Foundations of System Development

Martin Wirsing

in cooperation with
Axel Rauschmayer

WS 05/06

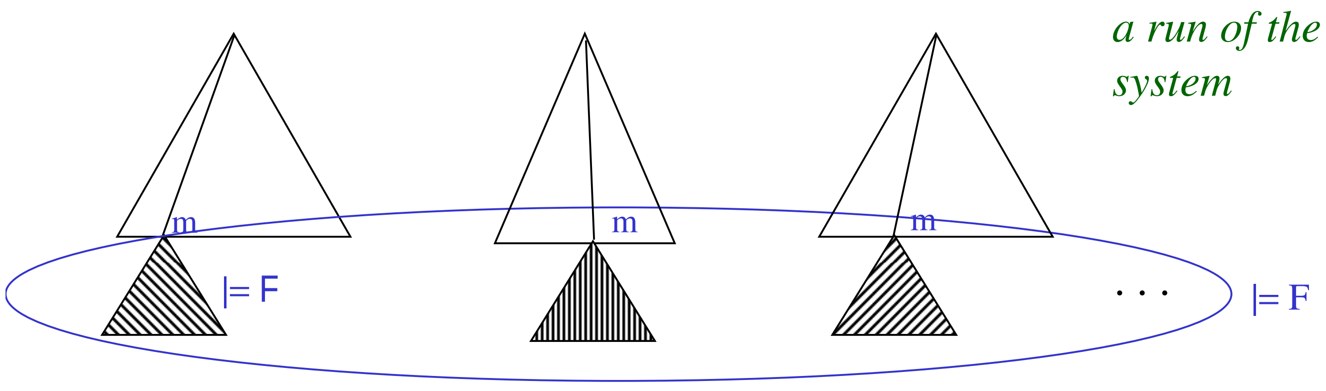
Ausblick:
Systematische Entwicklung Mobiler Systeme

Goals

- **Modelling and Developing Systems Using UML and MTLA**
  - **MTLA – Mobile Temporal Logic of Actions**
  - **State diagrams with mobility**
  - **Correct state diagram refinement**

1. MTLA

- MTLA extends TLA by
  - **location information**
    - m[F] formula F holds at location m if m exists



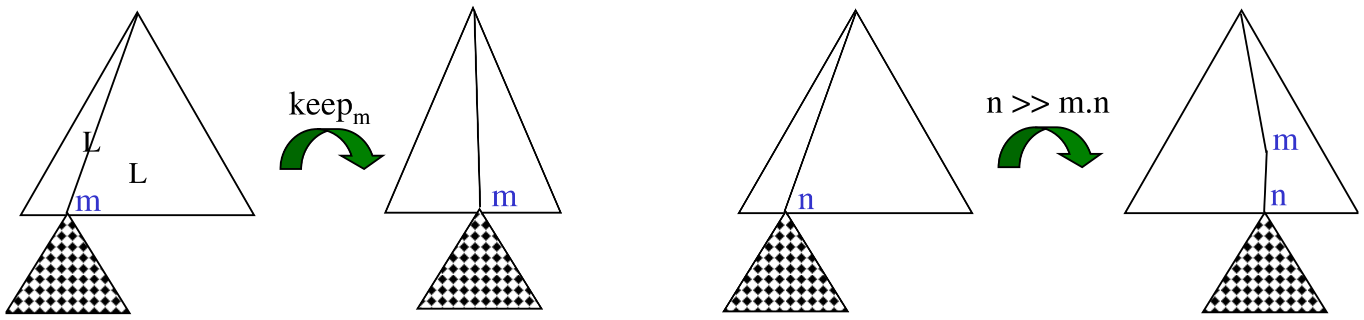
1. MTLA

move actions

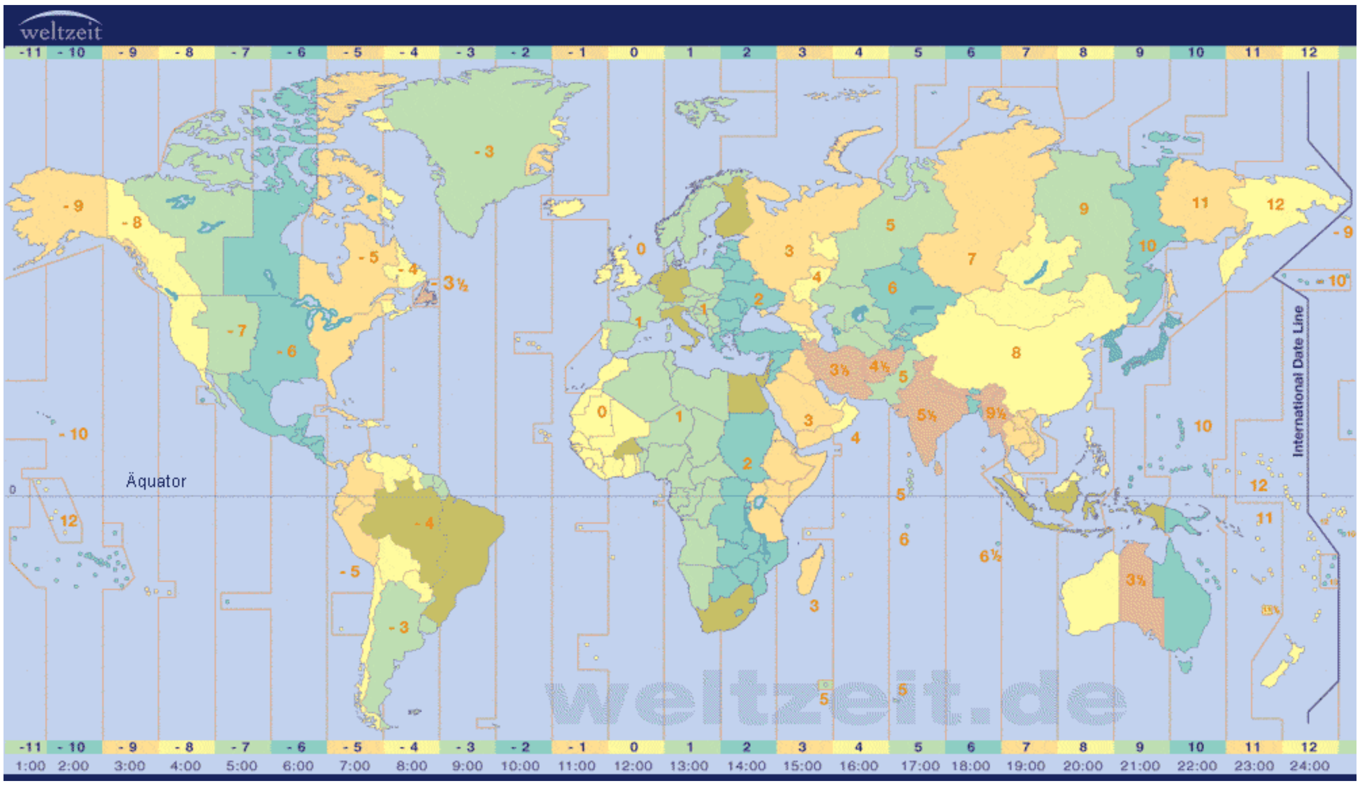
- keep$_m$ the topology below m does not change
- n >> m.n the tree below n moves below m

More generally,

- α.n >> β.n the subtree of path α below n moves to the tree below β



1. MTLA Example: Mobile Clock - Time Zones



1. MTLA Example: Mobile Clock

Syntactically sugared spec

$[\text{false}]_{\text{value}}$

tz_i non mobile: $[\text{false}]_{tz_i}$

MODULE WorldClock = [

EXTENDS Naturals

VARIABLES hr, value

NAMES clock, tz_{-11} , … , tz_0 , … , tz_{12}

Network $\equiv$ Rigid(value) $\wedge$ NonMobile(tz_{-11}) $\wedge$ **…** $\wedge$ NonMobile(tz_{12})

$\bigwedge_{i,j \in (-11..12),\, i<>j} tz_i[tz_j[\text{false}]]$

$\wedge$ (tz_{-11}.clock **v** **…** **v** tz_{12}.clock)

tz_i<clock<true>>

WCini $\equiv$ hr $\in$ (0..23)

$\wedge$ tz_{-11}.value = -11 $\wedge$ **…** $\wedge$ tz_{12}.value = 12

tz_i<value = i>

WCnxt $\equiv$ hr' = hr+1 mod 24

WChangeTZ$_{i,j}$ $\equiv$ tz_i.clock >> tz_j.clock

$\wedge$ clock[hr' = (hr + tz_j.value –tz_i.value) mod 24]

WCSave $\equiv$ Network $\wedge$ WCini $\wedge$

$\wedge$ [$\bigvee_{i,j \in (-11..12),\, i<>j}$ WChangeTZ$_{i,j}$ **v** WCnxt]$_{hr}$

$\bigwedge_{i \in (-11..12)}$ [$\bigvee_{j \in (-11..12),\, i<>j}$ WChangeTZ$_{i,j}$]$_{-tzi.clock}$

1. MTLA Example: Mobile Clock

- Fairness conditions for Mobile Clock would be
  - Weak fairness of WCnxt
- No fairness requirement for WChangeTZ (the clock is allowed to remain in a time zone)
- The following specification

  WC $\equiv$ WCsave $\wedge$

  WF$_{hr}$(WCnxt)

  ensures that the clock will always advance.

1. MTLA System Specifications

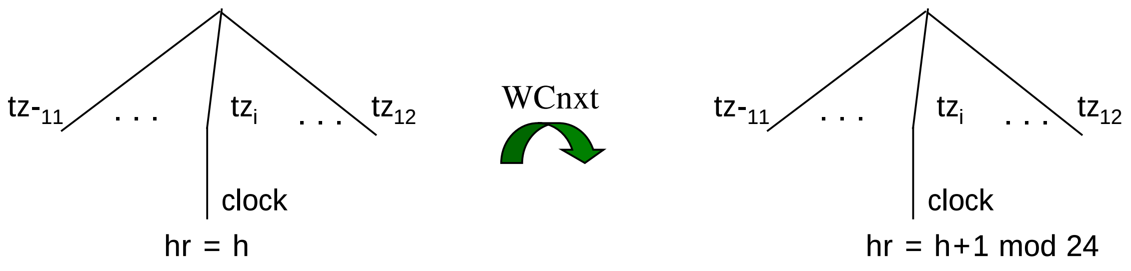
- MTLA system specifications add action formulas for change of locations:
  - Most MTLA system specifications are of the form

    $\text{Init} \wedge \quad [\text{Next}]_v \wedge \quad [\text{Next}]_S \wedge \quad L$

    where

    $[\text{Next}]_S$ specifies that Next is unchanged or the location information S changes.

2. Transition Systems for Mobility



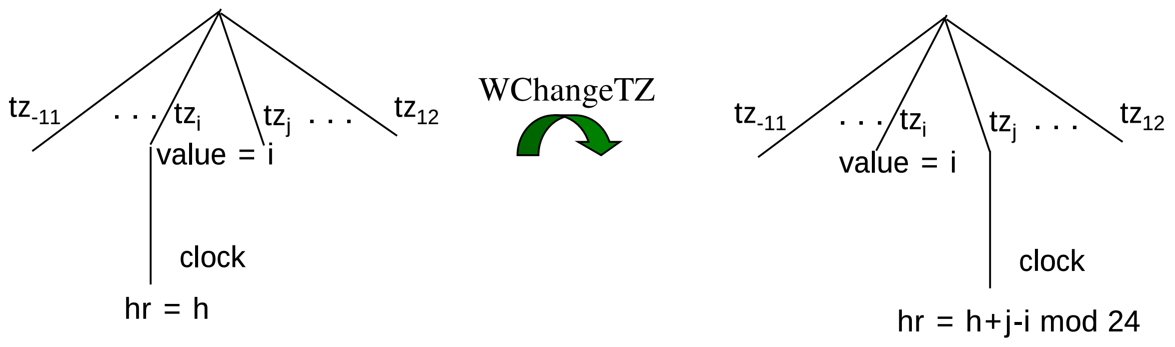
Configurations (t, λ)

t	finite tree, edges labelled by unique names
λ	assigns local states to nodes

Computations $\sigma = (t_0, \lambda_0), (t_1, \lambda_1), \ldots$

2. Transition Systems for Mobility: Configurations

- A configuration is defined with respect to a non-empty universe |I| and a set V_f of (flexible) variables:
  - A **configuration** is a pair (t, λ) where
    - $t = (N_t, <_t)$ is a finite, non-empty tree and
    - $\lambda : N_t \times V_f \to |I|$ assigns a value to every variable in V_f at every location $n \in N_t$.
- Example



$\lambda_0(\text{clock}, \text{hr}) = h$, $\lambda_1(\text{clock}, \text{hr}) = (h+j-i) \bmod 24$

Rigid values as in TLA: $\xi(tz_1, \text{value}) = 1, \ \ldots, \xi(tz_5, \text{value}) = 5$,

2. Transition Systems for Mobility: Trees

- A finite, non-empty tree t is given by a
  - strict partial order $(N_t, <_t)$
  - over a finite set $N_t \subset N$ of names
  - with distinctive root ε

- The subtree of a tree $t = (N_t, <_t)$ rooted at node n is defined by

$$t{\downarrow}n = \begin{cases} (\{m \in \mathsf{N} \mid m <_t n\}, <'_t) & \text{if } n \in \mathsf{N}_t \\ \text{empty} & \text{otherwise} \end{cases}$$

- where $<'_t$ is the restriction of $<_t$ to the subtree of n, i.e.

$$= \quad <_t \cap (\{m \in \mathsf{N} \mid m <_t n\} \times \{m \in \mathsf{N} \mid m <_t n\})$$

3. pMLTL: Linear Temporal Logic

- LTL is a logic for specifying properties of runs
- LTL formulas are built by using
  - first order logic operators (negation, implication, quantifiers),
  - modal operators for specifying temporal properties
    - F "F holds always; i.e. in all states of the run"
    - ○F "F holds in the next state"

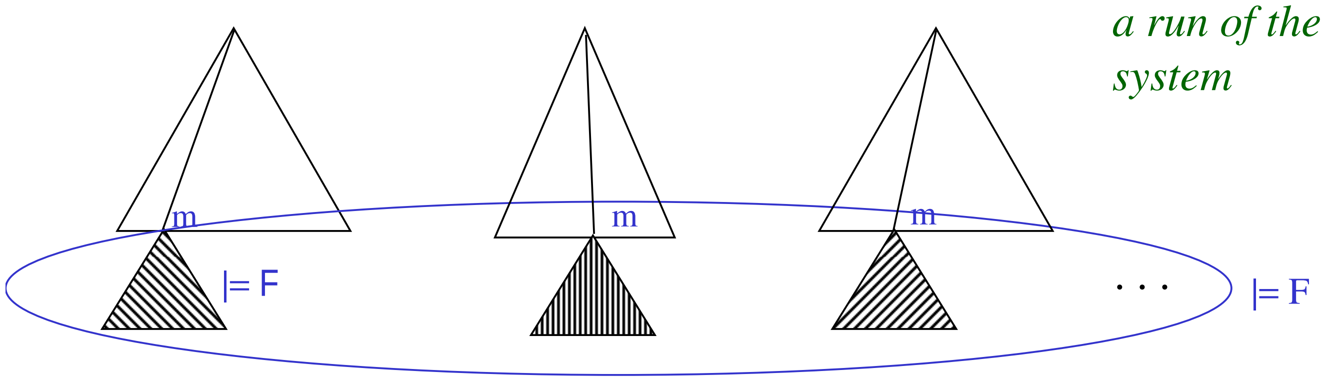
3. pMLTL: Syntax

- Propositional Linear Temporal Logic for Mobility [Zappe 05]
- pMLTL is the **propositional fragment of LTL** extended by mobility operators.
- Syntax
  - Let V be a countable set of propositional variables and N a countable set of names (for representing locations).
  - **Formulas** are inductively defined by

F ::=	V	propositional (boolean) variables
	$\mid$ F => F $\mid \neg$ F	classical propositional logic
	$\mid$ F $\mid$ ○F	"always" , "next"
	$\mid$ m[F] $\mid$ keep_m	mobility operators

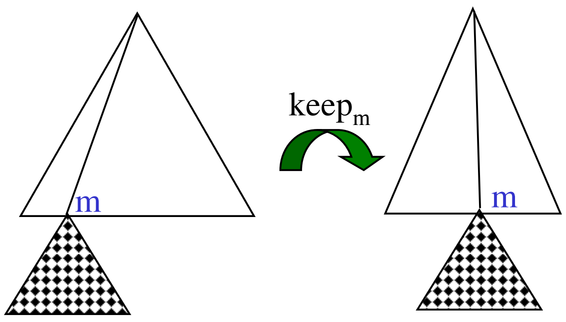
3. pLMTL: Semantics informally

- **location information**
  - m[F] formula F holds at location m **if** m exists
  - m<F> formula F holds at location m **and** m exists



3. pMLTL: Semantics informally

- keep_m the topology below m does not change



- “Move” can be expressed:
  - $n \gg m.n \;=_{def} n\langle true\rangle \wedge \circ\; m\langle n\langle true\rangle\rangle \wedge \text{keep}_n$

3. pMLTL: Semantics

- Let $\sigma = (t_0,\lambda_0), (t_1,\lambda_1), \ldots$ be a run, and n be a name.
- We define

  $\sigma, n \models F$ "F holds for σ at node n"

  inductively:

  - $\sigma, n \models v \quad \mathit{iff} \quad n \in \mathsf{N}_0^{\varepsilon} \mathit{\ and\ } v \in \lambda_0(n)$
  - $\sigma, n \models \neg F \quad \mathit{iff} \quad \sigma, n \not\models F$
  - $\sigma, n \models F \Rightarrow G \quad \mathit{iff} \quad \sigma, n \not\models F \mathit{\ or\ } \sigma, n \models G$

3. pMLTL: Semantics (continued)

- $\sigma, n \models m[F] \quad \mathit{iff} \quad m \not<_0 n \mathit{\ or\ } \sigma, m \models F$
- $\sigma, n \models \mathsf{keep}_m \quad \mathit{iff} \quad t_0{\downarrow}n.m = t_1{\downarrow}n.m$
- $\sigma, n \models \circ F \quad \mathit{iff} \quad n \notin \mathsf{N}_1 \mathit{\ or\ } \sigma|_1, n \models F$
- $\sigma, n \models \Box F \quad \mathit{iff} \quad \mathit{for\ all\ } i \geq 0 \mathit{\ either\ } n \notin \mathsf{N}_j \mathit{\ for\ some\ } j \leq i \mathit{\ or\ } \sigma|_i, n \models F$

- Validity
  - $\sigma \models F$ iff σ, n $\models$ F for all names n
  - $\models F$ iff σ, n $\models$ F for all names n and all runs σ

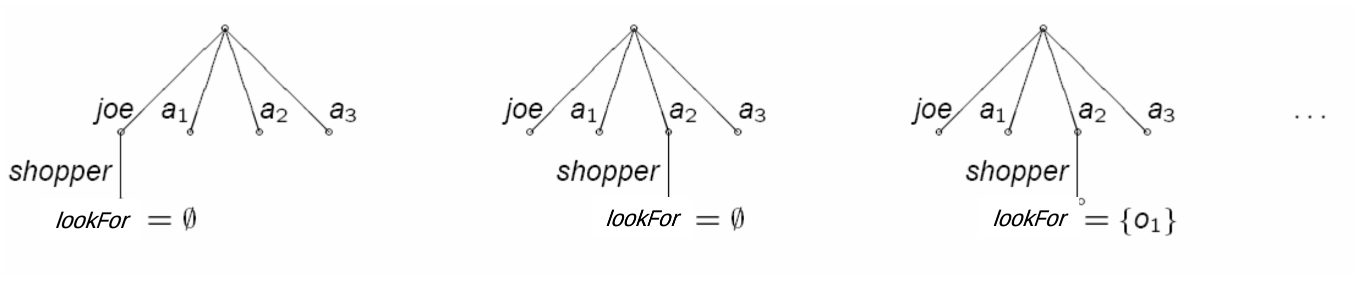
3. pMLTL: Derived Operators

- m<F> "F holds at m **and m exists"**
  - $m\langle F\rangle =_{def} \neg m[\neg F]$
- "Move" can be expressed:
  - $n \gg m.n =_{def} n\langle true\rangle \wedge \circ\, m\langle n\langle true\rangle\rangle \wedge keep_n$
- <> F "F holds eventually"
  - $\Diamond F =_{def} \neg \quad (\neg F)$

4. MTLA: Notations (for action formulas)

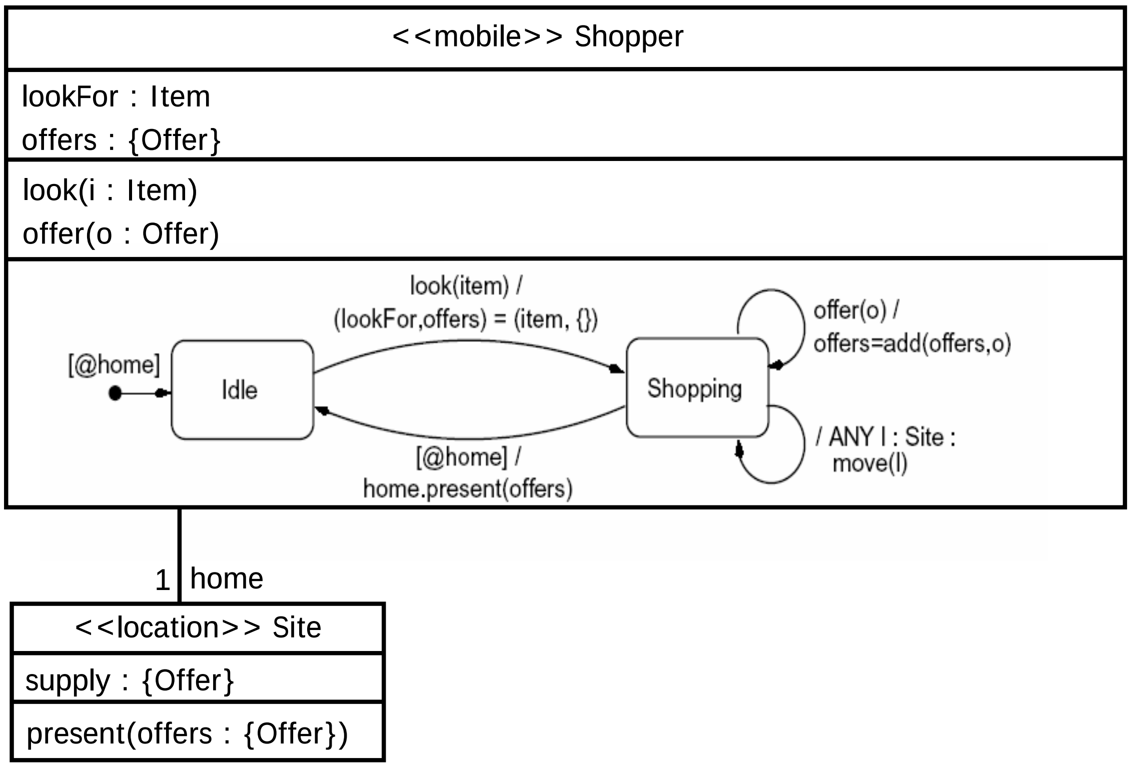
- For any action A, state function t, and any pure spatial formula S (i.e. not containing temporal operators), define
  - $[A]_t \equiv A \vee t = t'$ $[A]_S \equiv A \vee (S \Leftrightarrow \circ S)$
  - $\langle A\rangle_t \equiv A \wedge \neg(t = t')$ $\langle A\rangle_S \equiv A \wedge \neg(S \Leftrightarrow \circ S)$
  - $[A]_{-S} \equiv [S \Rightarrow A]_S$

4. MTLA Example: Mobile Shopper



- A mobile shopper gets the request of finding offers for an item, e.g. for different flights.
- He visits several shops, collects the offers and returns home (after some time).

4. MTLA Example Shopper: UML solution



4. MTLA Example Shopper: Direct specification in MTLA

Assume: fixed, finite set Net of names, joe ∈ Site, shopper not in Site

Network topology

Topology ≡ $\bigwedge_{n,m \in Site}$ n<m[**false**]>	all nodes present at top level

Initial condition

Init ≡ joe<shopper<**true**>>	shopping agent in domain joe. . .
∧ shopper[ctl = “Idle”]	. . . and in “Idle” state

shopper.ctl = “idle”
abbreviates: **shopper<true>** ∧ **shopper[ctl = “idle”]**

Prepare shopper to shop for item x

Prepare(x) ≡

shopper<**true**> ∧ **o** shopper<**true**>	shopping is (and stays) here
∧ shopper[ctl = “Idle”]	state changes from “idle” . . .
∧ **o** shopper[ctl = “Shopping”]	. . . to “shopping”
∧ **o** shopper[lookFor = x ∧ offers = {}]	initialize lookFor and offers

4. MTLA Example Shopper (continued)

Remaining state-changing actions

GetOffer ≡. . .	get an offer and insert into “offers”
PickOffer ≡. . .	select among offers in “offers”

Move among network nodes

$\mathrm{Move}_{n,m}$ ≡

n<shopper<**true**>>	shopping agent is in n’s domain
∧ shopper[ctl = “Shopping”]	and is in “Shopping” state
∧ n.shopper >> m.shopper	shopper moves to m’s domain,
∧ UNCHANGED(shopper.offers, shopper.lookFor, shopper.ctl)	preserving local state

Overall specification (ignoring fairness)

Shopper ≡

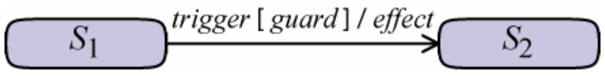
Topology ∧ Init

∧ $[\mathrm{joe}[(\exists\, x : \mathrm{Prepare}(x)) \vee \mathrm{PickOffer}] \vee \bigvee_{n \in Site} n[\mathrm{GetOffer}]]_{vars}$

∧ $\bigwedge_{n \in Site} [\bigvee_{m \in Site} \mathrm{Move}_{n,m}]_{\text{-}n.shopper}$

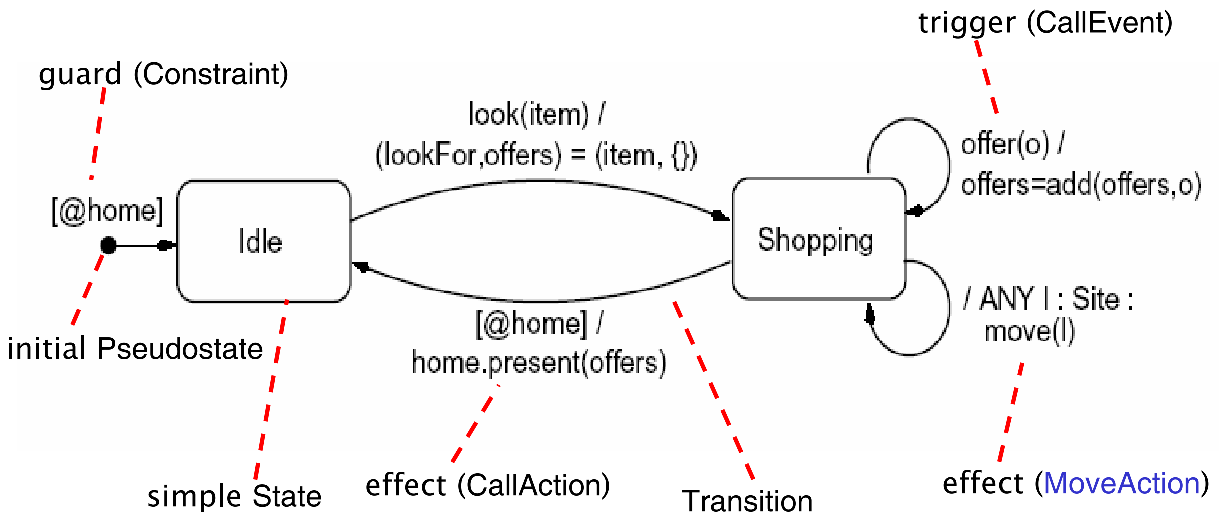
5. Mobile State Machines

- **State machines** model the behavior of (single) objects.
- **History and predecessors**
  - 1950's: Finite State Machines: Huffmann, Mealy, Moore
  - 1987: Harel Statecharts: conditions and hierarchical (and/or) states
  - 1994: ROOM Charts: run-to-completion (RTC) step
- State machines model **behavior**
  - using states interconnected ...
  - with transitions triggered ...
  - by event occurrences.

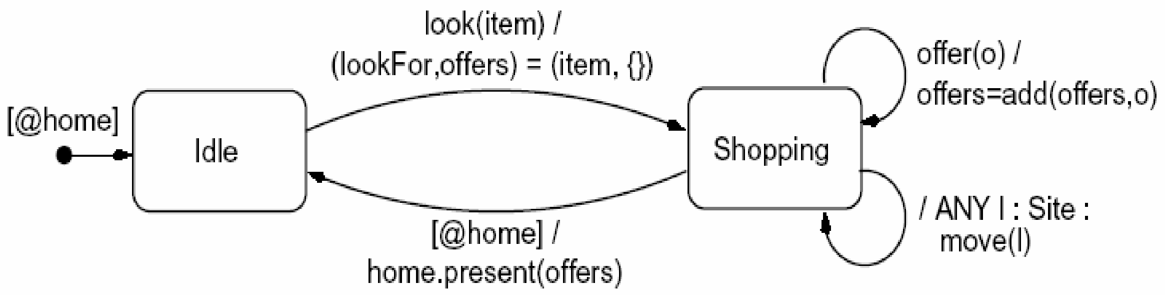


- Goal of the **extension to mobility**
  - include location information and move operations into the state machine behaviour

5. Mobile state machines: Example Shopper



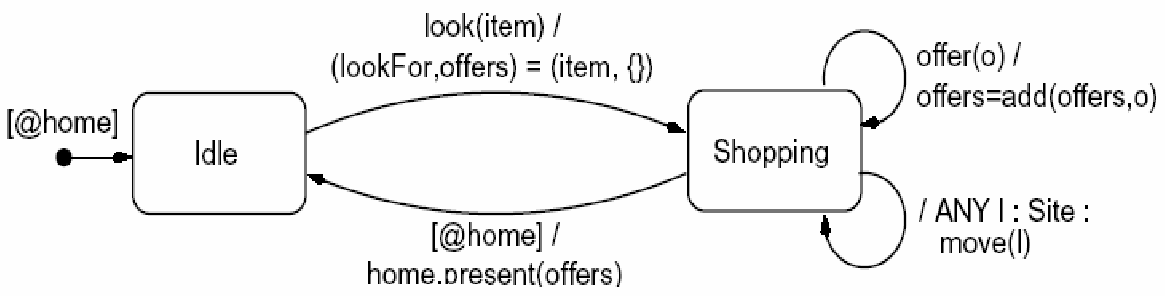
5. Mobile state machines: Example Shopper



Extensions

- guards $e_1 \prec e_2$
- $@e \triangleq \mathsf{self} \prec e$
- special move action

5. Mobile state machines: Example Shopper



Extensions

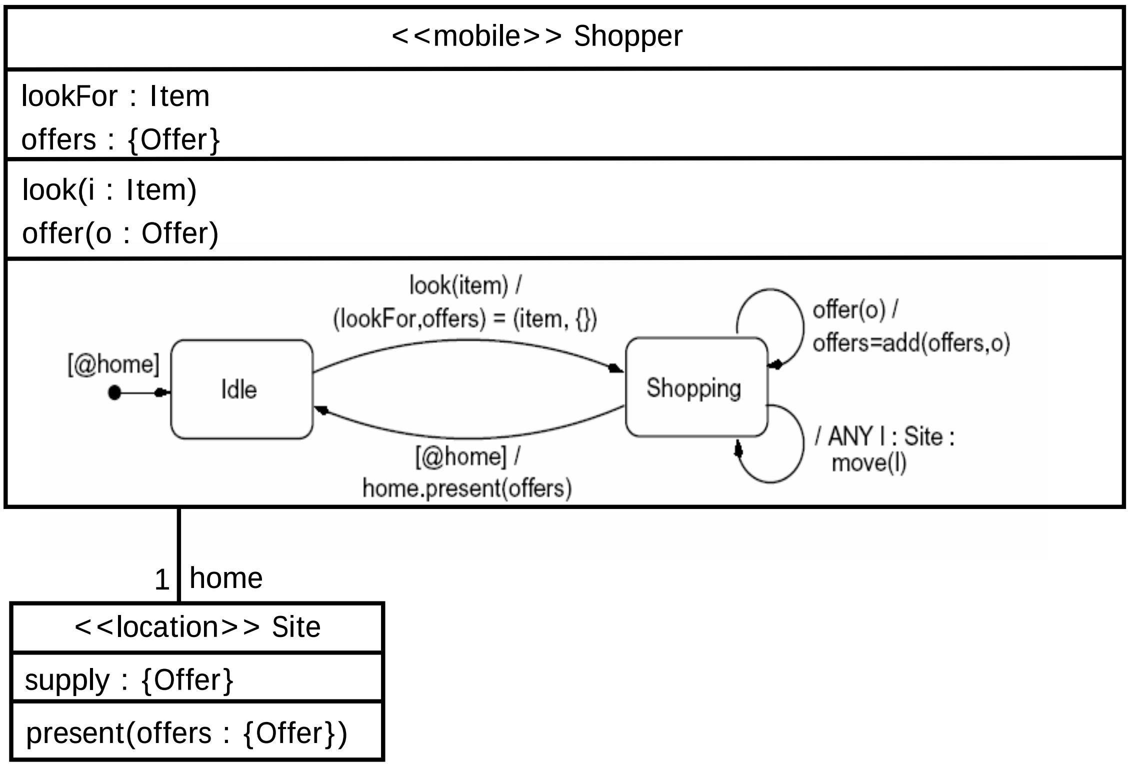
- guards $e_1 \prec e_2$
- $@e \triangleq \mathsf{self} \prec e$
- special move action

Simplifications

- no state hierarchy
- no pseudo states
- only asynchronous communication
- actions restricted to

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

5. Mobile state machines: Example Shopper

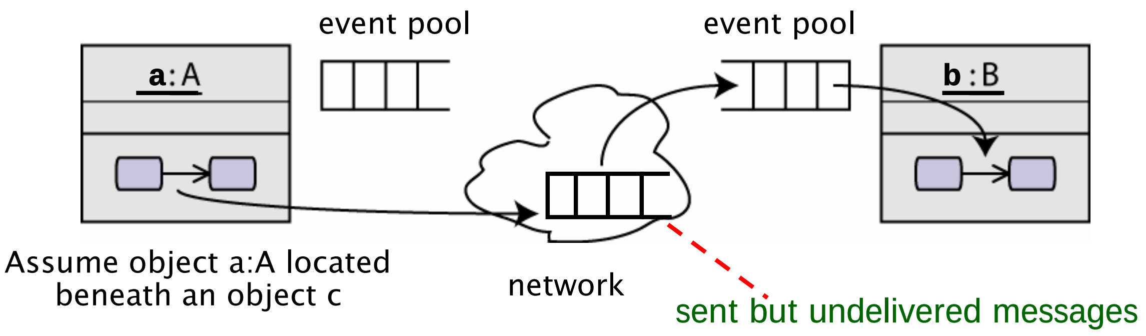


5. Mobile state machines: MTLA Semantics

- **UML mobile state machines**
  - semi-formal graphical notation
  - semantics and formal foundation non-obvious
  - no notion for reasoning on mobile systems
  - no abstract notion of refinement
- **Translation of state machines to MTLA**
  - Define control states and event queues
  - Translate every transition
  - Specify the behaviour of the whole state machine/several state machines

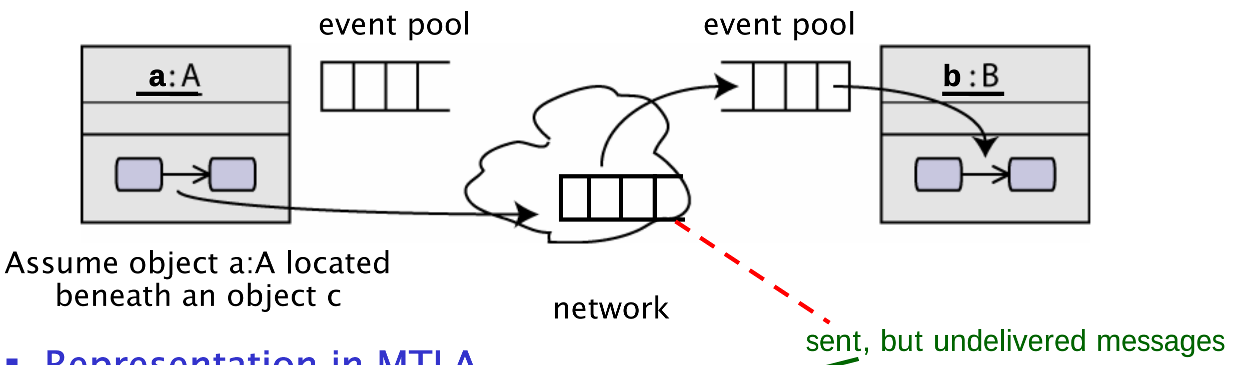
5. Semantics of state machines
Basic Idea

- Communicating state machines

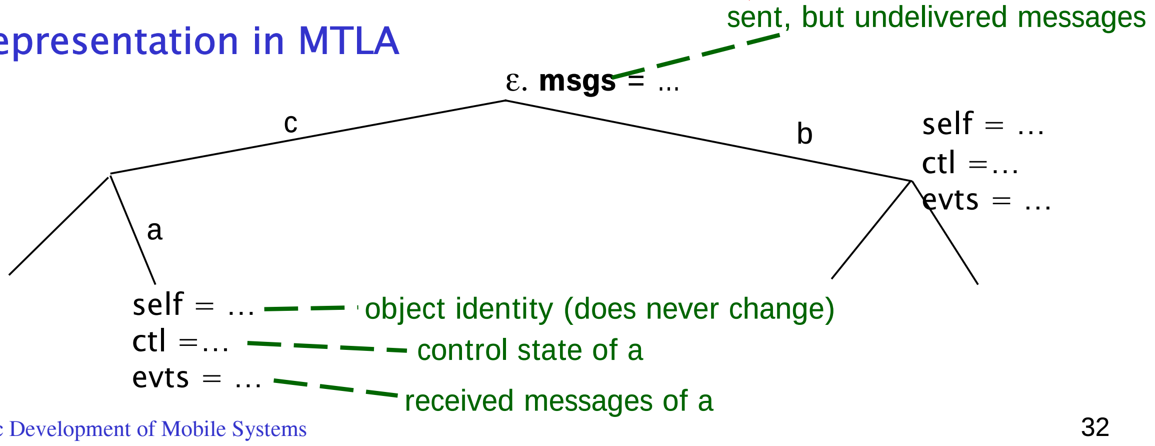

event pool
event pool
a:A
b:B
Assume object a:A located beneath an object c
network
sent but undelivered messages

5. Semantics of mobile state machines
Basic Idea

- Communicating state machines

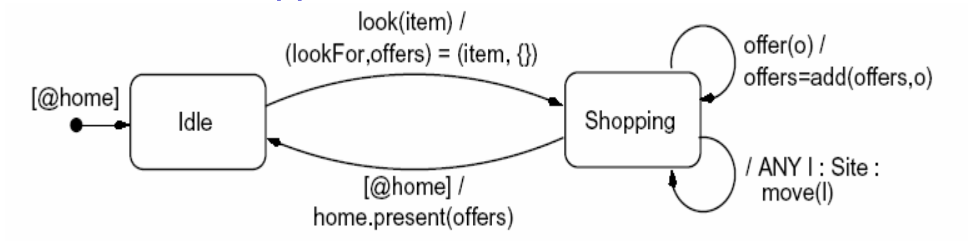

event pool
event pool
a:A
b:B
Assume object a:A located beneath an object c
network
sent, but undelivered messages

- Representation in MTLA



5. Semantics of mobile state machines: Example Transition Translation

- State machine of shopper



- Translation to MTLA

Translation of guard [@home]

$\bigwedge_{l \in Loc} [\text{false}]_{l.ag}$

$$
\begin{aligned}
Present(ag) \equiv\ & \wedge \bigvee_{l \in Obj} ag.home = l.self \wedge l.ag\langle \mathbf{true} \rangle \\
& \wedge ag.ctl = \mathsf{Shopping} \wedge ag.ctl' = \mathsf{Idle} \\
& \wedge \mathrm{UNCHANGED}\ \langle ag.lookFor, ag.offers, ag.home, ag.evts \rangle \\
& \wedge msgs' = msgs \cup \{\langle ag.home, \mathsf{present}, ag.offers \rangle\} \\
& \wedge Stationary(ag) \\
Move(ag) \equiv\ & \bigvee_{l \in Loc} \wedge l.self \in Loc \\
& \qquad \wedge ag.ctl = \mathsf{Shopping} \wedge ag.ctl' = \mathsf{Shopping} \\
& \qquad \wedge \mathrm{UNCHANGED}\ \langle ag.lookFor, ag.offers, ag.home, ag.evts \rangle \\
& \qquad \wedge msgs' = msgs \wedge \varepsilon.ag \gg l.ag
\end{aligned}
$$

Translation of ANY l : move(l)

6. Refinement of mobile systems

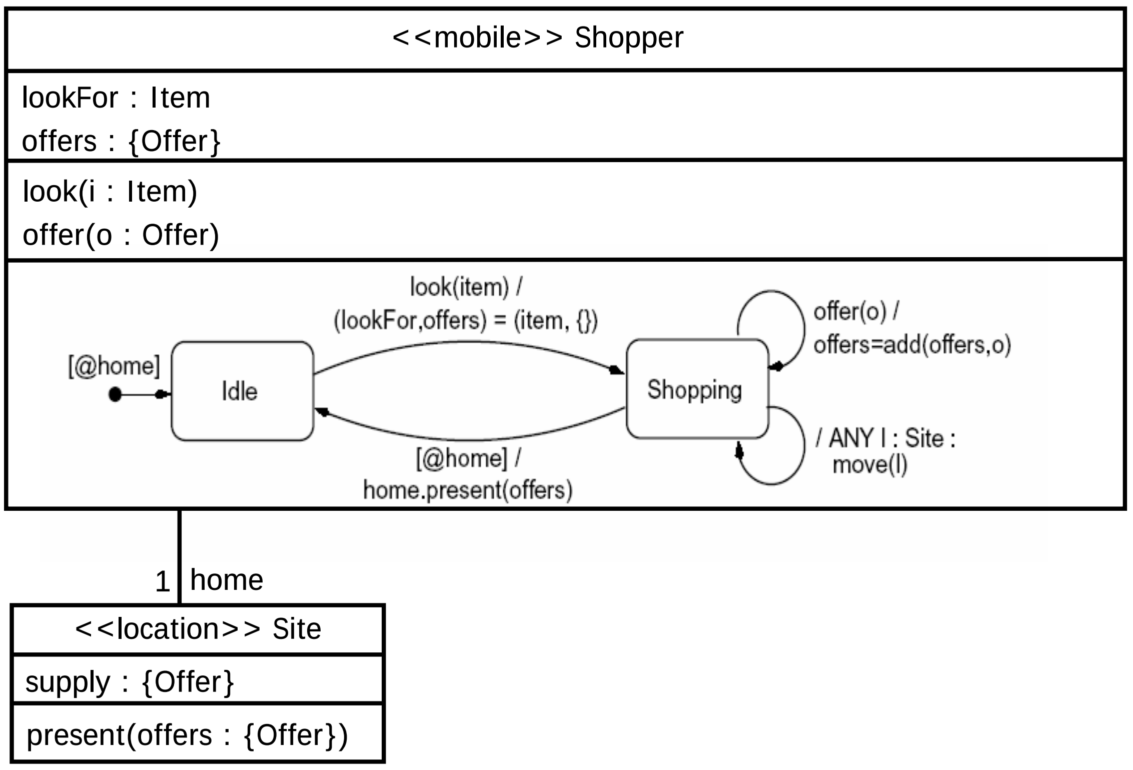
- **Operation refinement**
  - decompose high-level operations
  - represented by implication (stuttering invariance)

    (Action Refinement as in TLA, see earlier)

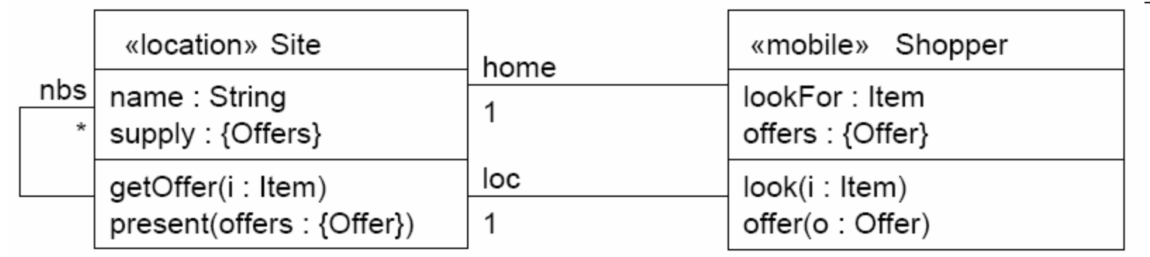
- **Spatial decomposition** (Location Refinement)
  - refine high-level location *n* into a tree (with root named *n*)
  - in general also distribute local state of *n*

- **Virtualisation of locations** (Location and Move Refinement)
  - implement high-level location *n* by structurally different hierarchy
  - preserve external behavior : *n* hidden from high-level interface

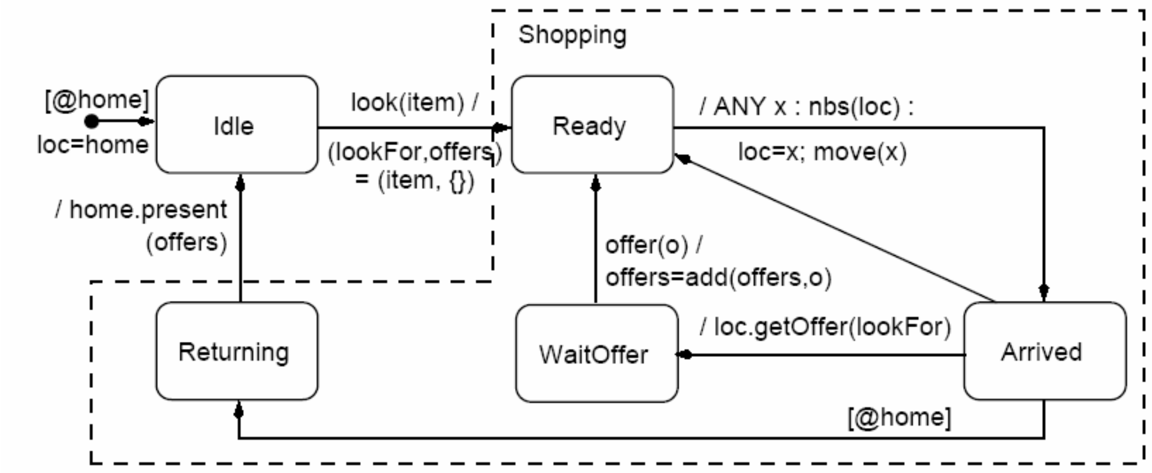
6.1 Refinement of Mobile State Machines: Operation Refinement of Shopper



6.1 Operation Refinement of Shopper



- Refine state Shopping by 4 states:



6.1 State Machine Refinement

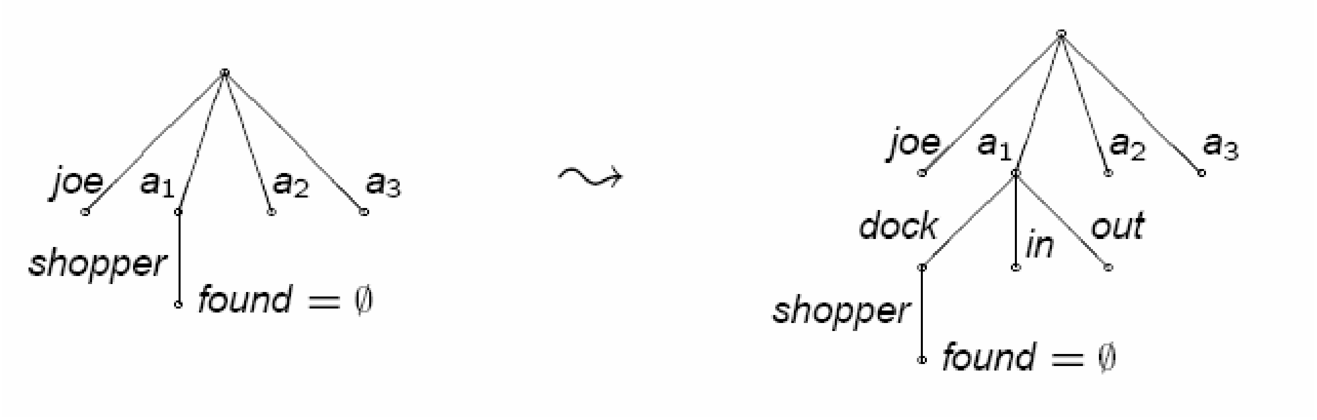
- **State machine refinement is based on**
  - an invariant Inv^R of the refined state machine,
  - an abstraction function Abs: $State^R \rightarrow State^M$
    mapping the states of R to the corresponding states of M,
  - a global hypothesis H on the refined system (e.g. Assumptions H on the spatial hierarchy.
- **Example**
  - **Invariant** of refined shopper:
    (ag.ctl = Returning => @home) $\wedge$ ag.loc $\in$ Site
  - **Abs** maps the states
    Ready, Arrived, WaitOffer, and Returning to state Shopping
  - **Global hypothesis**: Here an assumption on the spatial hierarchy:
    $\forall$ s $\in$ Site : nbs(s) $\subset$ Site

6.1 Example: Refinement Proof

- **Inductive invariant:** RfndShopper => Inv(ag):
  The only non-trivial case is the transition $\text{Arrived2Returning}^{\text{RfndShopper}}$ to state Returning: because of the guard, Inv(ag) holds in the post state
- **Step simulation**
  - **Initial State:** H $\wedge$ $\text{Init}^{\text{RfndShopper}}$ => $\text{Init}^{\text{Shopper}}$(ag): Obvious
  - Any **action of RfndShopper** implies validity of corresponding high-level action:
    - $\text{look}^{\text{RfndShopper}}$ implies $\text{look}^{\text{Shopper}}$: holds obviously (actions have identical definition);
    - $\text{move}^{\text{RfndShopper}}$ implies $\text{move}^{\text{Shopper}}$: holds because of global hypothesis on neighbours;
    - $\text{Arrived2Ready}^{\text{RfndShopper}}$: stuttering step for Shopper;
    - $\text{Arrived2WaitOffer}^{\text{RfndShopper}}$: stuttering step for Shopper;
    - $\text{offer}^{\text{RfndShopper}}$ implies $\text{look}^{\text{Shopper}}$: holds obviously (actions have identical definition);
    - $\text{Arrived2Returningr}^{\text{RfndShopper}}$: stuttering step for Shopper;
    - $\text{Returning2Idle}^{\text{RfndShopper}}$ implies $\text{present}^{\text{Shopper}}$: holds because of inductive invariant.

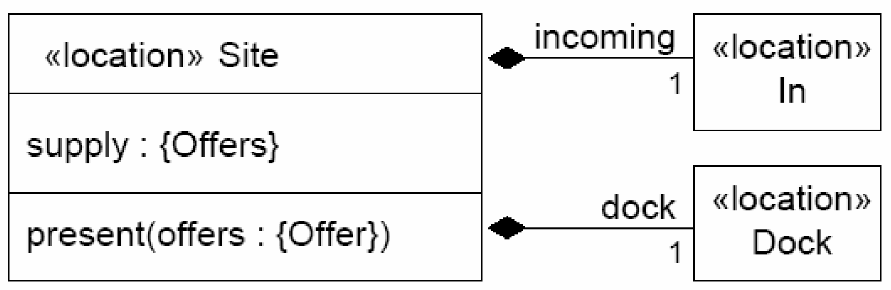
6.2 Spatial decomposition

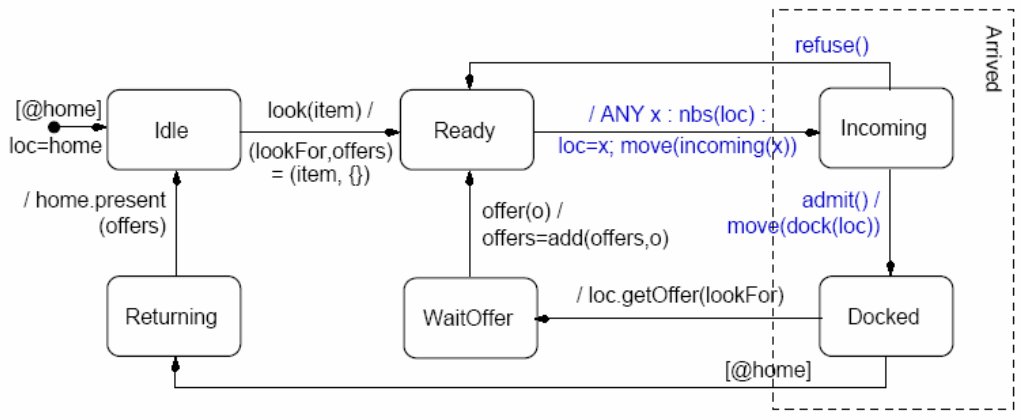
Suppose visiting agents are kept in a “dock” location

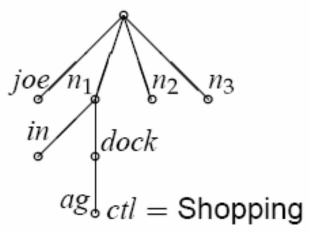


- Still conforms to the original specification
  - formula *Shopper* doesn’t mention locations *dock*, *in*, *out*
  - location *shopper* is still below location a_1

6.2 **Application to State Machines Introducing sublocations**







6.2 Application to State Machines Introducing sublocations

- Acceptable spatial refinement
  - **Invariant** of docked shopper:

    $(ag.ctl = Incoming \Rightarrow @loc) \wedge ag.loc \in Site$
  - **Abs** maps the states

    Incoming, Docked to state Arrived
  - **Global hypothesis**:

    Each site contains and is associated with an "in" location and a "dock" location

$$\bigwedge_{l \in Site} \wedge\ l.l\_in\langle true\rangle \wedge l.l\_dock\langle true\rangle$$
$$\wedge\ incoming(l.self) = l\_in.self \wedge dock(l.self) = l\_dock.self$$

6.2 Spatial decomposition in detail

Refined move actions

- Ready2Incoming ≡ **move to *incoming* location maps to high-level move**

  $\wedge\ ag.ctl = Ready \wedge ag.ctl' = Incoming \wedge \ldots$

  $\bigvee_{l \in Loc} (l.self \in nbs(loc) \wedge ag.loc' = l.self \wedge \varepsilon.ag \gg l.l\_in.ag)$

  Because: $\varepsilon.ag \gg l.l\_in.ag \equiv (ag\langle true\rangle \wedge \circ\, l.l\_in.ag\langle \mathbf{true}\rangle \wedge keep_{ag})$

  implies $(ag\langle true\rangle \wedge \circ\, l.ag\langle \mathbf{true}\rangle \wedge keep_{ag}) \equiv \varepsilon.ag \gg l.ag$

- Incoming2Docked ≡ **move to *docked* location invisible at high level**

  $\wedge\ ag.ctl = Incoming \wedge ag.ctl' = Docked \wedge \ldots$

  $\wedge \bigvee_{l \in Loc} (ag.loc = l.self \wedge \varepsilon.ag \gg l.l\_dock.ag)$ (well-defined because of hypothesis)

  Because: Invariant @loc implies $l.ag\langle true\rangle$;

  with $\varepsilon.ag \gg l.l\_dock.ag$ we get

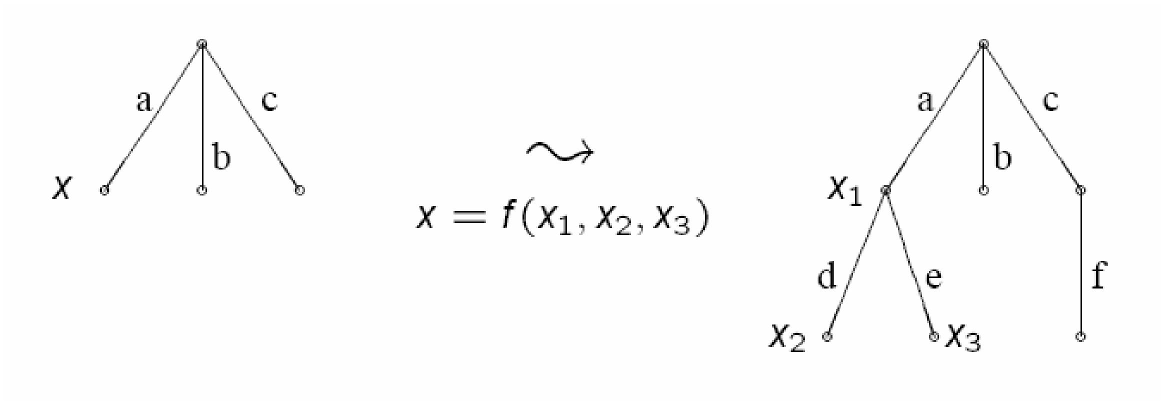
  $l.ag \gg l.l\_dock.ag \equiv (l.ag\langle true\rangle \wedge \circ\, l.l\_dock.ag\langle true\rangle \wedge keep_{ag})$

  This implies $(l.ag\langle true\rangle \wedge \circ\, l.ag\langle true\rangle \wedge keep_{ag}) \equiv \mathbf{l.ag} \gg \mathbf{l.ag}$

➢ The refined specification again **implies** the original one.

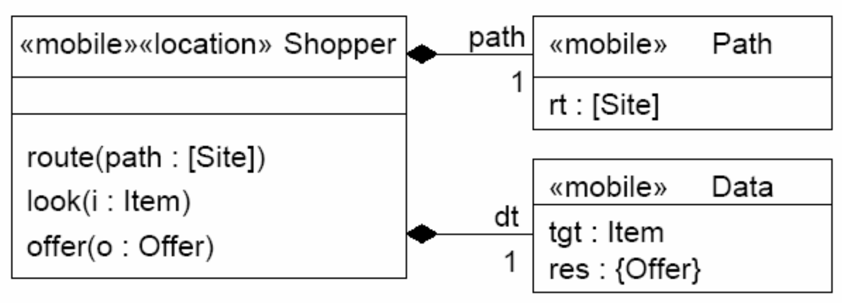
6.2 Spatial decomposition: general case

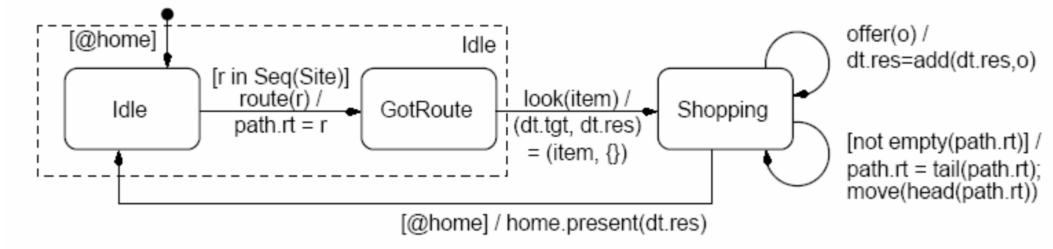
- Usually, decomposition requires distribution of state

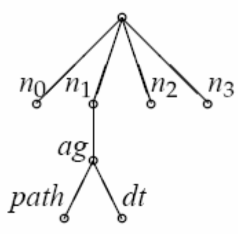


- Refinement is then expressed as   Impl => ∃ a.x : Spec
- local state variable *x* hidden from high-level interface; refinement mapping for realising *x* has to be defined

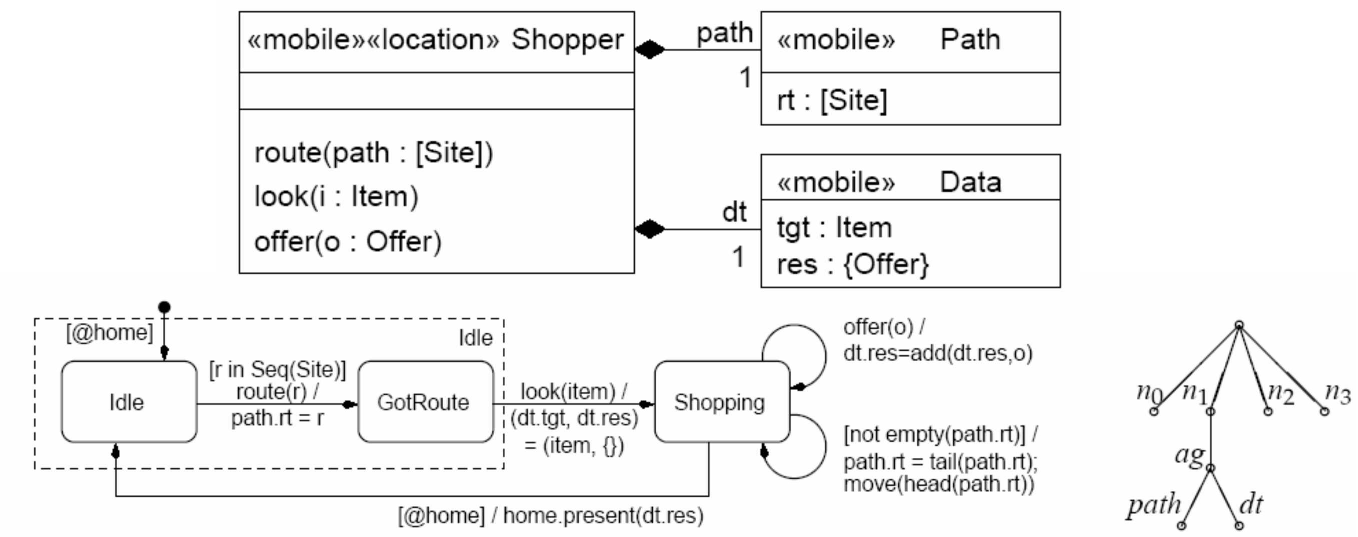
6.2 Application to State Machines: Distribution of agent state







6.2 Application to State Machines: Distribution of agent state

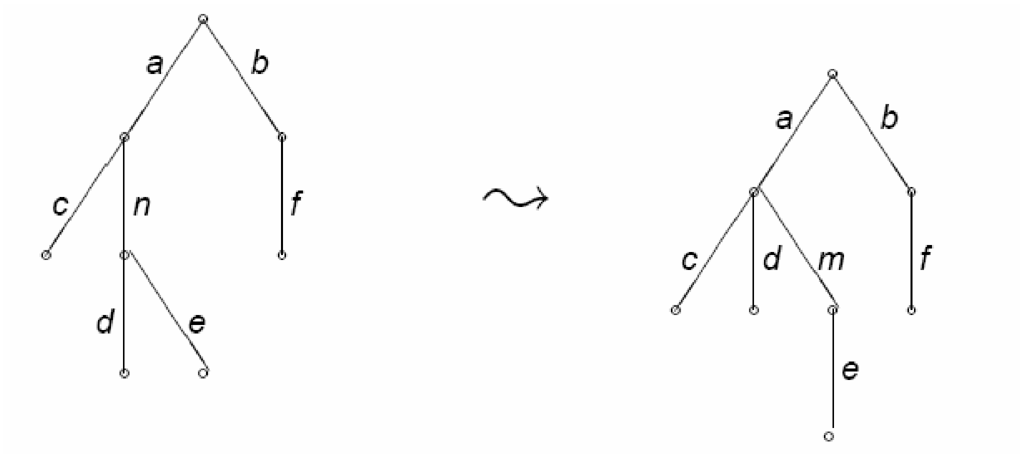


Straightforward extension of proof obligations

- hiding of high-level state components (*lookFor*, *offers*)
- extend refinement mapping to compute hidden state
  - *dt.tgt* → *lookFor*, *dt.res* → *offers*
- invariant ensures preservation of observable behavior

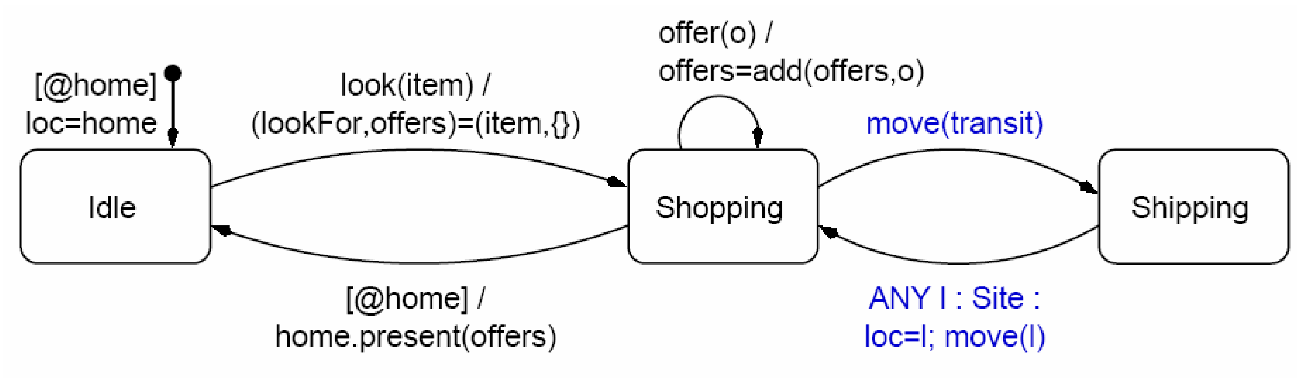
6.3 Virtualisation of locations

- Modify spatial hierarchy



- Location *n* hidden from interface: Impl => ∃*n* : Spec
- Preserve external behavior, except for location *n*

6.3 Application to State Machines: Slow Shopper



- Modification of spatial hierarchy with transit **not** in Site
- non-atomic moves invalidate $(\mathrm{V}_{l \in \mathrm{Site}}\ l.ag\langle true\rangle)$
- have weaker refinement at system level

  Impl => ∃ ag : Spec

! Nonstandard def of ∃ !

Summary: MTLA and Mobile State Machines

- **MTLA – Mobile Temporal Logic of Actions :**
  - Specification logic of mobile systems
  - Spatio-temporal refinement
- **Mobile UML state machines**
  - support move actions and location information
  - Formal Semantics in MTLA
- **Spatial refinement concepts** explained at UML level
  - state machine refinement (operation refinement)
  - introducing sublocations
  - distribution of agent state
  - virtualisation of locations