On Refinement of Mobile UML State Machines

Alexander Knapp[1], Stephan Merz[2], and Martin Wirsing[1]

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

Abstract. We study the semantics and refinement of mobile objects, considering an extension of core UML state machines by primitives that designate the location of objects and their moves within a network. Our contribution is twofold: first, we formalize the semantics of state machines in MTLA, an extension of Lamport's Temporal Logic of Actions with spatial modalities. Second, we study refinement concepts for state machines that are semantically justified in MTLA.

1 Introduction

Software development for mobile computing and computations has to extend the traditional methods and techniques by adequate notations for describing mobile entities and design steps for refining models with mobile entities into implementations. For mobile systems, correctness of such implementations presents a major concern, as mobile agents may be roaming through a network and must be guaranteed to work reliably in different locations and in different environments.

In this paper, we attempt to combine semi-formal modelling for mobile systems with formal semantics and refinement techniques. For modelling, we consider an extension of state machines in the "Unified Modeling Language" (UML [13]) for mobility. We first formalize the semantics of mobile state machines in MTLA [10], an extension of Lamport's Temporal Logic of Actions with spatial modalities. Building on this logical semantics, we study refinement concepts for mobile state machines. In particular, we consider two notions of spatial refinements: On the one hand, mobile objects can be split into a hierarchy of cooperating mobile objects. On the other hand, non-hierarchical decompositions of locations can be justified to form correct refinements by abstracting from the details of the roaming strategy of a mobile object.

There has been much interest in formalizing concepts of UML as well as in semantic foundations for mobile computations, and we mention only the most closely related work. Deiß [6] has presented an encoding of (Harel) Statecharts in TLA, without considering either mobility or refinement. However, most formalizations of mobile computation are based on either process algebras as in [5,11] or on state machine models as in [8], sometimes accompanied by logics to describe system behavior [4,12], but without considering refinement. Our notion of refinement of state machines is partly inspired by [14,15]; a related notion has been elaborated in [16].

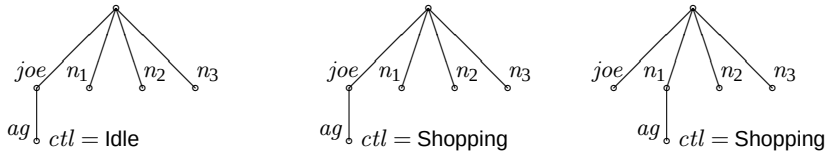


Fig. 1. Prefix of a run.

1.1 Mobile UML

Mobile UML [2,3] extends the Unified Modeling Language (UML [13]) by concepts for modelling mobile computation. The extension is described in terms of the UML itself, using stereotypes and tagged values as meta-modelling 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. An actual movement of a mobile object is performed by a move action that takes the target location as its parameter.

1.2 MTLA

The logic MTLA [10] is an extension of Lamport's Temporal Logic of Actions [9] intended for the specification of systems that rely on mobility of code. Due to space restrictions, we refer to [10] for precise definitions of its syntax and semantics[3] and only recall the basic intuitions and notations.

Similar as in the Ambient calculus [5] due to Cardelli and Gordon, we represent a configuration of a mobile system as a finite tree of nested locations. Mobility is reflected in modifications of the location hierarchy, as agents move in and out of nested domains. Whereas in the Ambient Calculus, each node of a configuration tree is associated with a process, MTLA associates a local state with every node. A run is modeled as an ω-sequence of configuration trees. For example, Fig. 1 shows three configurations of a system run. The first transition models a local action that changes the value of the local attribute ctl associated with location ag. The second transition represents a move of ag from the location joe to the location $n1$. Locations carry physical names: each name occurs at most once in any configuration tree.

MTLA contains both temporal and spatial modalities. Its formulas are evaluated over runs, at a given location. As is standard for (linear-time) temporal logics, temporal operators refer to the truth value of formulas at suffixes of a run. For example $\Box F$ asserts that F holds of all suffixes of the run, at the current location.

Similarly, spatial operators shift the spatial focus of evaluation, referring to locations below the current one. For example, the formula $m[F]$ asserts that F is true at location m (strictly) below the current location of evaluation, provided such a location exists, otherwise $m[F]$ is trivially satisfied. The dual formula $m\langle F\rangle$ asserts that the location m occurs beneath the current location, and that F holds there. We frequently use a

[3] In this paper, we do not use the "everywhere" operator $\overline{\Box}$ of [10].

more convenient dot notation to refer to local attributes at a given location and write, e.g., $ag.ctl$ to refer to the value of the attribute ctl of (the object denoted by) ag.

As in TLA, we use formulas to describe systems and their properties. State transitions are specified using action formulas that contain primed variables, as in $ag.ctl = \mathsf{Idle} \wedge ag.ctl' = \mathsf{Shopping}$. MTLA adds a transition formula $\alpha.n \gg \beta.n$ where n is a name and α and β are sequences of names. This formula asserts that the subtree rooted at name n within the subtree indicated by α is moved below the path β. The next-state relation of a system is specified by a formula $\Box[A]_v$ asserting that every transition that modifies the expression v must satisfy the action formula A. Similarly, $\Box[A]_{\alpha.n}$, where n is a name and α is a sequence of names stipulates that every transition that removes or introduces location n below the subtree indicated by α must satisfy A.

Hiding of state components can be expressed in MTLA using existential quantification. For example, $\exists\, ag.ctl : F$ holds if one can assign some value to the attribute $ag.ctl$ at every state such that F holds of the resulting run. (As in TLA, the precise definition is somewhat more complicated in order to preserve invariance under stuttering.) One may also quantify over names and write $\exists\, n : F$; this hides the name as well as all its attributes. Despite their somewhat complicated semantics, these quantifiers observe standard proof rules. In particular, we have the introduction axioms

$$\begin{array}{ll} (\exists\text{-ref}) & F\{t/n, t_1/n.a_1, \ldots, t_k/n.a_k\} \Rightarrow \exists\, n : F \\ (\exists\text{-sub}) & m\langle \mathbf{true} \rangle \Rightarrow \exists\, n : m.n\langle \mathbf{true} \rangle \qquad (m \not\equiv n) \end{array}$$

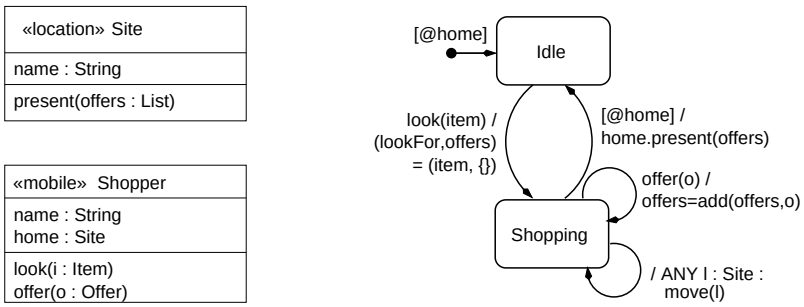
The axiom ($\exists$-ref) asserts that $\exists\, n : F$ can be derived by finding a "spatial refinement mapping" that substitutes witnesses for the hidden name n as well as for its attributes. The axiom ($\exists$-sub) allows us to introduce a new sublocation n of an existing location m.

2 Statecharts and their MTLA Semantics

We introduce mobile statecharts and provide a formal semantics based on MTLA. We illustrate mobile state machines by means of the "shopper" example: A mobile shopping agent is sent out by Joe to gather offers for some item in several shops; when returning to Joe, the shopping agent presents all offers that it has found.

2.1 State Machines for Mobility

UML state machines, an object-oriented variant of Statecharts as defined by Harel [7], are an expressive and feature-rich class of state transition systems with a complex semantics [17]. In this paper, we consider a restricted class of state machines, but extended by a special move action. In particular, we consider neither hierarchical nor pseudo-states, with the exception of a single initial state per state machine. We consider only events triggered by asynchronous signals (excluding call, time, and change events) and ignore deferred events. Although our encoding could be extended to encompass all 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. (b) State machine for the shopper.

Fig. 2. High-level model for the shopper.

Transitions of state machines are labelled by *triggers*, *guards*, and *actions*, any and all of which can be absent. Triggers denote signal receptions; they are of the form $\mathsf{op}(par)$ where op is the name of an operation declared in the class and par is a list of parameters. Guards are Boolean expressions over the attributes of the class and the parameters that appear in the trigger clause. Besides, 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. The most common form is $\mathsf{self} \prec e$, requiring the current object to be located below e, which we abbreviate to $@e$. Actions indicate the response of an object, beyond the state transition. For simplicity, we assume that all actions are of the form

$$\mathsf{ANY}\ x : P : upd; send; move$$

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 $e.\mathsf{op}(par)$ 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, and the location x bound by ANY. Figure 2(b) shows an initial state machine for our shopping agent, based on the class diagram of Fig. 2(a). For the subsequent refinements, we will not explicitly indicate the class diagrams, as they can be inferred from the elements that appear in the state machines.

Our interpretation of transitions deviates in certain ways from the UML standard. First, the UML 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 interpretation favors non-determinism and is therefore more appropriate for specifications at higher levels of abstraction. As a

second, minor deviation, we allow guards to appear in transitions from a state machine's initial state.

2.2 MTLA Semantics of State Machines

We formalize systems of interacting, mobile state machines in MTLA. The formalization enables us to prove properties about systems specified in UML. We will also use it to justify correctness-preserving refinement transformations.

In MTLA, every object is represented by a MTLA location whose local state includes a unique, unmodifiable identifier. 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 Locations), 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 whose nesting reflects the spatial relationships between objects.

The local state at each node represents the attributes declared for the corresponding object. Besides, we use the additional attributes $self$ to denote the unique object identifier, 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 (we assume that no attribute is of name $self$, ctl or $evts$). Objects interact asynchronously by sending and receiving messages. The communication network is represented 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. For lack of space, we do not give a precise, inductive definition of the translation, but only indicate its general form. 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 $x.self$ 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 $o.ctl = \mathsf{src} \wedge o.ctl' = \mathsf{dest}$.

A trigger $\mathsf{op}(par)$ 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

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

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 UNCHANGED $o.evts$ 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

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

The representation of an action consists of action formulae for multiple assignment, sending a message, and moving. If an action shows an ANY $x : P$ quantifier the conjunction $acts$ of these formulae are bound by a disjunction $\bigvee_{x \in Loc} P_o^x \wedge acts^x$. In more detail, a multiple assignment to attributes is represented by a formula

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

where $a_{k+1}, \ldots, a_n$ are the attributes of o that are not modified by the assignment and x is the variable bound by ANY. Sending a message $e.\mathsf{op}(par)$ is modelled by adding a tuple of the form $\langle e_o^x, \mathsf{op}, par_o^x \rangle$ to the network $msgs$. For actions that do not send a message we add the conjunct $msgs' = msgs$. A final conjunct constrains the moves caused by the current action. If the action contains a clause $\mathsf{move}(e)$, we add a conjunct

$$\bigvee_{l \in Loc} l.self = e_o^x \wedge \mathfrak{e}.o \gg l.o$$

that asserts that o will move to (the location with identity) e_o. Otherwise we add the conjunct $\bigwedge_{l \in Loc} [\textbf{false}]_{l.o}$ to indicate that the object does not enter or leave any location in Loc.

To model the reception of new events by the object, we add an action $RcvEvt(o, e)$ 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 $DiscEvt(o)$ that discards events that do not have associated 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 from 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

$$IC(o) \equiv \wedge\, Init(o) \wedge o.evts = \langle\rangle \wedge \Box[Next(o)]_{attr(o)} \wedge \Box[\textbf{false}]_{o.self} \wedge \bigwedge_{l \in Loc} \Box[Next(o)]_{l.o} \quad (1)$$

$$C(o) \equiv \exists\, o.ctl, o.evts : IC(o) \quad (2)$$

The "internal" specification $IC(o)$ asserts that the initial state must satisfy the initial condition, that all modifications of attributes of o and all moves of o (entering or leaving any location of Loc) are accounted for by the next-state relation, and that the object identity is immutable. Here, $attr(o)$ denotes the tuple consisting of the explicitly declared attributes and the implicit attributes ctl and $evts$. For example, the formula $IShopper(ag)$ shown in Fig. 3 defines the behavior of an object ag of class Shopper introduced in Fig. 2(b). The "external" specification $C(o)$ is obtained from $IC(o)$ by hiding the implicit attributes ctl and $evts$.

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

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

Fig. 3. MTLA specification of the shopper behavior (see Fig. 2(b))

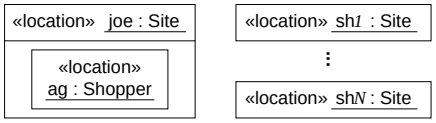


Fig. 4. Object diagram for the shopper example.

in Fig. 4. This configuration can be translated into the formula

$$
\begin{array}{ll}
\exists\, msgs : & \wedge \bigwedge_{i=1}^{N} sh_i\langle self = \mathsf{shop}\text{-}i \wedge joe[\mathbf{false}] \wedge \bigwedge_{j=1}^{N} sh_j[\mathbf{false}] \rangle \wedge \mathit{Site}(sh_i) \\
& \wedge\ joe\langle self = \mathsf{joe} \wedge \bigwedge_{i=1}^{N} sh_i[\mathbf{false}] \rangle \wedge \mathit{Site}(joe) \\
& \wedge\ joe.ag\langle self = \mathsf{shopper} \rangle \wedge \mathit{Shopper}(ag) \\
& \wedge \bigwedge_{l \in Loc} \Box[\mathbf{false}]_{l.sh_1, \ldots, l.sh_N, l.joe} \\
& \wedge\ msgs = \langle\rangle \wedge \Box\big[\bigvee_{o \in Obj} \mathit{Next}(o)\big]_{msgs}
\end{array}
\qquad (3)
$$

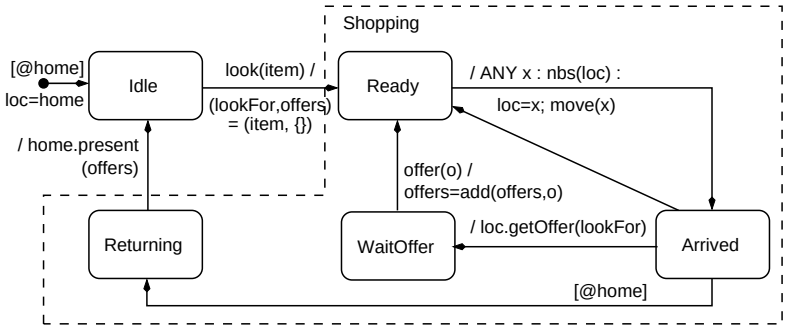


Fig. 5. Refined state machine for the shopper.

The formula in the scope of the existential quantifier asserts that the configuration contains the $N+1$ sites $sh_1, \ldots, sh_N$ and joe, and a shopping agent ag. Moreover, joe and the shops are immobile and unnested locations, whereas ag is situated beneath joe. 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, via existential quantification, the set of messages in transit, which is implicit at the UML level.

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

One purpose of our formalization is to prove properties about a system of objects. For the shopper example, we can deduce that the shopping agent is always located at its home agent or at one of the shops, expressed by the formula

$$\Box\Big(\bigvee_{l \in Loc} l.ag\langle \mathbf{true} \rangle\Big) \tag{4}$$

3 Refinement of state machines

In an approach based on refinement, interesting correctness properties of systems can already be established for models expressed at a high level of abstraction. These properties are preserved by subsequent models that provide a more detailed description of the system under development. In this paper, we focus on the refinement of state machines, and we add a "spatial" dimension to refinement in order to reveal more structure of the object hierarchy. In particular, a single high-level object can be refined into a tree of sub-objects. Throughout, we assume that the public interface of a refining class contains that of the refined one, and that the sets Loc and Obj corresponding to the refining model are supersets of those of the refined model.

3.1 Interface Preserving Refinement

Typically, refinement reduces the degree of non-determinism in a system description. For example, consider the state machine for the shopping agent shown in Fig. 5, which imposes a number of constraints with respect to the state machine shown in Fig. 2(b). After arriving at a new shop location (whose identity is recorded in the additional attribute loc), the agent may now either query for offers via a new message getOffer or immediately move on to another neighbor location. In the former case, it 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. 5 is a refinement of the one shown in Fig. 2(b) 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 are either explicitly allowed or 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 $nbs(l)$ contains only identities in $Site$, for all $l \in Loc$, each transition of the refined model either corresponds to a transition of the abstract model or to a stuttering transition, e.g. the transition from Arrived to WaitOffer.

We now formalize this intuition by defining what we mean by asserting that a state machine R refines another state machine M for a class C. Basically, refinement is represented in linear-time formalisms by trace inclusion or, logically, by validity of implication. However, we will be a little more precise about the context in which M and R are supposed to be embedded. Both machines are specified with respect to attribute and method signatures Σ^R and Σ^M that include all method names that appear in transition labels (either received or sent), and we assume that Σ^R extends Σ^M. Similarly, the sets Obj^R and Loc^R of MTLA names for the objects and the location objects at the level of the refinement should be 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 stated by an MTLA state predicate H. Thus, we say that R refines M under hypothesis H if for all system specifications Sys^M and Sys^R such that Sys^R results from Sys^M by replacing all occurrences of the MTLA specifications $C^M(o)$ by $C^R(o)$ and by conjoining some formulas such that Sys^R implies $\Box H$, the implication $Sys^R \Rightarrow Sys^M$ 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 $\eta(s) = (Inv(s), Abs(s))$ where $Inv(s)$ is a set of MTLA state predicates, possibly containing spatial operators, and where $Abs(s)$ 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. A proof of the following theorem appears in the appendix.

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 $Inv(s)$ of* MTLA

state predicates and a state $Abs(s)$ of M. If all of the following conditions hold then R refines M under hypothesis H. We write $\overline{\varphi}$ for

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

where $e\restriction_\Sigma$ denotes the subsequence of elements 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 formalized by formula $A(o, par)$ and corresponding state $Abs(s)$ of M where $B_1(o, par_1), \ldots, B_m(o, par_m)$ are the* MTLA *formulas formalizing the outgoing transitions of $Abs(s)$, $attr^M(o)$ is the tuple of attributes defined for M and Loc^M is the set of locations for M:*

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

Theorem 1 ensures that R can replace M in any system of mobile objects satisfying the syntactical requirements, subject to hypotheses H. In particular, all properties established for the high-level system will be preserved by the implementation.

In order to prove that the state machine of Fig. 5 refines that of Fig. 2(b) (with respect to $H \equiv \forall s \in Site : nbs(s) \in Site$) 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 theorem 1. In particular, the transitions leaving state Arrived do not modify the shopping agent's attributes or the projections of the messages to the high-level signature, and are therefore allowed by condition (3) of Theorem 1.

Theorem 1 can also be used to justify refinements that modify the spatial hierarchy of locations. Consider the state machine shown in Fig. 6. 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. 2(b) and 5, this version moves the shopping agent first to the "incoming" sublocation of the target location. If the agent is accepted by the host, as modelled 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. Here we assume that every location $l \in Loc$ contains sublocations l_in and l_dock. Moreover, we assume functions incoming and dock to look up the id's of the corresponding sub-locations for a given network site.

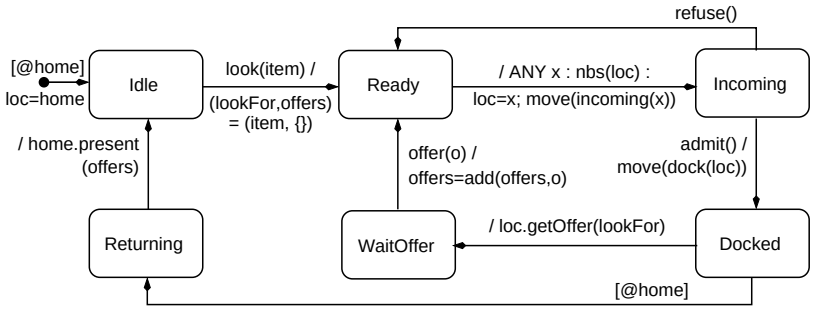


Fig. 6. Spatial refinement of the network sites.

Formally, Theorem 1 can again be used to show that the "docked" shopper of Fig. 6 is a refinement of that shown in Fig. 5 with respect to the hypothesis

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

The states Incoming and Docked 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 the Ready to the Incoming state will be formalized by an MTLA action formula equivalent to $\bigvee_{l \in Loc^R} l.ag \gg m\_in.ag$, which implies the corresponding formula $\bigvee_{l \in Loc^M} l.ag \gg m.ag$ formalizing the move between the high-level states Ready and Arrived, using the hypothesis H. Similarly, H and the invariant establish that the move between the Incoming and Docked states maps to a stuttering action: clearly, 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. This observation led us to discard the attribute indicating the location of a mobile object proposed in [3].

3.2 Interface Refinement I: Spatial Distribution of State

Frequently, refinements of the spatial hierarchy will be accompanied by a distribution of the high-level attributes over the hierarchy of sublocations of the refined model. For a simple example, departing again from the high-level shopper of Fig. 2(b), consider the state machine shown in Fig. 7. Here 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-

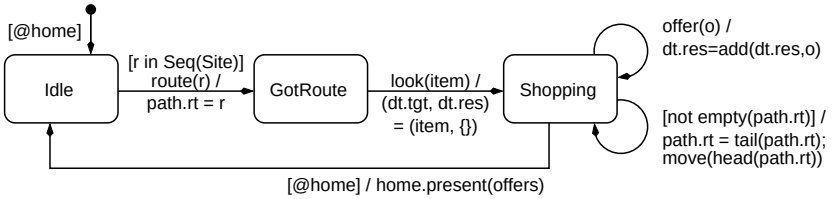


Fig. 7. Spatial refinement of the shopper.

level shopper by attributes tgt and res assigned to the dt sub-agent.[4] The transition from Idle to GotRoute determines the route of the agent. It is guarded by the condition $r \in Seq(Site)$, 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. 7 is admissible provided that the public interface is preserved. We will therefore assume that the attributes item and offers have been marked as private attributes in the class diagram for the abstract shopper, ensuring that no other object relies on their presence.

Formally, we modify slightly the MTLA formalization of state machines, taking into account the visibility (either "private" or "public") of attributes. We redefine the external specification of the behavior of an object o of class C with private attributes $a_1, \ldots, a_k$ as the MTLA formula

$$C(o) \;\equiv\; \exists\, o.a_1, \ldots, o.a_k, o.ctl, o.evts : IC(o) \tag{5}$$

where $IC(o)$ is defined as before by formula (1). Since the specification of an object system in formula (3) is based on the external object specification, private attributes 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 Theorem 1, provided that the private attributes of the high-level object can be computed from those of the implementation via a refinement mapping [1]. The relation between the two diagrams R and M is therefore given by the mapping η as before, complemented by terms $t_1, \ldots, t_k$ that represent the values of the private high-level attributes $a_1, \ldots, a_k$. 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 $t_1, \ldots, t_k$, we now write $\overline{\varphi}$ for*

$$\varphi\{Abs(o.ctl)/o.ctl, o.evts{\restriction}_{\Sigma^M}/o.evts, msgs{\restriction}_{\Sigma^M}/msgs, t_1/o.a_1, \ldots, t_k/o.a_k\}$$

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 $o.a_1, \ldots o.a_k$ if the conditions of Theorem 1 hold for this new interpretation of substitution.

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

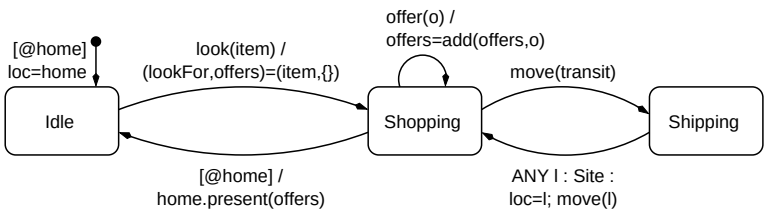


Fig. 8. State machine for the "slow shopper".

Proof. The proof is analogous to that of Theorem 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. □

For the example shown in Fig. 7, the hypothesis is

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

The implementation states Idle and GotRoute 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 Theorem 2 are then easily verified.

3.3 Interface Refinement II: Virtualisation of Locations

So far, when refining locations, we have always taken care to preserve the hierarchy of locations that existed at the abstract level. In MTLA, and also in the version of mobile UML that we consider in this paper, the location of an agent cannot be specified precisely, but only relatively to the locations of the other objects of the model at the current level of abstraction. The conditions of Theorems 1 and 2 are therefore sufficient to ensure that all MTLA formulas are preserved, even if they contain spatial operators. 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.

For example, the previous specifications of the shopping agent have all assumed that moves between locations happen atomically. Figure 8 presents a variation of the original state machine of Fig. 2(b) 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 cannot be mapped to any high-level action. Even more, the MTLA formula representing the "slow shopper" does not imply the formula encoding the original specification. In particular, the invariant formula (4) asserting that the shopping agent is always located at some location that represents a network site does not hold of the slow shopper.

Such a relationship can be formalized by considering a weaker notion of refinement in which we abstract from some of the names that occur in the original specification. In

our running example, one may argue that the name of the shopping agent should not 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 $Impl \Rightarrow \exists\, n : Spec$ 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 attempting 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 Section 1.2. For the slow shopper, we first use rule ($\exists$-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 in any case 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, an implementation could also be based on a traditional client-server solution instead of using mobile agents.

4 Conclusion

We have studied the applicability of the logic MTLA proposed in [10] in view of formalizing Mobile UML State Machines [3] and establishing refinement relationships between models described in this language. A configuration of a mobile system is represented as a tree of names, and mobility is reflected by changes to the name hierarchy. MTLA provides for local state at every node in the tree, which simplifies the formalization of state-based notations such as UML state machines that may manipulate object attributes. The operators of MTLA have been designed to support system refinement; in particular, all spatial operators refer to nodes arbitrarily deep beneath the current node and not just its children as in other spatial logics, e.g. [4].

We have assumed some simplifications and restrictions for our formalization of Mobile UML state machines. In particular, we assume that spatial relationships are specified using constraints $e_1 \prec e_2$, comparing the relative positions of two objects at the current level of abstraction. This assumption has been essential to obtain a sound and elegant representation of refinement as implication of specifications for mobile systems.

Our main objective has been the study of three fundamental refinement principles, focusing on refinements of the spatial hierarchy. We have indicated sufficient conditions for verifying refinement. However, these conditions are incomplete: in particular, it is well known that refinement mappings need to be complemented by devices such as history and prophecy variables for completeness results [1]. We have also ignored liveness and fairness properties in this paper, and we have mostly restricted ourselves to proving refinement "object by object". We intend to study adequate composition and decomposition concepts in future work.

References

1. Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 81(2):253–284, May 1991.
2. Hubert Baumeister, Nora Koch, Piotr Kosiuczenko, Perdita Stevens, and Martin Wirsing. UML for global computing. In C. Priami, editor, *Global Computing. Programming Environments, Languages, Security, and Analysis of Systems*, volume 2874 of *Lecture Notes in Computer Science*, pages 1–24, Rovereto, Italy, 2003. Springer-Verlag.
3. Hubert Baumeister, Nora Koch, Piotr Kosiuczenko, and Martin Wirsing. Extending activity diagrams to model mobile systems. In M. Aksit, M. Mezini, and R. Unland, editors, *Objects, Components, Architectures, Services, and Applications for a Networked World (NODe 2002)*, volume 2591 of *Lecture Notes in Computer Science*, pages 278–293, Erfurt, Germany, 2003. Springer-Verlag.
4. 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.
5. Luca Cardelli and Andrew Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.
6. Thomas Deiß. An approach to the combination of formal description techniques: Statecharts and TLA. In K. Araki, A. Galloway, and K. Taguchi, editors, *Integrated Formal Methods (IFM 1999)*, pages 231–250, York, UK, 1999. Springer-Verlag.
7. David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.
8. Thomas A. Kuhn and David von Oheimb. Interacting state machines for mobility. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *Proc. 12th Intl. FME Symposium (FM2003)*, volume 2805 of *Lecture Notes in Computer Science*, pages 698–718, Pisa, Italy, September 2003. Springer-Verlag.
9. Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
10. Stephan Merz, Júlia Zappe, and Martin Wirsing. A spatio-temporal logic for the specification and refinement of mobile systems. In M. Pezzè, editor, *Fundamental Approaches to Software Engineering (FASE 2003)*, volume 2621 of *Lecture Notes in Computer Science*, pages 87–101, Warsaw, Poland, April 2003. Springer-Verlag.
11. 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.
12. 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.
13. Object Management Group. Unified Modeling Language Specification, Version 1.5. Specification, OMG, 2003. `http://cgi.omg.org/cgi-bin/doc?formal/03-03-01`.
14. Barbara Paech and Bernhard Rumpe. A new concept of refinement used for behaviour modelling with automata. In *FME'94*, volume 873 of *Lecture Notes in Computer Science*, pages 154–174, Barcelona, Spain, 1994. Springer-Verlag.
15. Peter Scholz. A refinement calculus for Statecharts. In *Intl. Conf. Fundamental Approaches to Software Engineering (FASE'98)*, volume 1382 of *Lecture Notes in Computer Science*, pages 285–301, Lisbon, Portugal, 1998. Springer-Verlag.
16. Michael Schrefl and Markus Stumptner. Behavior-consistent specialization of object life cycles. *ACM Trans. Software Engineering and Methodology*, 11(1):92–148, 2002.
17. Michael von der Beeck. Formalization of UML-statecharts. In M. Gogolla and C. Kobryn, editors, *Proc. 4th Int. Conf. UML (UML 2001)*, volume 2185 of *Lecture Notes in Computer Science*, pages 406–421. Springer, 2001.

A Proof of theorem 1

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

$$\models ISys^R \Rightarrow \overline{\overline{ISys^M}} \tag{6}$$

where $\overline{\overline{F}}$ denotes the formula

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

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

From (6), we obtain the assertion $\models Sys^R \Rightarrow Sys^M$ by the standard introduction and elimination rules for existential quantification.

For the proof of (6), 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 $C^R(o)$. In particular, 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 (6) is proved 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,

$$\models ISys^R \Rightarrow Init^M(o_i) \wedge o_i.evts = \langle\rangle$$

and therefore it follows that

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

Similarly, $Next^R(o_i)$ and $Next^M(o_i)$ as well as $attr^R(o_i)$ and $attr^M(o_i)$ 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

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

observe that for every action $A(o_i, par)$ other than $RcvEvt(o_i, e)$ or $DiscEvt(o_i)$, its definition is such that

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

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

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

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

This completes the proof of (6), and thus of theorem 1. $\Box$