. 10(a). The states Idle and GotRoute of the state machine of Fig. 10 are both mapped to the abstract state Idle. The invariant mapping assigns the state formula $ag.path.rt \in Seq(Site)$ to the states GotRoute and Shopping. Finally, the refinement mapping is defined by substituting $ag.dt.res$ and $ag.dt.tgt$ for $ag.offers$ and $ag.lookFor$, respectively. All proof obligations of Thm. 2 are then easily verified.

4.4 Virtualisation of Locations

Whereas the notions of spatial refinement that we have considered so far have introduced new (sub-)objects, we have taken care to preserve the hierarchy of

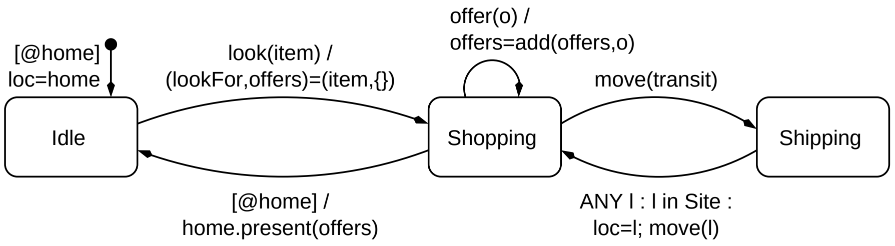


Figure 11. State machine for the "slow shopper".

the objects present at the abstract levels. Together with the choice of modalities of MTLA that can only specify how objects are nested with respect to one another, but cannot describe the precise location of an object, we have thus been able to represent refinement as implication and to preserve all MTLA properties. However, it can occasionally be desirable to allow for refinements that do not at all times preserve the spatial relationships imposed by the original specification. We therefore propose a final refinement principle for mobile systems where some location n of the high-level specification can be replaced by locations with different names and possibly different structure. Of course, the external behavior should be preserved—in particular, the name n should be hidden in the abstract model.

For example, Fig. 11 presents a variation of the original state machine of Fig. 5 where the agent moves to an intermediate transit location, which is not included in *Site*, before moving to the next site. (A subsequent refinement could add more structure to the transit location, modeling the transport of the agent across the network.) We cannot use Theorems 1 or 2 to prove that this model refines the original one because the move to the transit location during the transition from the Shopping to the Shipping state cannot be mapped to any high-level action. In fact, the MTLA formula representing the "slow shopper" does not imply the formula encoding the original specification, and the invariant formula (1) asserting that the shopping agent is always located at some location that represents a network site does not hold of the slow shopper.

Such relationships can be formalized by considering a weaker notion of refinement, abstracting from some of the names that occur in the original specification. In our running example, the name of the shopping agent should not actually be part of the interface: the purpose of the system is that the agent's home site learns about offers made by other network sites; the use of a mobile agent is an implementation detail. We say that an object system formalized by an MTLA formula *Impl* refines another system formalized by *Spec* up to hiding of name n if the implication $Impl \Rightarrow \boldsymbol{\exists}\, n : Spec$ holds. In general, the behavior required of object n at the abstract level may be implemented by several implementation objects, hence it does not appear useful to give a "local" rule, similar to Theorems 1 and 2, that attempts to prove refinement by considering a single state machine at a time. Instead, the strategy in proving such a refinement is to define a "spatial refinement mapping", using the rules given in Sect. 2. For the slow shopper, we first use rule (∃-sub) to introduce a new sublocation, say *l_virtual*, for every high-level location *l* and then define a refinement mapping that returns the implementation-level agent as long as it is not at the transit location, and otherwise the location *l_virtual* associated with the previous site visited as stored in the attribute loc. The local attributes of the high-level shopper are simply obtained from those of the implementation-level agent.

Refinement up to hiding of names allows for implementations that differ more radically in structure. For example, the single shopping agent of the initial specification could be implemented by a number of shopping agents that roam the network in parallel, cooperating to establish the shopping list. On the other hand, a correct implementation could also be based on a client-server solution instead of using mobile agents.

5 Conclusion and Further Work

We have defined the logic MTLA as an extension of TLA by spatial modalities. Our primary objective has been to identify adequate concepts of refinement for mobile systems, and to ensure that they are reflected by simple, yet precise relationships in the logic. In particular, this goal has motivated the definitions of the spatial modalities of MTLA as referring to deeply nested sub-locations and not just to immediate children as in most other spatial logics. Our exposition of MTLA in this article has been mostly semantical; a more comprehensive treatment of the logic and its meta-logical properties (decidability and axiomatization) appears in [18].

The formalism has been validated by an encoding of Mobile UML models. For this encoding, we have assumed some simplifications and restrictions of Mobile UML state machines. We have indicated sufficient conditions for verifying refinement of safety properties "object by object" where the proof obligations are directly drawn from the UML level and do not rely on the MTLA formalization. It would be interesting to combine our notion of refinement with refinement rules for adding and deleting transitions in UML state machines or changing transition labels [28]. We also intend to study adequate composition and decomposition concepts in future work. More broadly, we believe that semi-formal notations and concepts such as UML provide interesting opportunities to apply verification and refinement techniques. In particular, the proof obligations should be understandable at the semi-formal level and should not introduce unnecessary formal clutter, and adequate tool support will be essential.

References

[1] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, part I/II, Journal of Information and Computation 100 (1992) 1–77.

[2] L. Cardelli, A. Gordon, Mobile ambients, Theoretical Computer Science 240 (2000) 177–213.

[3] C. Fournet, G. Gonthier, The reflexive chemical abstract machine and the Join-calculus, in: Proceedings of the 23rd ACM Symposium on Principles of Programming Languages, ACM, St. Petersburg Beach, Florida, 1996, pp. 372–385.

[4] R. D. Nicola, G. Ferrari, R. Pugliese, Klaim: a kernel language for agents interaction and mobility, IEEE Trans. on Software Engineering 24 (5) (1998) 315–330.

[5] J. Vitek, G. Castagna, Seal: A framework for secure mobile computations, in: ICCL Workshop: Internet Programming Languages, 1998, pp. 47–77.

[6] L. Cardelli, A. Gordon, Anytime, anywhere. Modal logics for mobile ambients, in: Proceedings of the 27th ACM Symposium on Principles of Programming Languages, ACM Press, 2000, pp. 365–377.

[7] L. Caires, L. Cardelli, A spatial logic for concurrency (part I), Inf. and Comp. 186 (2) (2003) 194–235.

[8] R. D. Nicola, M. Loreti, A modal logic for Klaim, in: T. Rus (Ed.), Proc. Algebraic Methodology and Software Technology (AMAST 2000), Vol. 1816 of Lecture Notes in Computer Science, Springer-Verlag, Iowa, 2000, pp. 339–354.

[9] D. Sangiorgi, Extensionality and intensionality of the ambient logic, in: Proc. of the 28th Intl. Conf. on Principles of Programming Languages (POPL'01), ACM Press, 2001, pp. 4–17.

[10] L. Lamport, The Temporal Logic of Actions, ACM Trans. Prog. Lang. Syst. 16 (3) (1994) 872–923.

[11] H. Baumeister, N. Koch, P. Kosiuczenko, P. Stevens, M. Wirsing, UML for global computing, in: C. Priami (Ed.), Global Computing., Vol. 2874 of Lecture Notes in Computer Science, Springer-Verlag, Rovereto, Italy, 2003, pp. 1–24.

[12] H. Baumeister, N. Koch, P. Kosiuczenko, M. Wirsing, Extending activity diagrams to model mobile systems, in: M. Aksit, M. Mezini, R. Unland (Eds.), Objects, Components, Architectures, Services, and Applications for a Networked World (NODe 2002), Vol. 2591 of Lecture Notes in Computer Science, Springer-Verlag, Erfurt, Germany, 2003, pp. 278–293.

[13] D. Latella, M. Massink, H. Baumeister, M. Wirsing, Mobile UML statecharts with localities, Technical report 37, CNR ISTI, Pisa, Italy (2003).

[14] Object Management Group, UML 2.0 Superstructure Specification, Final Adopted Specification ptc/03-08-02, OMG, `http://www.uml.org/` (2003).

[15] S. Merz, A more complete TLA, in: J. Wing, J. Woodcock, J. Davies (Eds.), FM'99: World Congress on Formal Methods, Vol. 1709 of Lecture Notes in Computer Science, Springer-Verlag, Toulouse, France, 1999, pp. 1226–1244.

[16] G. Ostertag (Ed.), Definite Descriptions: A Reader, MIT Press, Cambridge, Mass., 1998.

[17] L. Lamport, How to write a long formula, Research Report 119, Digital Equipment Corporation, Systems Research Center (Dec. 1993).

[18] J. Zappe, Towards a mobile TLA, Ph.D. thesis, Institut für Informatik, Ludwig-Maximilians-Universität München, Munich, Germany, to appear (2005).

[19] M. Abadi, S. Merz, On TLA as a logic, in: M. Broy (Ed.), Deductive Program Design, NATO ASI series F, Springer-Verlag, 1996, pp. 235–272.

[20] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Sci. Comp. Program. 8 (3) (1987) 231–274.

[21] M. von der Beeck, A Structured Operational Semantics for UML-Statecharts, Softw. Syst. Model. 1 (2) (2002) 130–141.

[22] R. Back, J. von Wright, Refinement calculus—A systematic introduction, Springer-Verlag, 1998.

[23] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[24] M. Büchi, E. Sekerinski, A foundation for refining concurrent objects, Fundamenta Informaticae 44 (1, 2) (2000) 25–61.

[25] ClearSy System Engineering, Méthode B et Atelier B, `http://www.clearsy.com/html/b.htm`.

[26] KeesDa Corporation, Paradigm unifying system specification environments for proven electronic design, `http://www.keesda.com/pussee/`.

[27] M. Abadi, L. Lamport, The existence of refinement mappings, Theor. Comput. Sci. 82 (2) (1991) 253–284.

[28] S. Meng, Z. Naixiao, L. S. Barbosa, On Semantics and Refinement of UML Statecharts: A Coalgebraic View, in: J. R. Cuellar, Z. Liu (Eds.), Proc. 2nd IEEE Int. Conf. Software Engineering and Formal Methods, IEEE Computer Society, 2004, pp. 164–173.