Performance Analysis: a Creative Art or a Routine Activity?

Vincenzo Grassi

Università di Roma "Tor Vergata", Italy
vgrassi@info.uniroma2.it

(in collaboration with: Vittorio Cortellessa, Raffaela Mirandola)

Goals of this presentation

- General guidelines for a systematic approach to non-functional requirements validation

- Methodologies for performance requirements validation in mobile systems

Performance requirements validation

Art?

Routine?

- A popular performance analysis book:
  - Raj Jain, *The **Art** of Computer Systems Performance Analysis*, J. Wiley & Sons

Performance requirements validation

- Goals:
  - intellectual pleasure for who performs the analysis

  - useful indications for who designs computing systems and software applications

  - the second goal can be achieved only if performance requirement validation is strongly embedded within the design process

Requirements in system design

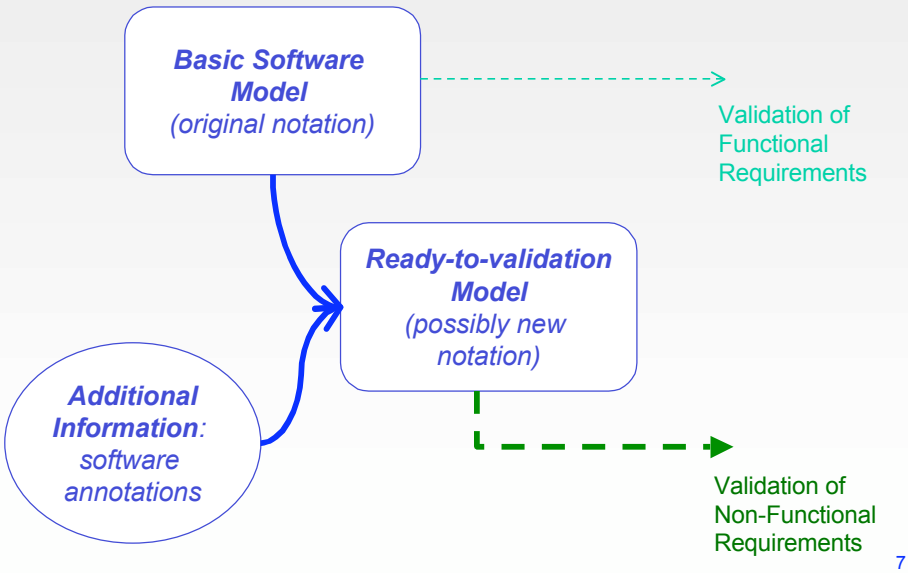
- **Functional** : statements of services the software system should provide, how it should react to particular inputs and behave in particular situations
- **Non-functional** (**NFR**): constraints on the services offered by the software system affecting the software quality
  - a performance requirement is a special case of NFR
- Examples:
  - **functional** : a password is required to be input after the user logs in
  - **non-functional**: the operation *f* must end, on average, 2 seconds after user input

The current status of NFRs validation

- Common practice: software models mostly validated versus functional requirements rather than versus non-functional ones
  - Different (and often not available) skills required for NFR modeling and validation
  - Short time to market, i.e. quickly available software products (even if performing quite poorly) “seem” to be more attractive nowadays!
  - Little information related to NFRs is usually available early in the lifecycle because related decisions are taken later

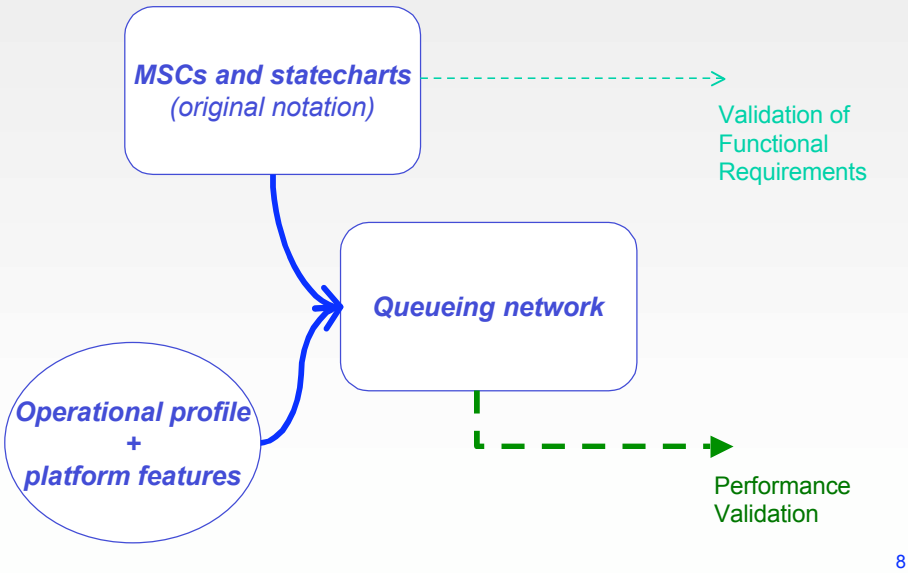
- But early validation of NFRs is important!
  - In the early phases of the lifecycle a validation of NFRs may prevent late inconsistencies hard to fix
  - NFRs are ever more critical in modern (possibly distributed) component-based software systems

A general scheme for NFR validation …



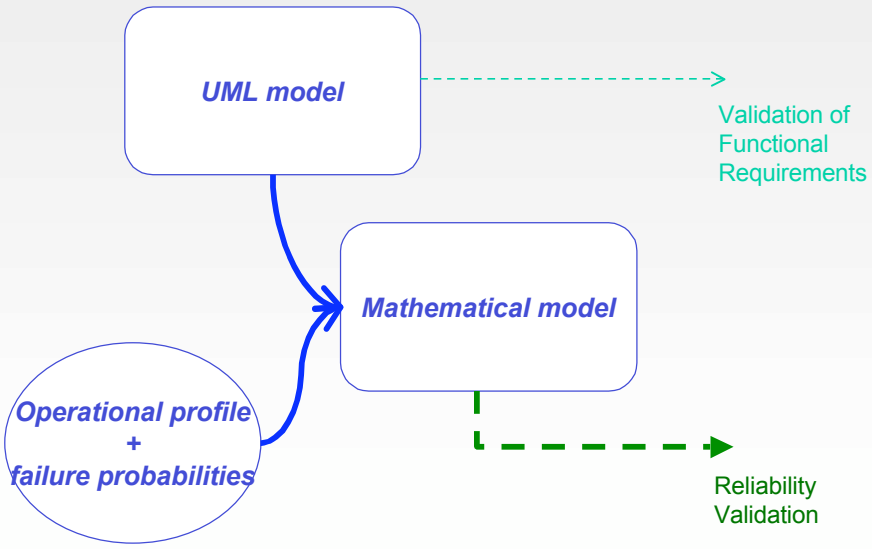
7

… an example ...



8

… and yet another example



9

Filling the gap between software development and NFR validation

- Goal: minimize the “perturbation” introduced into the usual development process
- How to achieve it :
  - reduce the freedom degrees
    +
  - minimize additional required skills
    - define a ”NFR validation space”
    - define a path within that space
    - minimize the effort required to implement the path

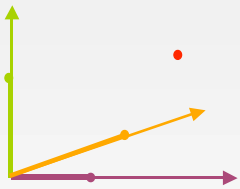
10

Dimensions of the "NFR validation space" (1)

- System domain *(SYS_DOM)*
- Original notation for the basic system model *(START_NOT)*
- Non-functional attribute to be validated *(NFA_TYPE)*
- Type and amount of (additional) missing information to collect *(MISS_INF)*
- Target (ready-to-validation) model notation *(TARG_NOT)*
- Adopted technique to collect missing information *(COLL_TECH)*
- Solution technique for target model processing *(SOL_TECH)*

Dimensions of the "NFR validation space" (2)

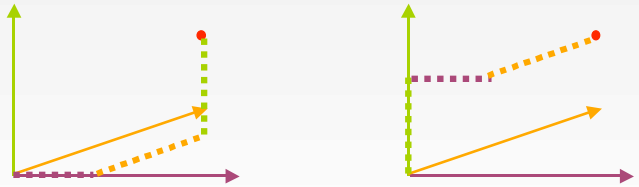
- Selecting a validation methodology means selecting a point in the NFR validation space



- *SYS_DOM* = "don't care"
- *START_NOT* = Unified Modeling language (UML)
- *NFA_TYPE* = reliability
- *MISS_INF* = operational profile, failure probabilities
- *TARG_NOT* = Bayesian stochastic model
- *COLL_TECH* = repository (operational profile), unit testing (failure probabilities)
- *SOL_TECH* = numerical simulation

A path in the NFR validation space (1)

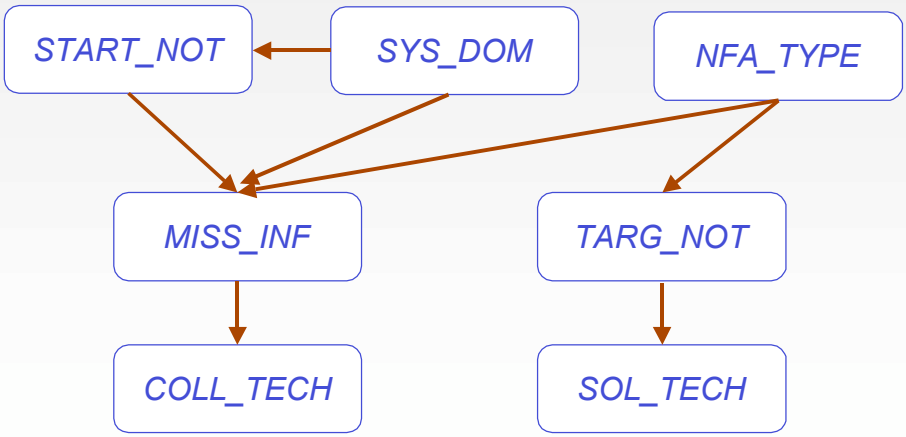
- How to get there?



13

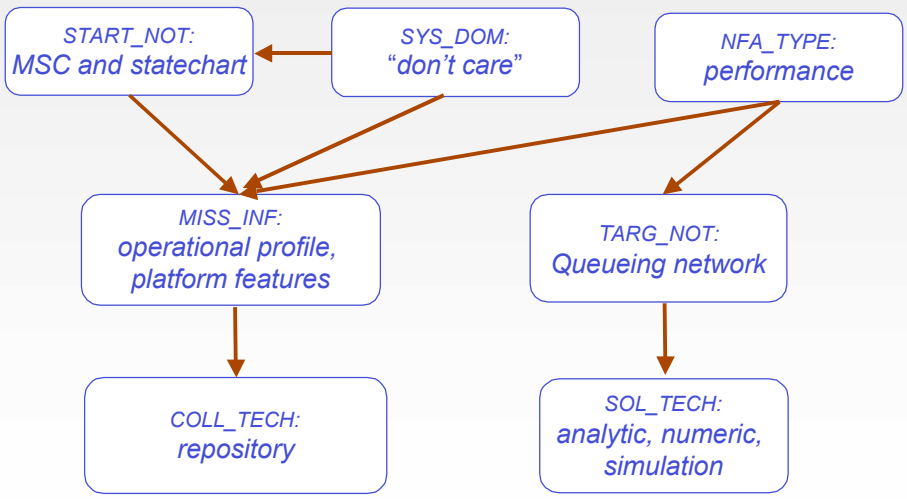
A path in the NFR validation space (2)

- Dependencies among dimensions of the NFR validation space



14

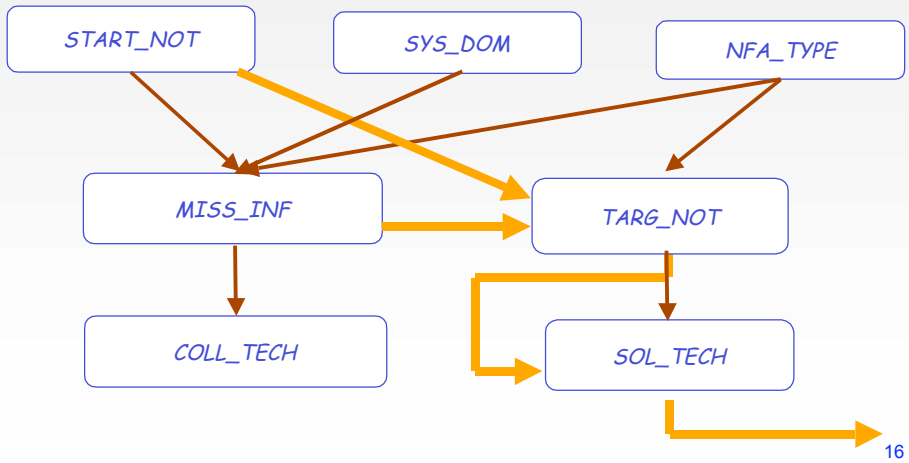
… and an example



15

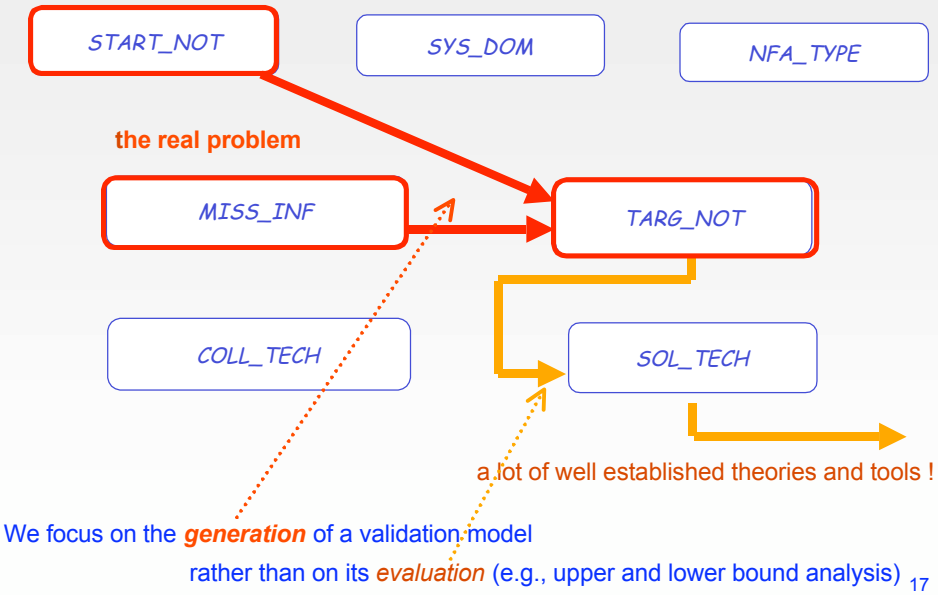
Minimizing the effort to implement the path (1)

- … this is not all the story (just the *identification* of a methodology)
- → : the real path we are interested in for NFR validation (the ***implementation*** of the methodology)



16

Minimizing the effort to implement the path (2)

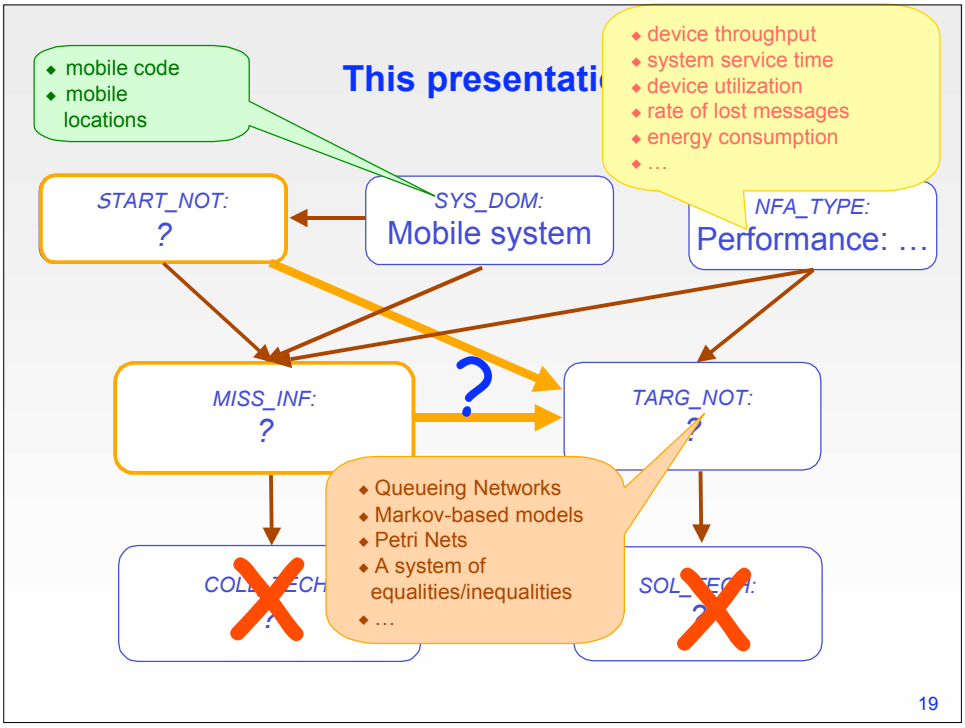


17

From design oriented models to NFR analysis oriented models

- Model transformation
  - central issue for MDD (Model Driven Development) approaches to system development
- Usually intended as transformation (refinement) of "functional" models
  - OMG's MDA (Model Driven Architecture)
    - Platform Independent Models (PIMs)
      - » PIM(1) → PIM(1) → … PIM(m)
    - Platform Specific Models (PSMs)
      - » PIM(m) → PSM(1) → … PSM(n)
- Transformation from "functional" to "non functional" models
  - a different type of transformation
  - should be embedded within MDD approaches
    - based on the same set of tools
      - » MOF, QVT, ...

18



Performance Analysis of Mobile Systems

Vincenzo Grassi

Università di Roma "Tor Vergata", Italy
vgrassi@info.uniroma2.it

(in collaboration with: Vittorio Cortellessa, Raffaela Mirandola, Antonino Sabetta)

SFM-O5

20

Outline

- The “Mobile Systems” domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility “styles”
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

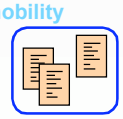
SYS-DOM = “mobile system”

- mobile system: a computing system with mobile entities
- which mobile entities
  - physical
    - computing nodes
    - persons
    - vehicles
    - ...

physical mobility

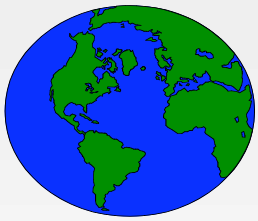
  - virtual
    - stateless code
    - stateful processes
    - file systems
    - ...

virtual mobility

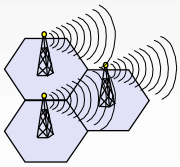


Location spaces for mobile entities (1)

- Physical mobile entities
  - basically, they move in a 2- or 3-dimensional continuous space



  - sometimes, we could be interested in coarser physical location spaces (discrete spaces)
    - cells in a wireless cellular network

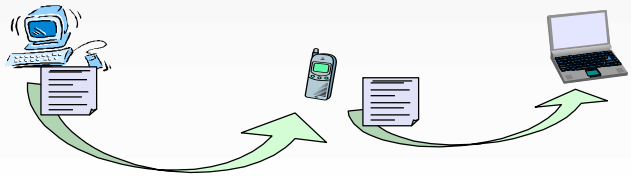


    - buildings/rooms in a campus

    - …

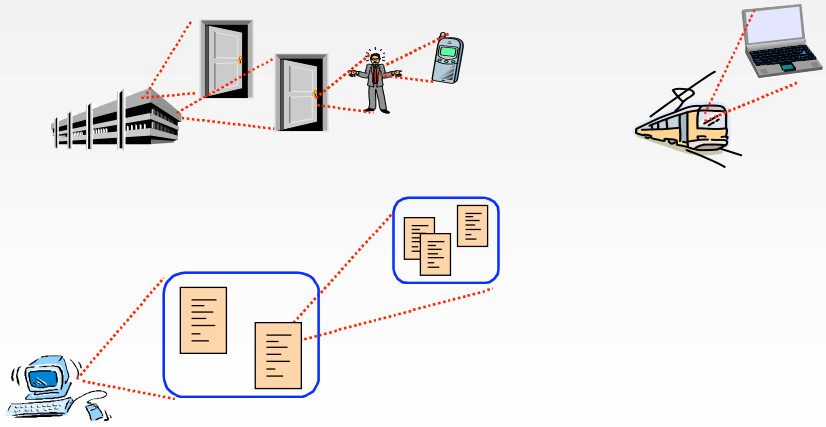
Location spaces for mobile entities (2)

- Virtual mobile entities
  - they move in discrete location spaces
    - computing nodes
    - execution environments
    - ...



Location spaces for mobile entities (3)

- Locations can be nested, and can move too !
  - for both physical and virtual mobility



Outline

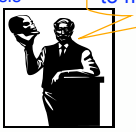
- The “Mobile Systems” domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility “styles”
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

Physical and virtual mobility

- Why a virtual entity moves
  - because the physical entity that hosts it does move
    - "involuntary" movement
  - because its (or of some other virtual entity) intrinsic logic makes it move
    - "voluntary" movement
- "Voluntary" movement of virtual entities (aka *code mobility*)
  - *location-aware* approach to software system design
  - not necessarily related with physical movement
  - a way for taking advantage of /coping with the different attributes of locations of heterogenous and dynamic environments
    - wide area environments
    - (physically) mobile environments
- Physical *vs.* virtual mobility
  - physical mobility: a design constraint
  - virtual mobility: a design tool
  - both are possible only if a "channel" exists between the source and target location

Code mobility as a design tool

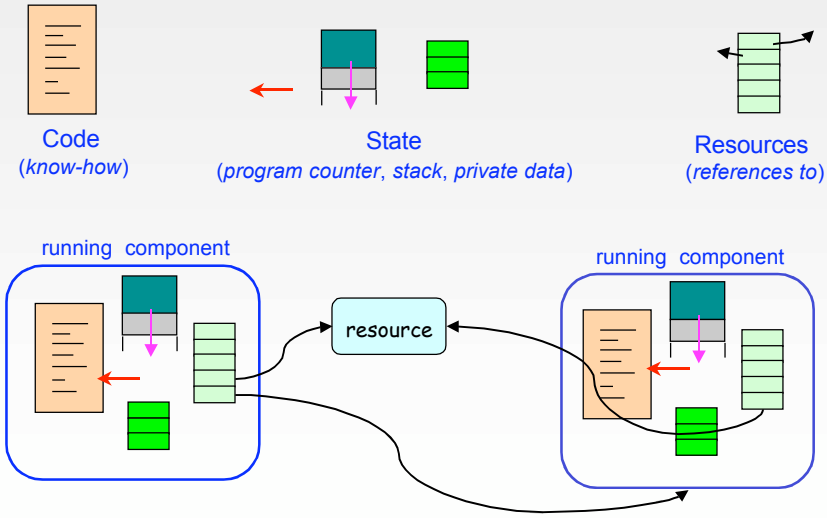
- Code mobility is not process migration
  - process migration is a (distributed) OS issue, out of the control of the software application designer
  - main goal: global load balancing
- Code mobility is an application level design paradigm/ technology
  - moving components becomes a tool in the hands of the application designer
  - mainly used to achieve application level *adaptation*
    - service customization
    - dynamic functionality extension
    - fault-tolerance
    - …
  - one of the main goal of adaptation strategies is to maintain some acceptable performance (QoS) level
    - no general recipe, need of case-by-case analysis



Outline

- The “Mobile Systems” domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility “styles”
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

Structure of components of a distributed (mobile) application

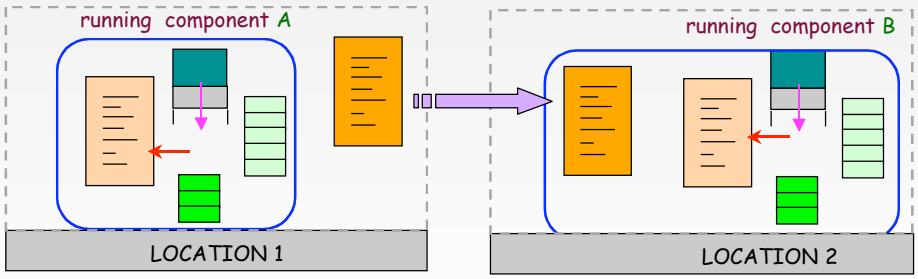


Mobile code styles

- "standard" paradigms to architect mobile code based applications
  - (Fuggetta, Picco, Vigna - *IEEE Trans. on Software Engineering, 24(5) 1998*)
- 1: styles where only code moves
  - Remote Execution
  - Code on Demand
  - …
- 2: styles where code & state move together
  - Strong Mobile Agent
  - Weak Mobile Agent
  - …

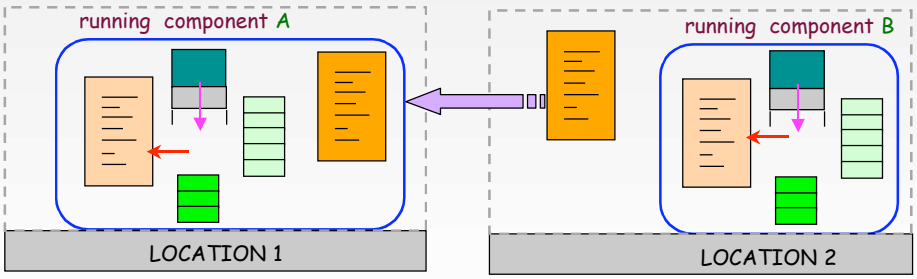
Mobile code styles (1): only code moves

- **Remote Execution** (REV)
  - A sends some code to another location, delegating another running component B to execute it on its behalf



Mobile code styles (1): only code moves

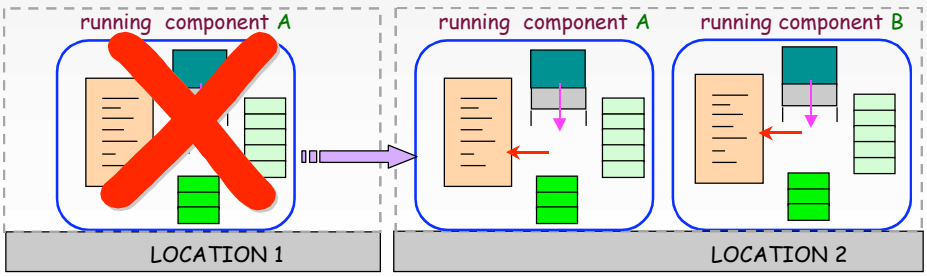
- **Code On Demand** (COD)
  - A fetches some code from another location, extending its functionality



33

Mobile code styles (2): code+state move together

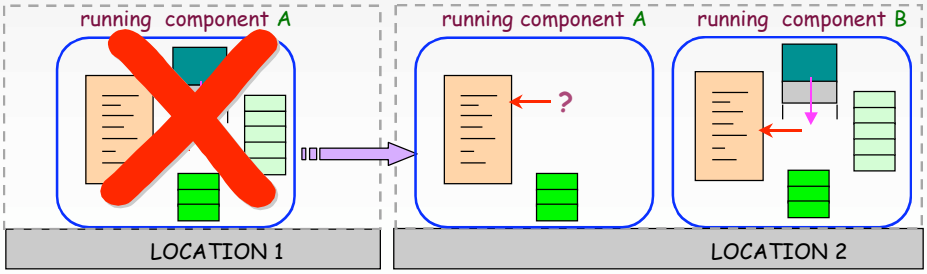
- ***strong*** **Mobile Agent** (MA)
  - A moves to another location, with all its state (program counter+stack+ private data)
  - A can resume its execution at the new location from the exact point where it was stopped



34

Mobile code styles (2): code+state move together

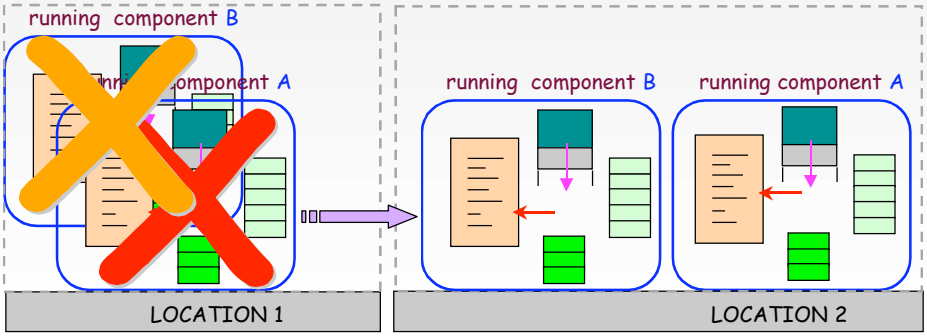
- ***weak* Mobile Agent** (MA)
  - A moves to another location, with its private data only
  - need of methods to determine the point where A resumes execution



35

Mobile code styles (2): code+state move together

- **"Close-To"**
  - B "is associated" to A, and must remain close to it
  - if A moves (or is moved) to a new location, B follows it



36

The mobile systems domain: a summary

- Physical and virtual mobility
- Mobile locations
- Nested locations
- Several different “styles” for virtual mobility
- A last word about mobile systems
  - mobility is relevant (for us) because it changes the system *context*
  - mobility is not the only possible cause of context changes
  - modeling *variable context systems* rather than just mobile systems?

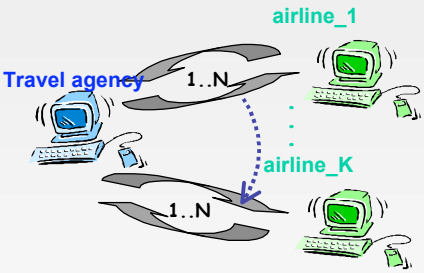
Outline

- The “Mobile Systems” domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility “styles”
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

Which START_NOT when SYS_DOM = “mobile system”?

- START_NOT evaluated with respect to :
  - **expressive power**
    - offered support for the modeling of the relevant features of mobile systems
  - **usability**
    - integration with other already used notations and design processes
    - offered support for “decomposability” issues in system design (“separation of concerns”)

Application example



- a travel agency periodically contacts K airlines to get information about tickets for some itinerary
- N interactions with each airline to get the information

- (Virtual) mobility styles
  - **C/S style**: N RPC’s with each airline
  - **REV style**: the code with the logic for the N interactions is sent to each airline, to be locally processed; only the final answer is sent back by each airline to the agency
  - **MA style**: a “collector” agent travels along the K airlines, collecting locally the information, and then reports it back to the agency

Outline

- The "Mobile Systems" domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility "styles"
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

START_NOT = Process Algebra

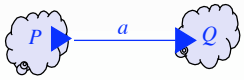
- Formalism to specify system behavior
  - rigorous syntax and semantics
  - compositional features (complex models built systematically from simpler ones)
  - typical syntax:

$P :=$	process definition
$\mathbf{0}$	null action
$\alpha.P$	action-prefix ($\alpha \in Act = \{\alpha \text{ in}x, \text{out}x, \ldots\}$)
$P + P$	non-deterministic composition
$P \parallel P$	parallel composition
...	

  - Operations $\mathbf{in}x$, $\mathbf{out}x$ denote input and output operations along a link named x, used for "synchronizing" parallel processes

$P := \mathbf{out}a.P_1$
$Q := \mathbf{in}a.Q_1$

yes: $P \parallel Q \rightarrow P_1 \parallel Q_1$, no: $P \parallel Q \not\rightarrow P \parallel Q_1$



  - P.A.'s for mobile systems differ in the operations included in the set *Act*

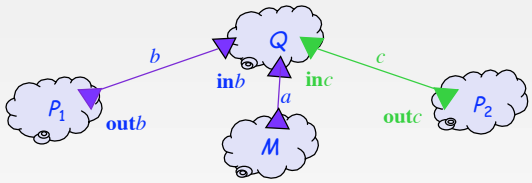
Process algebras & mobility

- expressive power
  - lack of explicit “primitives” to model mobile code styles (COD, REV, MA, …)
  - two ways of modeling locations (and mobility)
    - explicitly
    - implicitly
      - » could be somewhat less intuitive
- usability
  - lack of a clear separation of concerns between application logic and mobility
    - at least in "basic” process algebras for mobility
  - integration with existing design practices is still an issue
  - translation methodology to some TARG_NOT
    - “natural” extension of their syntax and semantics
      - » MISS_INF : expressed through *stochastic process algebras*
      - » TARG_NOT = Markov process

“implicit location”: π-calculus

(Milner, 1999)

- Mobility indirectly modeled as a change in the links a process “sees” to interact with other processes



- models *communication context* changes rather than mobility ⇒ **no explicit location** concept

- Achieved by defining: $Act = \{\pi\ \mathbf{in}x(y),\ \mathbf{out}x(y), \ldots\}$
  - y is a *link name* sent along the channel named x
  - $M := \mathbf{out}a(b).\ \mathbf{out}a(c).M$
    $Q := \mathbf{in}a(y).\ \mathbf{in}y.Q$
    $P_1 := \mathbf{out}b.P_1$
    $P_2 := \mathbf{out}c.P_2$

⇒ $P_1 \parallel M \parallel Q \parallel P_2$

"implicit location": HOπ-calculus

(Sangiorgi, 1992)

- "High-Order" π-calculus : $Act = \{\pi\ \mathbf{in}x(y),\ \mathbf{out}x(y), \ldots\}$
  - y is either a *link name* or a ***process name*** sent along the channel named x ⇨ facilitates modeling of code mobility
- **Travel agency** example: case with two airlines (K = 2)
  - **REV** style
  - C : code with the overall interaction logic with an airline

    SelecBest : selection of the best ticket offer

    F_i : i-th airline (i=1, 2)

    R_i : final response obtained from airline F_i (i=1, 2)

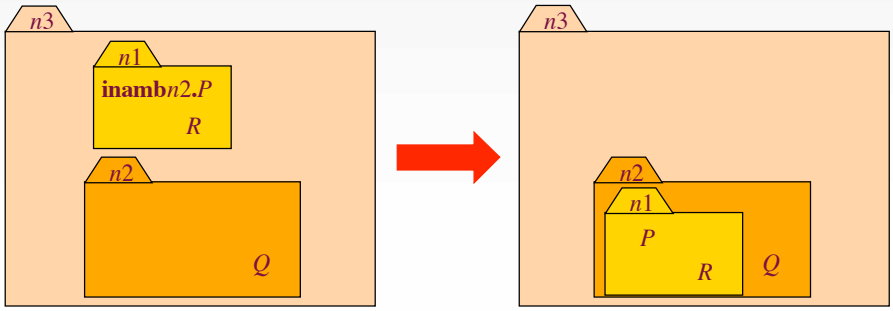
    a_i: channel used for communications between the travel agency and F_i

$$TravAg := \mathbf{out}a_1(C).\mathbf{in}a_1(x).\mathbf{out}a_2(C).\mathbf{in}a_2(x).SelectBest.TravAg$$
$$F_i := \mathbf{in}a_i(y).y.\mathbf{out}a_i(R_i).F_i$$
$$C := \ldots$$
$$SelectBest := \ldots$$
$$Sys := TravAg \parallel F_1 \parallel F_2$$

"explicit location": *Ambient* calculus

(Cardelli, Gordon, 1998)

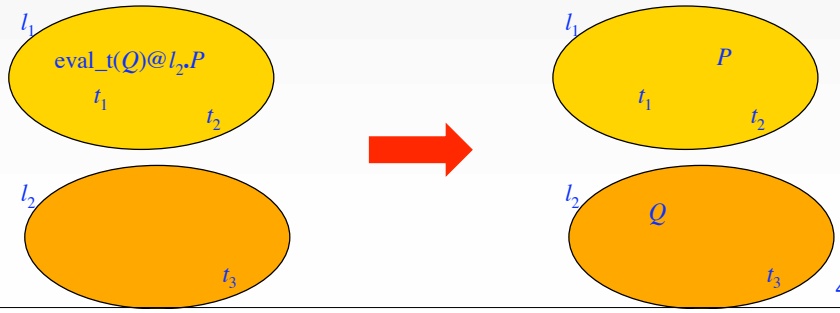
- Concept of "ambient": boundary (with a name) that encloses a set of processes
  - ambients can be entered or exited
  - a process "movement" corresponds to an ambient change ⇨ **explicit location** (= ambient) concept
    - direct modeling of moving and nested locations
- Achieved by defining: $Act = \{\pi\ \mathbf{in}(y),\ \mathbf{out}(y), \mathbf{inamb}n, \mathbf{outamb}n, \ldots\}$
  - n is an *ambient name*



"explicit location": *KLAIM*

(De Nicola *et al.*, 1998)

- **K**ernel **L**anguage for **A**gents **I**nteraction and **M**obility
  - "tuple space" based interactions (by *pattern matching*, based on primitives of the **Linda** coordination language [Carriero-Gelernter, 1989])
  - tuple spaces linked to locations ⇒ **explicit location** concept
    - but lack of direct modeling of moving and nested locations
- $Act = \{$ **in_t**$(t)@l$, **read_t**$(t)@l$, **out_t**$(t)@l$, **eval_t**$(t)@l$, … $\}$
  - t is a *tuple*, and l is a *location*



KLAIM

(De Nicola *et al.*, 1998)

- **Travel agency** example: case with two airlines (K = 2)
  - **REV** style
  - C : code of the overall interaction logic with an airline

    F_i : i-th airline (i=1, 2), located at loc_i

    R_i : final response obtained from airline F_i (i=1, 2)

$TravAg :=$ **out_t**(C, "inforeq")@loc_1.
in_t(x, "inforeply")@loc_1.
out_t(C, "inforeq")@loc_2.
in_t(x, "inforeply")@loc_2.$TravAg$

C := …

$SelectBest$:= …

$F_i :=$ **in_t**(y, "inforeq") @self.y.
out_t(R_i, "inforeply")@self.F_i

$Sys := TravAg \parallel F_1 \parallel F_2$

Outline

- The "Mobile Systems" domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility "styles"
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

from START_NOT = P.A. to TARG_NOT = M(R)P

- starting point: a P.A. with its associated *operational semantics*
  - labeled transition system: graph representing all the possible system evolutions
    - node = system state
    - transition = state change
    - label = identifier of the "activity" that causes a state change
  - transition relation specified by syntax-driven rules: $\frac{Premises}{Conclusions}$
- MISS_INF (in a probabilistic setting)
  - stochastic duration of system activities
  - probability of alternative activities
- (START_NOT=P.A.) + (stochastic MISS_INF) ➔ **Stochastic Process Algebra**
  - Exponentially distributed duration: labeled transition system

    ⇓

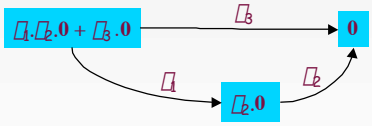
    transition diagram of a **Markov Process**
  - by adding a **reward** to states and/or transitions, we get a **Markov Reward Process**

Example

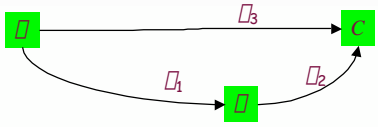
- Transition rules:

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ ,\ \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}\ ,\ \frac{P \xrightarrow{\alpha} P'}{P \| Q \xrightarrow{\alpha} P' \| Q}\ ,\ \frac{P \xrightarrow{\mathbf{out}_x} P', Q \xrightarrow{\mathbf{in}_x} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'}\ ,\ \ldots$$

- Process definition: $P := \alpha_1.\alpha_2.\mathbf{0} + \alpha_3.\mathbf{0}$
- Labeled transition system:



- Exponential activity duration: $P := (\alpha_1, \lambda_1).(\alpha_2, \lambda_2).\mathbf{0} + (\alpha_3, \lambda_3).\mathbf{0}$
- Markov Process:



Another example (travel agency)

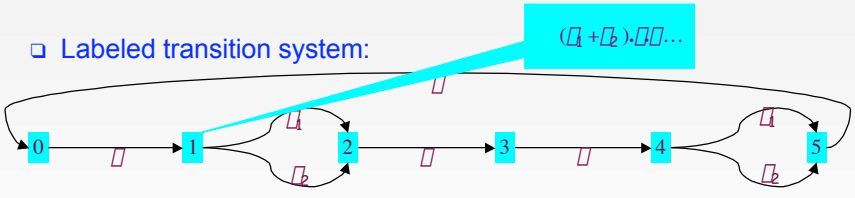
- Process definition: $TravAg := \mathbf{out}a_1(C).\mathbf{in}a_1(x).\mathbf{out}a_2(C).\mathbf{in}a_2(x).TravAg$

$F_i := \mathbf{in}a_i(y).y.\mathbf{out}a_i(R_i).F_i$

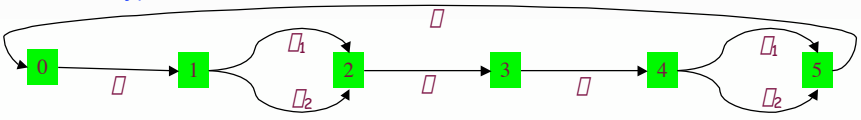
$C := (\alpha_1 + \alpha_2).\mathbf{0}$

$Sys := TravAg \parallel F_1 \parallel F_2$

- Labeled transition system:

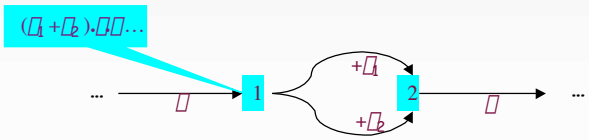


- Markov Process (by associating an exponential duration to each activity):



from ORIG_NOT = P.A. to TARG_NOT = M(R)P: a refinement

- Problem: how to give meaningful values to the exponential transition rates λ_i (and to the reward rates)
  - in real cases, huge number of states
- A proposal (Nottegar *et al.*, 2001): enhance the labels of the transition system with information about the inference rules used in the deduction
  - this information can be exploited to systematically determine the "costs" involved in the execution of a given operation



  - the cost of executing λ_i is given by the cost of action λ_i itself *plus* the cost of some (e.g., OS) scheduling operation (that actually implements the '+' operator)

Process algebras & mobility

- expressive power
  - lack of explicit "primitives" to model mobile code styles (COD, REV, MA, …)
  - two ways of modeling locations (and mobility)
    - explicitly
    - implicitly
      - » could be somewhat less intuitive
- usability
  - lack of a clear separation of concerns between application logic and mobility
    - at least in "basic" process algebras for mobility
  - integration with existing design practices is still an issue
  - translation methodology to some TARG_NOT
    - "natural" extension of their syntax and semantics
      - » MISS_INF : expressed through *stochastic process algebras*
      - » TARG_NOT = Markov process

Outline

- The "Mobile Systems" domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility "styles"
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

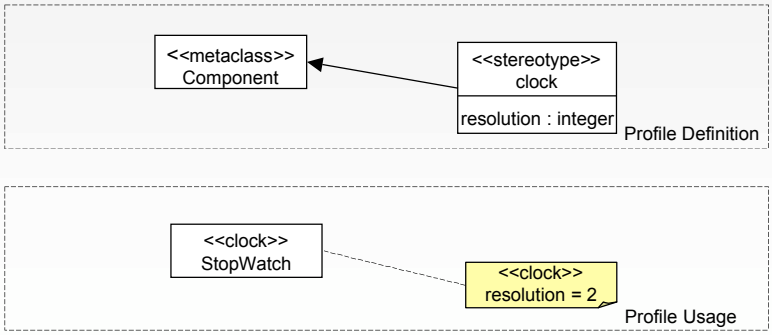
UML & mobility

- expressive power
  - limited support to mobility modeling offered by the "original" UML
  - UML customization through its *extension* mechanisms
    - lightweight extensions ("**profiles**")
    - heavyweight extensions
- usability
  - de facto standard notation for software design
  - promotes separation of concerns through different system views ("diagrams")
    - … but they must be properly used
  - translation methodology to some TARG_NOT
    - semantic problems
    - no "universal" methodology

UML profiles in a nutshell

Standard UML elements can be redefined as to represent concepts that are specific for a given modeling domain.

- Stereotypes
- Tags
- Profiles



START_NOT = UML (*standard*) (1)

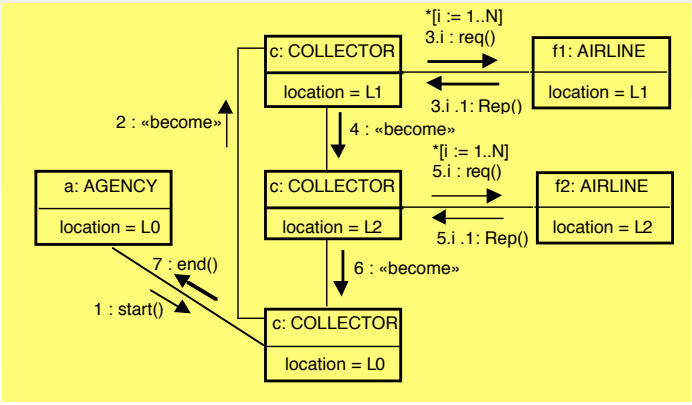
(Booch *et al.*, 1999)

- Standard UML mechanisms to model code mobility, as proposed in standard documents
- Based on the use of the `location` tagged value, and `become` (or `copy`) stereotype
  - `location` applies to a component to express its present location
  - `become` (`copy`) applies to a message to express a location change
    - `become` specifies a location change that preserves identity ➡ MA style, physical mobility
    - copy specifies the creation of an independent copy at a new location ➡ COD, REV styles
- Suggested use within a Collaboration Diagram (CD)
- MISS_INF
  - Size of messages and mobile components
  - Probabilities over non-deterministic choices

START_NOT = UML (*standard*) (2)

(Booch *et al.*, 1999)

- **Travel agency** example: case with two airlines (K = 2)
  - **MA** style



START_NOT = UML (*standard*) (3)

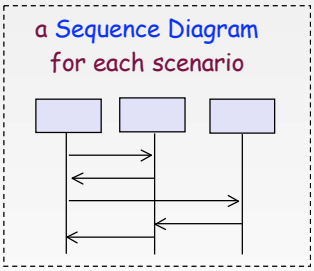
(Booch *et al.*, 1999)

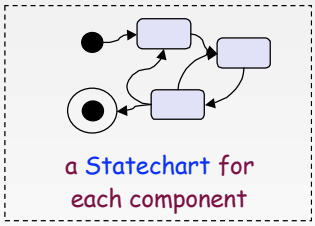
Drawbacks

- No separation of concerns (mix of different views)
  - Interaction *style*: a component behaves as a mobile agent
  - Interaction *logic*: the actual sequence of messages exchanged during an interaction
- Object proliferation
  - Different objects in the CD actually model the same object at different locations
  - Quickly leads to hardly read diagrams when the number of components and/or locations grows

ORIG_NOT = UML (Seq. Diags + Statecharts) (1)

(Merseguer *et al.*, 2000)



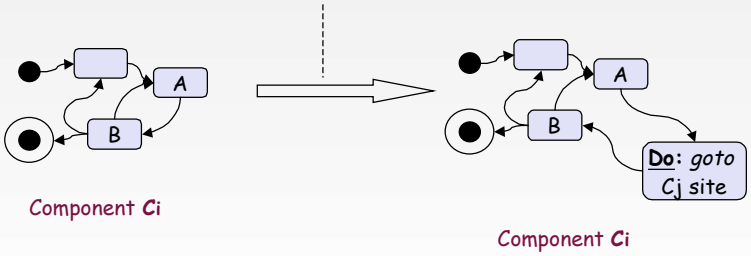


- MISS_INF
  - Size of messages and mobile components
  - Probabilities over non-deterministic choices

ORIG_NOT = UML (Seq. Diags + Statecharts) (2)

(Merseguer *et al.*, 2000)

A state is added to a component statechart to represent the movement of that component to the site of another component

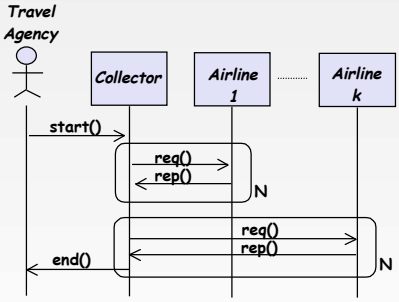


- **SYS_DOM** : logical (code) mobility only (MA style)
- **UML impact**: no extension (standard diags. only)

ORIG_NOT = UML (Seq. Diags + Statecharts) (3)

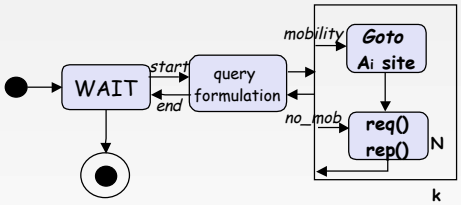
(Merseguer *et al.*, 2000)

TRAVEL AGENCY EXAMPLE

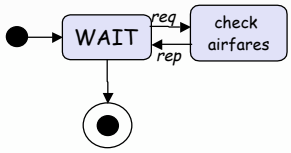


Sequence Diagram

Collector Statechart



i-th Airline Statechart

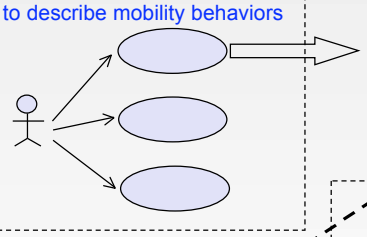


MISS_INF (performance annotations)
- Size of **request** message
- Size of **reply** message
- Size of **collector** component
- Probability of moving

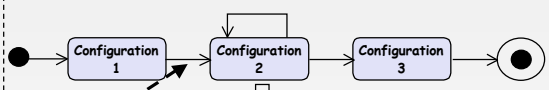
ORIG_NOT = UML (Use Case + Act. Diags) (1)

(Balsamo, Marzolla, 2003)

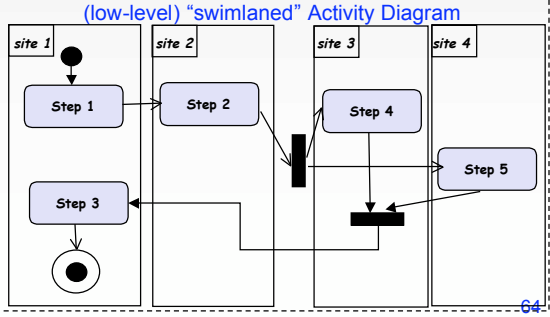
A Use Case Diagram to describe mobility behaviors



Each behavior is represented by a graph of configurations as an (high-level) Activity Diagram



The logic inside each configuration is modeled by a (low-level) "swimlaned" Activity Diagram

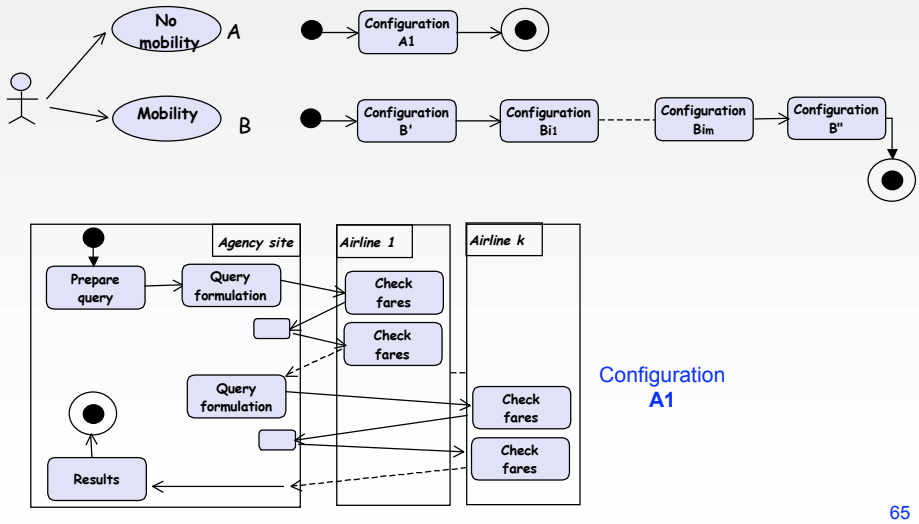


- **SYS_DOM**: mobile code + mobile locations, a (virtual or physical) **mobility action** leads to a configuration change
- **UML impact**: no extension (standard diags. only)

ORIG_NOT = UML (Use Case + Act. Diags) (2)

(Balsamo, Marzolla, 2003)

TRAVEL AGENCY EXAMPLE (MA style)

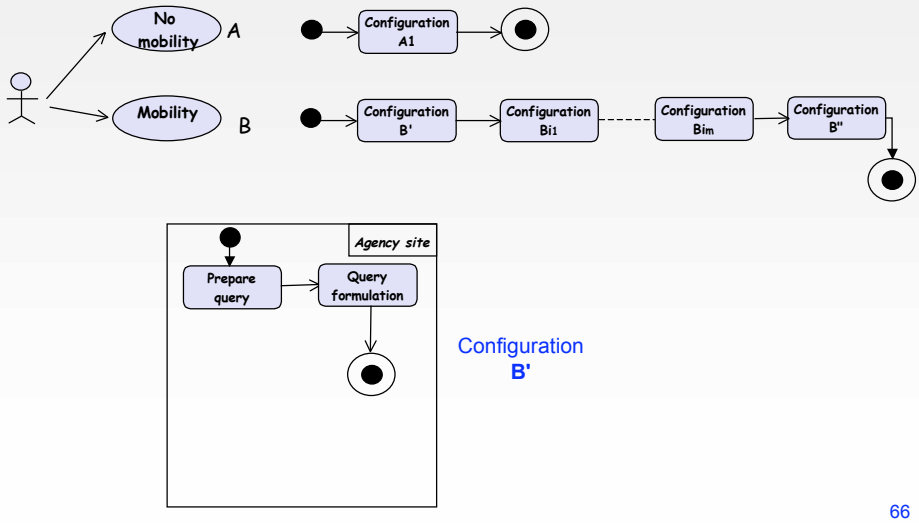


Configuration **A1**

ORIG_NOT = UML (Use Case + Act. Diags) (3)

(Balsamo, Marzolla, 2003)

TRAVEL AGENCY EXAMPLE (MA style)

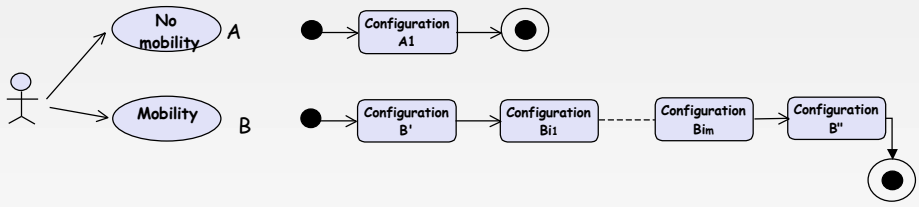


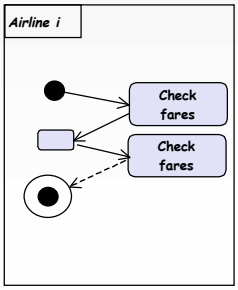
Configuration **B'**

ORIG_NOT = UML (Use Case + Act. Diags) (3)

(Balsamo, Marzolla, 2003)

TRAVEL AGENCY EXAMPLE (MA style)



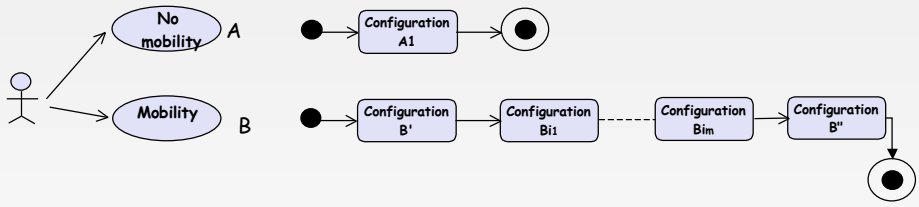


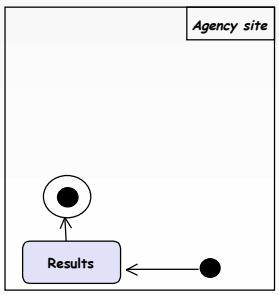
Configuration
B_{ij}

ORIG_NOT = UML (Use Case + Act. Diags) (3)

(Balsamo, Marzolla, 2003)

TRAVEL AGENCY EXAMPLE (MA style)





Configuration
B"

ORIG_NOT = UML (Seq&Coll Diags + stereotypes) (1)

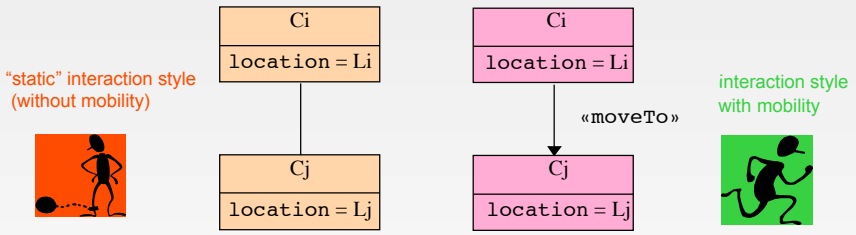
(Grassi, Mirandola, 2003)

- Use different diagrams to get separation of concerns
  - Sequence Diagram (SD) to model interaction ***logic***
  - Collaboration Diagram (CD) to model interaction ***style***
  - (+ Deployment Diagram with `become` stereotype to model mobile locations)

- New `moveTo` stereotype used in CD to model component mobility
  - standard meaning: the source component moves to the target location before starting any sequence of interactions with it
    - possible restrictions adding opportune conditions

ORIG_NOT = UML (Seq&Coll Diags + stereotype) (2)

(Grassi, Mirandola, 2003)

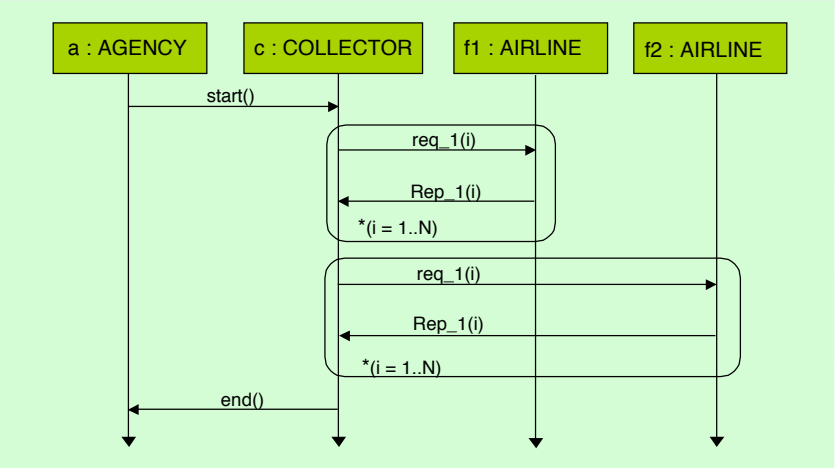


- SYS_DOM : mobile code (basically, MA style) + mobile locations
- UML impact : lightweight extension

ORIG_NOT = UML (Seq&Coll Diags + stereotype) (3)

(Grassi, Mirandola, 2003)

- **Travel agency** example: case with two airlines (K = 2)
  - **SD** for the interaction logic

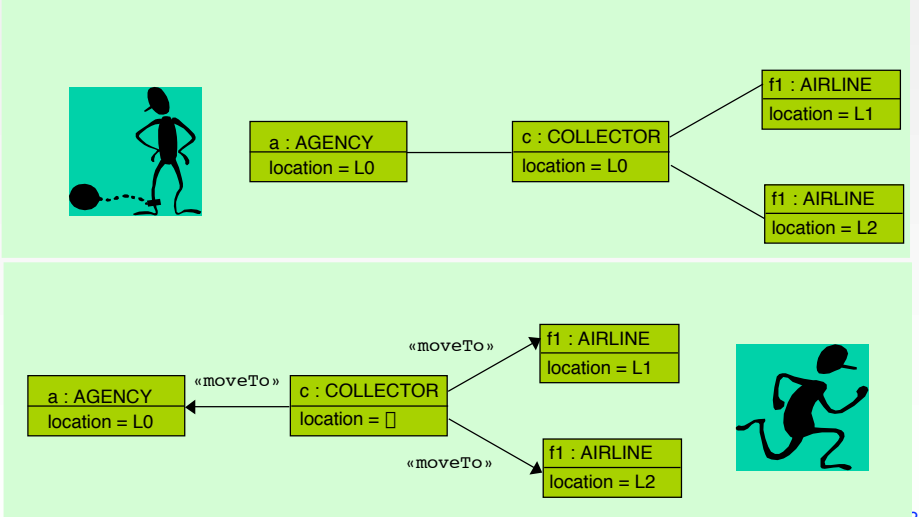


71

ORIG_NOT = UML (Seq&Coll Diags + stereotype) (4)

(Grassi, Mirandola, 2003)

- **Travel agency** example: case with two airlines (K = 2)
  - **CDs** for the interaction style

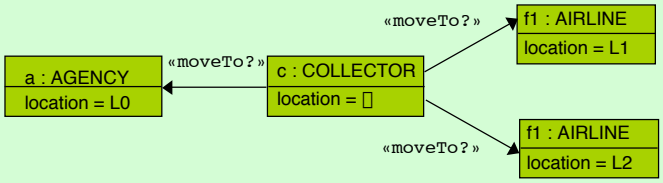


72

ORIG_NOT = UML (Seq&Coll Diags + stereotype) (5)

(Grassi, Mirandola, 2003)

- An extension to model design uncertainty about style
  - `moveTo?` stereotype
    - meaning: the source component "could" move to the target location
  - useful when the designer does not have enough information to choose between mobility and no-mobility styles
- **Travel agency** example
  - uncertainty about the mobility of the COLLECTOR component

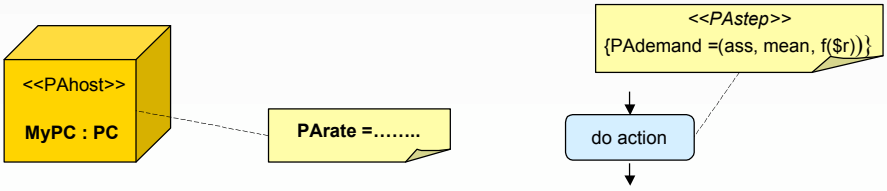


Expressing MISS_INF in UML

- MISS_INF = performance related information
- UML Profile for Schedulability, Performance and Time (SPT)
  - OMG standard
  - set of stereotypes and tags to express performance related information

    +

    modeling framework (*GRM*: *general resource modeling*) to provide a semantic framework for the profile elements and their usage

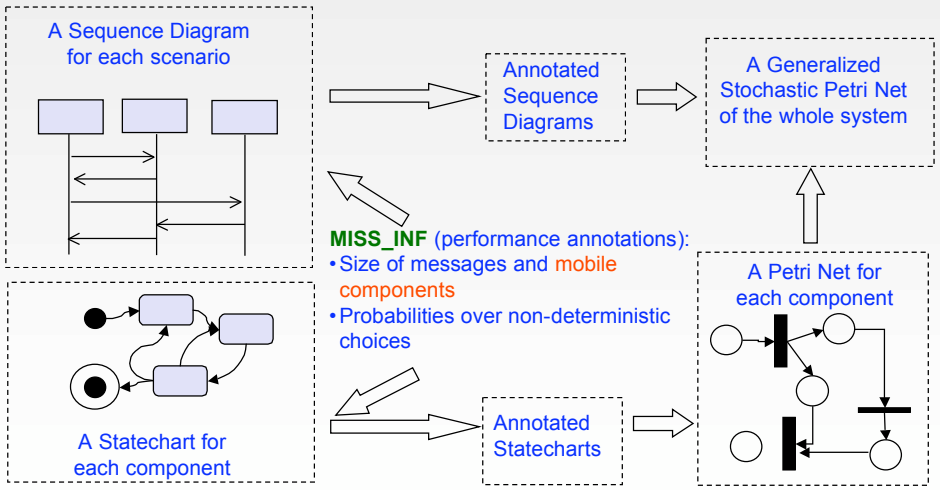


Outline

- The "Mobile Systems" domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility "styles"
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

TARG_NOT = Generalized Stoch. Petri Nets (1)

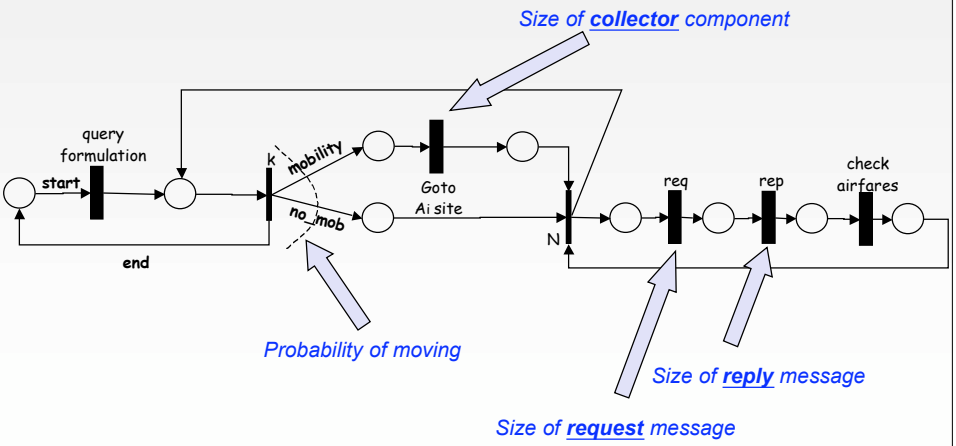
(Merseguer *et al.*, 2000)



TARG_NOT = Generalized Stoch. Petri Nets (2)

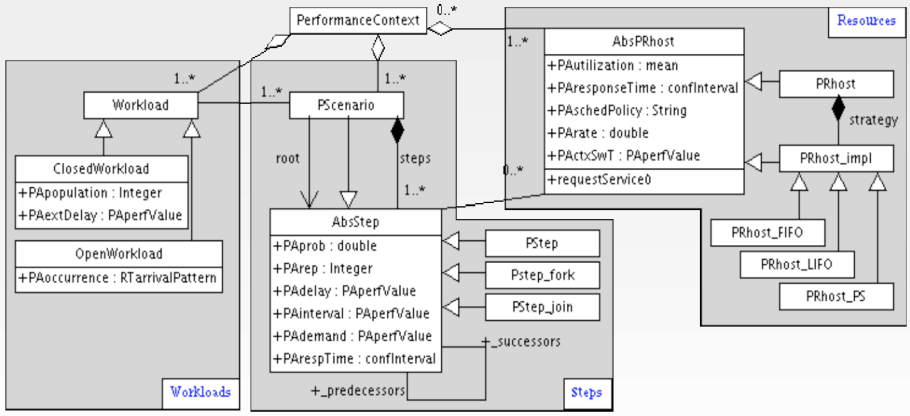
(Merseguer *et al.*, 2000)

TRAVEL AGENCY EXAMPLE



TARG_NOT = Simulation code

(Balsamo, Marzolla, 2003)



MISS_INF (performance annotations):

- Size of messages (in low-level Activity Diagrams)
- Size of mobile components (in high-level Activity Diagrams)
- Probabilities over all non-deterministic choices

TARG_NOT = MRP/MDP or *mob?*-EQN

(Grassi, Mirandola, 2002-2003)

- "Expected" results:
  - *yes/no* on using a code mobility based style
  - if yes, *when* and *where moving* or *not moving*
- TARG_NOT
  - **Markov Reward/Decision Process**
    - **interaction-based** performance measures (generated network traffic, consumed energy for interactions, …)
  - ***mob?*-EG** and ***mob?*-EQN**
    - **E**xecution **G**raph and **E**xtended **Q**ueueing **N**etwork, with additional features to model possible components mobility
    - **delay-based** performance measures (throughput, response time, …)

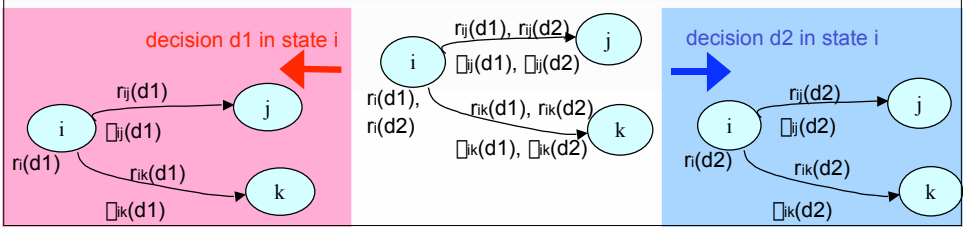
TARG_NOT = MRP/MDP

(Grassi, Mirandola, 2002-2003)

- TARG_NOT features
  - The stochastic features are used to model the (uncertainty about) system dynamics
  - The "reward" features are used to model the "cost" (performance related) of system operations
  - The decisional features are used to model alternative choices about components mobility
- MRP or MDP?
  - ORIG_NOT= UML + `moveTo` stereotype → TARG_NOT= **M**arkov **R**eward **P**rocess
  - ORIG_NOT= UML + `moveTo` & `moveTo?` stereotypes → TARG_NOT= **M**arkov **D**ecision **P**rocess

Markov Decision Process

- MDP: extends MRP by introducing the concept of a set of different *decisions* associated to each state
  - Transition rates and rewards are decision dependent
  - Selecting (deterministically) a decision in a state means that this decision will be taken each time the state is visited
  - A *policy* is a particular selection of decisions in all the model states
  - A MDP actually models a family of different MRPs, one for each possible policy
  - *Optimal policy*: a policy that minimizes a given reward measure (several existing algorithms to find it)



Translation to TARG_NOT = MRP/MDP

(Grassi, Mirandola, 2002-2003)

- Proposed approach : ASAP methodology (Grassi-Mirandola, 2002, 2003)
- *State*: possible configuration of components locations
- State *transition*: occurrence of either an interaction between components, or a location change
  - Component interaction: transition to the same state
  - Location change: transition to a different state
- Transition *reward*: "cost" of the action (interaction or movement) associated to that transition
  - In principle, we associate rewards only to transitions, not to states
- *Decision* associated to a state: selection of a mobility or no-mobility style for components that are source of `moveTo?` messages, when the components location corresponds to that state
  - The selection of a style influences the components behavior, and hence the possible transitions and rewards from a state

ASAP

(Grassi, Mirandola, 2002-2003)

- Basic problem: how information present (or to be added ➔ **MISS_INF**) in the UML diagrams can be exploited to build a MRP/MDP

ASAP Algorithm to generate the state space

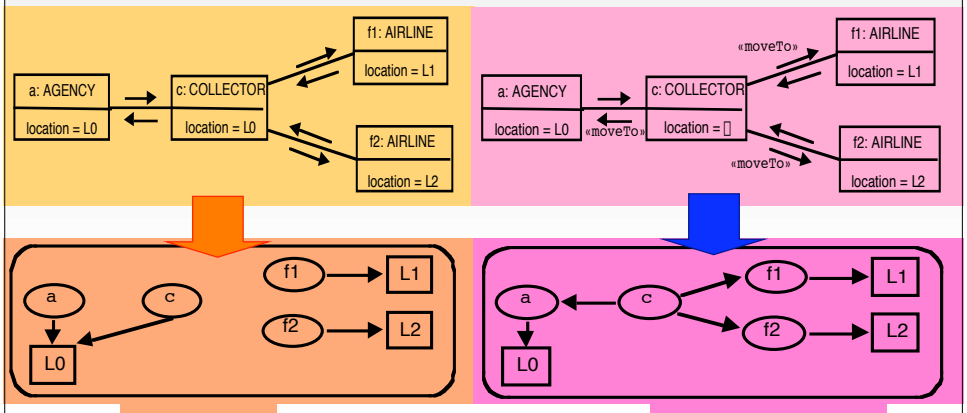
ASAP Algorithm to generate the state transitions

- possible decisions associated to a state
- (decision dependent) transition probabilities
- (decision dependent) transition rewards

ASAP: MRP/MDP state space generation

(Grassi, Mirandola, 2003)

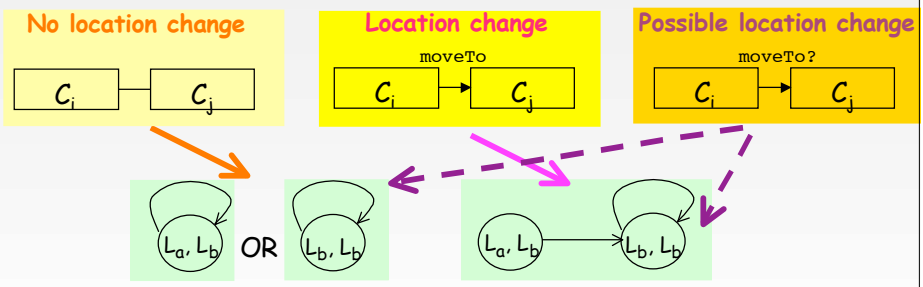
- State space = Loc(C1) x Loc(C2) x … Loc(Cn)
  - Loc(Ci): set of possible locations for component Ci
    - calculated from info obtained by the CD that models the interaction style
- Travel agency example:



ASAP : MRP/MDP transitions generation (1)

(Grassi, Mirandola, 2003)

- Translation rules: from UML-CD patterns to MDP patterns:



- Which transition and reward rates?

ASAP : MRP/MDP transitions generation (2)

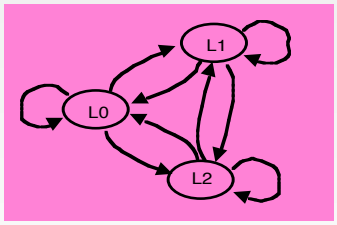
(Grassi, Mirandola, 2003)

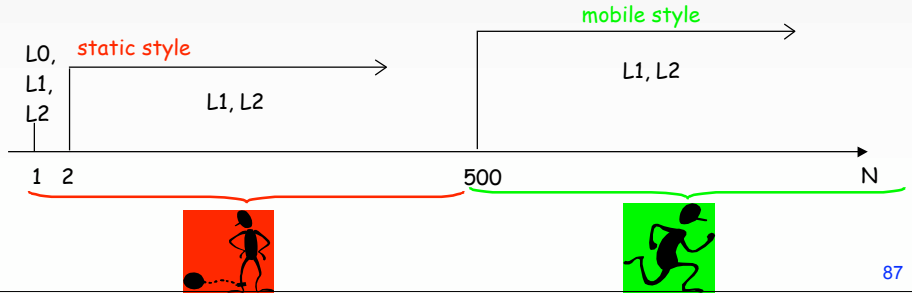
- Needed info:
  - **transition rates**: the frequency of the interactions between components Ci and Cj can be derived from the SD that models the interaction logic, as the ratio of the number of interactions between Ci and Cj and the overall number of interactions
    - **already available info**: number of interactions (can be "counted" in the SD)
    - **MISS_INF**: branching probabilities, usage probability of different SDs, … (derived from annotations added to the UML diagrams)
  - **reward rates**: the average "cost" of an interaction between Ci and Cj is the sum of the costs of all the interactions in SD between Ci and Cj, divided by their number
    - **MISS_INF**: cost of each interaction (derived from annotations added to the UML diagrams)

ASAP: travel agency example

(Grassi, Mirandola, 2003)

- System parameters:
  - N: number of consecutive request-replay messages between the collector and an airline (N = 1, 2, …)
  - *req*: request size (*req* = 1)
  - *Rep*: replay size (*Rep* = 1)
  - *m*: collector size (*m* = 1000)
- Which location for the collector?
  optimal policies:





TARG_NOT = *mob?*-EG/*mob?*-EQN (1)

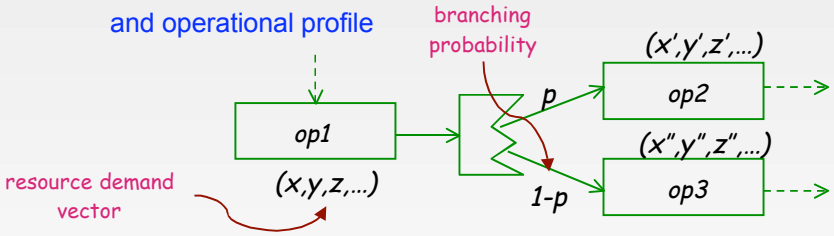
(Grassi, Mirandola, 2002)

- TARG_NOT = *mob?*-EG/*mob?*-EQN
  - basically, EG/EQN with additional features to model uncertainty about *static* or *mobile* style
    - needed to cope with UML models using the `moveTo?` stereotype
    - reduces to classic EG/EQN in case of UML models using only the `moveTo` stereotype
  - suited to analyze delay-based performance measures
- *mob?*-EG or *mob?*-EQN ?
  - interest in stand-alone analysis only → TARG_NOT= *mob?*-EG
  - interest also in contention analysis → TARG_NOT= *mob?*-EQN

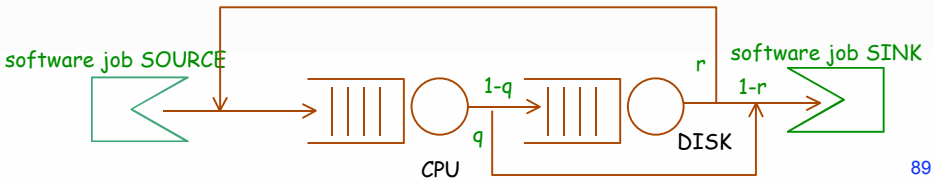
TARG_NOT = *mob?*-EG/*mob?*-EQN (2)

(Grassi, Mirandola, 2002)

- **EG**: basically, a Flow Graph plus info about resource demand and operational profile



- **EQN**: can be used to model services offered by the hardware platform, and the contention to use them
  - the EG can be mapped onto a QN to get a joint software/hardware model (*SPE approach*, C. Smith 1990)



89

Translation to TARG_NOT = *mob?*-EG/*mob?*-EQN

(Grassi, Mirandola, 2002)

- Basic problem: how information present (or to be added ➔ **MISS_INF**) in the UML diagrams can be exploited to build a *mob?*-EG/*mob?*-EQN

- Proposed approach : PRIMAmob-UML methodology (Grassi, Mirandola, 2002)

  UML diagrams & `moveTo` stereotypes
  +
  "performance oriented" annotations

  Semi-automatic derivation

  Queueing network model of the system dynamics

90

Translation to TARG_NOT = *mob?*-EG/*mob?*-EQN : PRIMAmob-UML

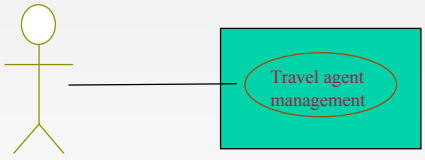
(Grassi, Mirandola, 2002)

Use cases	→	user operational profile
SD+CD	→	*mob?*-meta-EG (only the EG structure)
mob?-meta-EG + DD	→	*mob?*-EG-instance (with demand vector)
DD + *mob?*-EG-instance	→	*mob?*-EQN model

PRIMAmob-UML:travel agency example (1)

(Grassi, Mirandola, 2002)

From use cases to Operational profile

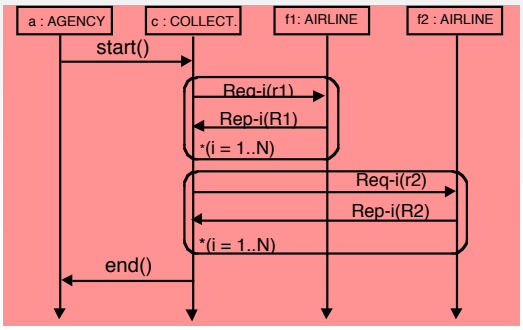


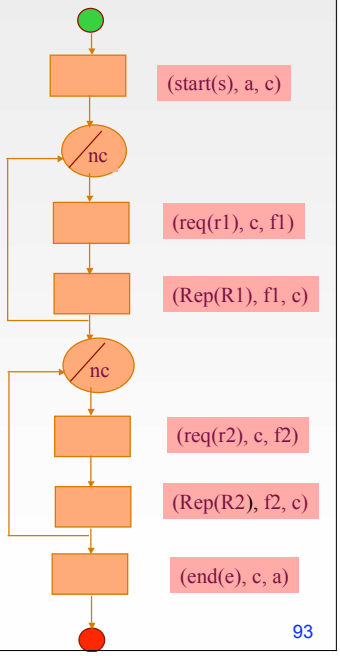
User

P(1)=1

PRIMAmob-UML (2): meta-EG

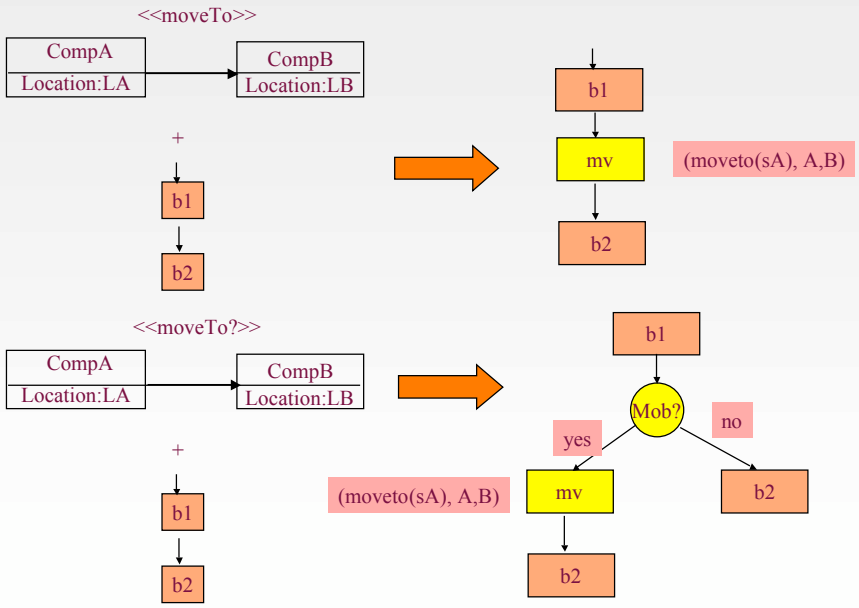
From SD to meta-EG
(EG "structure" only)





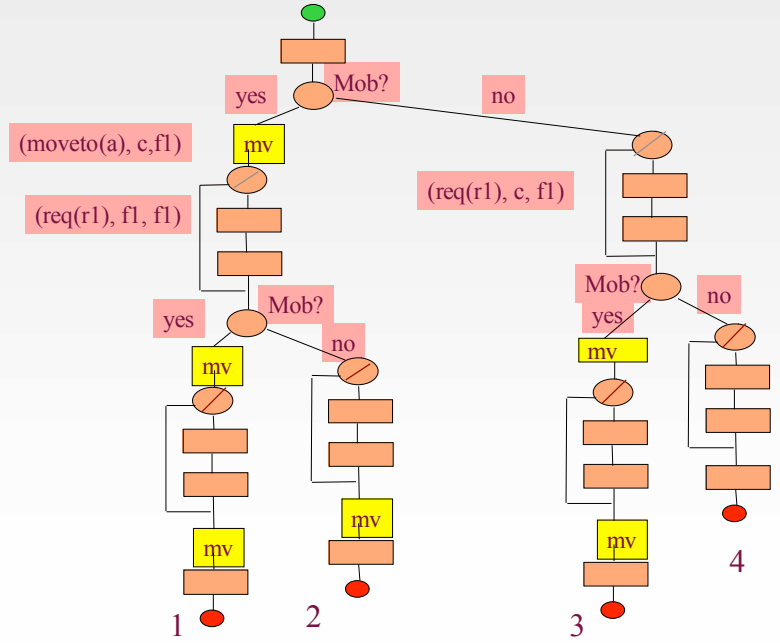
93

PRIMAmob-UML (3): introducing mobility …



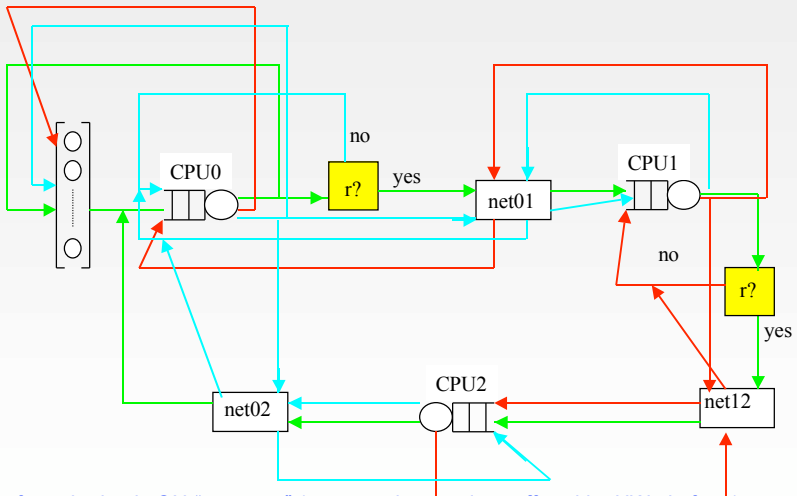
94

PRIMAmob-UML (4): *mob?*-meta-EG



PRIMAmob-UML (5): *mob?*-EQN

From DD + *mob?*-EG-instance to EQN



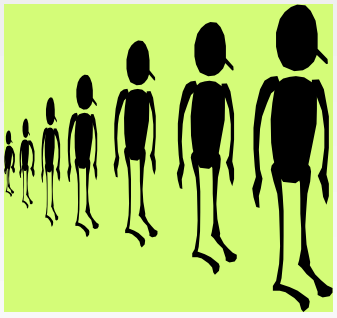
from the basic QN "structure" (representing services offered by HW platform) …
… to the "complete" EQN, obtained by mapping the *mob?*-EG-instance paths onto the QN structure

PRIMAmob-UML (6): contention-based analysis

- r=R=1, *m*=1000
- 3 different platform configurations (characterizations):
  - **C1**: devices CPU0, CPU1 and CPU2 slower than networks
  - **C2**: devices and networks with comparable speed
  - **C3**: devices CPUi (i=0,1,2) faster than networks

By varying the number N_{jobs} of jobs and the number N of interactions with each airline site

PRIMAmob-UML (7): results



N_{jobs}=3,5,10

C1: for N <=2000

C2:

N_{jobs}>3, N<=2000

N_{jobs}=3, N<=2000

C3: for N <=2000

UML & mobility

- expressive power
  - limited support to mobility modeling offered by the “original” UML
  - UML customization through its *extension* mechanisms
    - lightweight extensions (“**profiles**”)
    - heavyweight extensions
- usability
  - de facto standard notation for software design
  - promotes separation of concerns through different system views (“diagrams”)
    - … but they must be properly used
  - translation methodology to some TARG_NOT
    - semantic problems
    - no “universal” methodology

Review of UML based approaches to performance modeling and analysis of mobile systems

- Focused on the definition of translation algorithms from START_NOT to TARG_NOT
- Less attention to modeling (and design) issues
  - unsatisfactory modeling of mobile systems features
  - general lack of attention for design “decomposability” issues

Outline

- The “Mobile Systems” domain
  - physical mobility
  - virtual mobility
    - motivations
    - virtual mobility “styles”
- Methodologies for performance validation of mobile systems
  - Process Algebras -based methodologies
    - Process Algebras as START_NOT
    - transformation to TARG_NOT
  - UML-based methodologies
    - UML as START_NOT
    - transformation to TARG_NOT
    - A UML profile for mobile systems

UML-PMS : a UML profile for mobile systems

- Focused on modeling and design issues
  - attention given to the modeling of mobile system features
    - nested locations, virtual mobility styles, ...
  - "separation of concerns”
    - models of mobility (both physical and virtual) separated from models of the application logic and of the system structure
    - ease “plug-and-play” of different mobility policies at the application modeling level, to support “what if” experiments
  - usability
    - lightweight UML extension
    - compliant with the UML 2.0 specification
      - » only requires UML 2.0 compliant tools
    - compliant with the UML-SPT profile
- Translation to TARG_NOT
  - borrows algorithms from other UML based approaches
    - (as long as they are compliant with UML-SPT and UML 2.0)

UML-PMS : outline

Modeling **mobility** in UML-PMS
- Physical mobility: locations and moving locations
- Virtual mobility: mobile software components
- Dynamics of mobility: <<MobilityManager>>

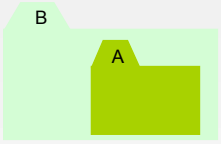
Modeling the missing information about **performance**
- Annotated UML-PMS models: mobility models + SPT

Representing mobile code paradigms in UML-PMS
- Code On Demand (COD)
- Remote Evaluation (REV)
- Mobile Agent (MA)

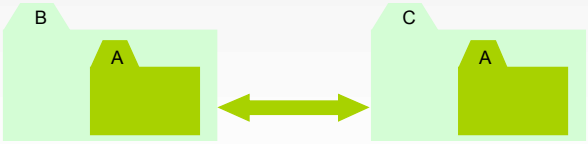
Performance Analysis based on UML-PMS models: an example

UML-PMS : “abstract” location model

- location
  - discrete location space
  - basic model : a nesting relationship between an entity and some container
    - A “is located at” B



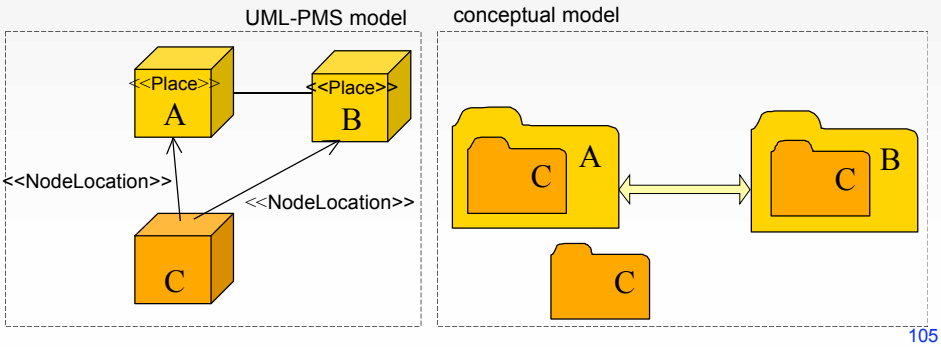
- location change
  - a modification of a nesting ("located at") relationship
    - A “is located at” B A “is located at” C



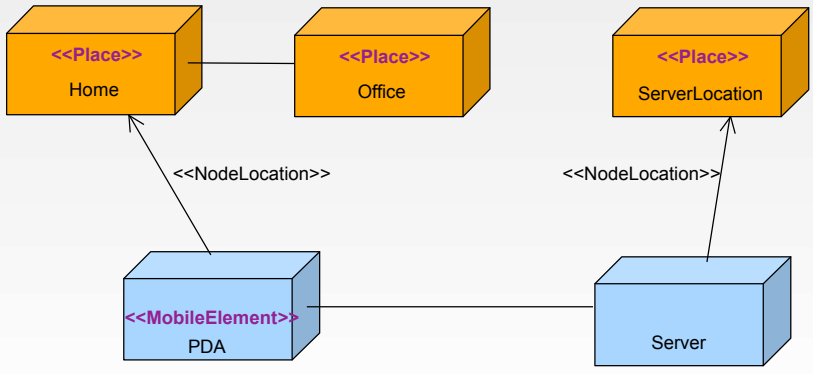
- based on the “ambient calculus” model (Cardelli, Gordon - 1998)
  - recursive relationship
  - uniform model for physical and logical mobility

UML-PMS : physical location model (1)

- physical (mobile) entities modeled as UML 2.0 *Nodes*
  - UML-PMS `<<Place>>` stereotype if they can be the location of other entities
  - UML-PMS `<<MobileElem>>` stereotype if they can change their location
  - UML-PMS `<<NodeLocation>>` stereotype to model the location of a node

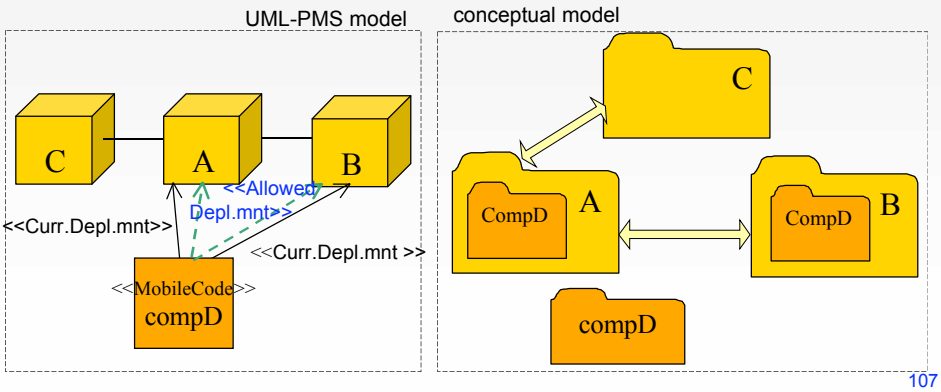


UML-PMS : physical location model (2)

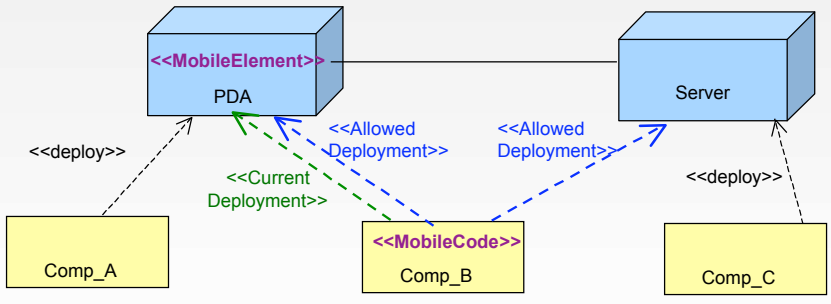


UML-PMS : virtual location model (1)

- virtual (mobile) entities modeled as UML 2.0 *Artifacts*
  - UML-PMS `<<MobileCode>>` stereotype if they can change their location
  - UML-PMS `<<CurrentDeployment>>` stereotype to model their location
  - UML-PMS `<<AllowedDeployment>>` stereotype to model allowed locations



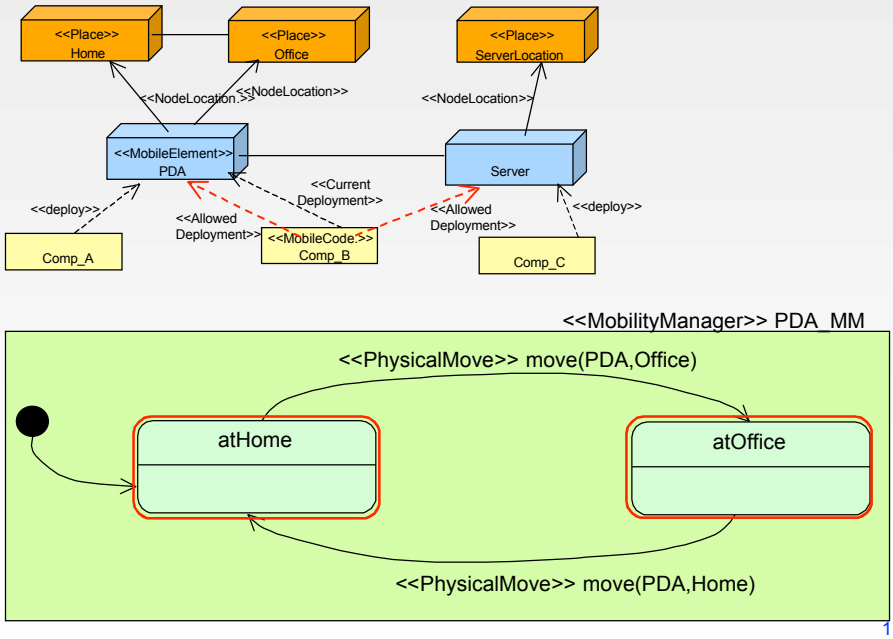
UML-PMS : virtual location model (2)



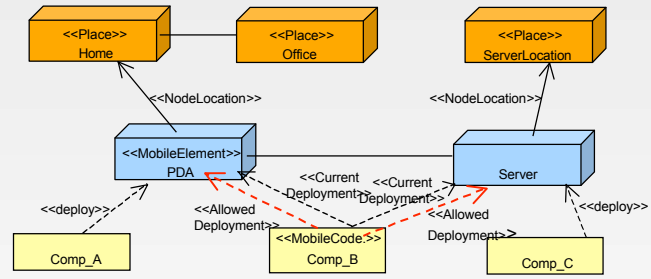
UML-PMS: adding dynamics to the model

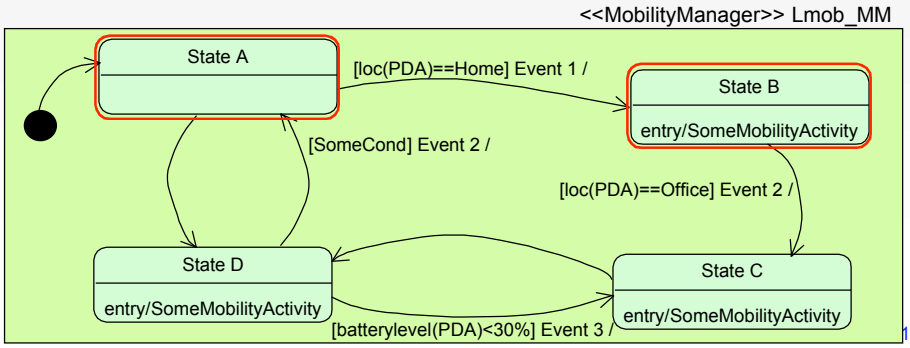
- stereotyped basic "mobile activities"
  - *MoveActivity*
    - *PhysicalMove*
    - *VirtualMove*
  - *BeforeMoveActivity*
  - *AfterMoveActivity*
  - *AllowDeploymentActivity*
  - *DenyDeploymentActivity*
  - *… (possibly others) ...*
- *MobilityManager*
  - a state machine that can dispatch mobile activities, built as a suitable composition of basic activities
  - it encapsulates all the “mobility logic”, keeping it separate from the application logic
    - different mobility managers can be modularly plugged into the same system model, to model different mobility scenarios and strategies
  - mainly a *modeling abstraction*, not a “real” entity

Adding dynamics to the model : physical mobility



Adding dynamics to the model : virtual mobility





11

UML-PMS : outline

Modeling **mobility** in UML-PMS
- Physical mobility: locations and moving locations
- Virtual mobility: mobile software components
- Dynamics of mobility: <<MobilityManager>>

Modeling the missing information about **performance**
- Annotated UML-PMS models: mobility models + SPT

Representing mobile code paradigms in UML-PMS
- Code On Demand (COD)
- Remote Evaluation (REV)
- Mobile Agent (MA)

Performance Analysis based on UML-PMS models: an example

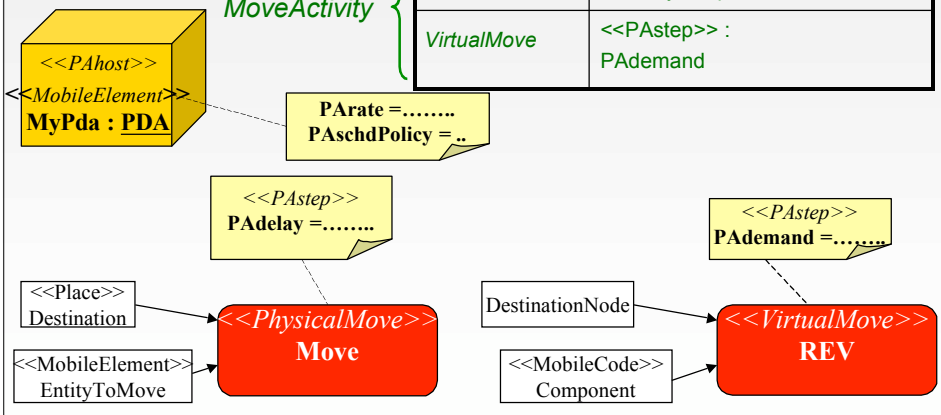
112

UML-PMS: modeling of performance characteristics

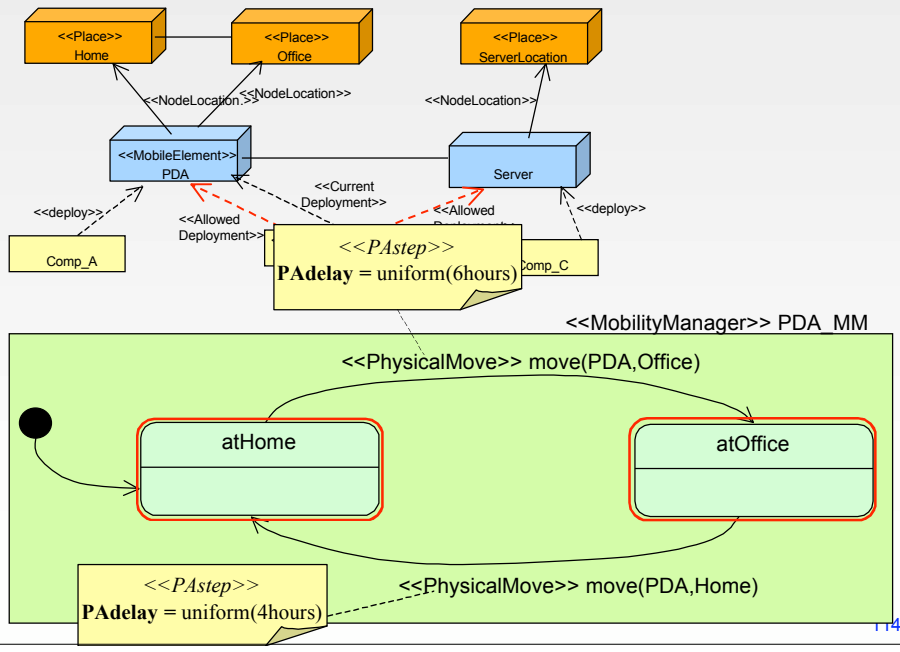
Adding to UML-PMS models performance annotations from the UML Profile for Schedulability, Performance and Time (SPT)

UML-PMS	UML-SPT
Node (possibly mobile)	<<PAhost>>, <<PAresource>> : PArate, PAschdPolicy, ...
PhysicalMove	<<PAstep>> : PAdelay, PAprob
VirtualMove	<<PAstep>> : PAdemand

MoveActivity



UML-PMS + UML-SPT: example of physical mobility



UML-PMS : outline

Modeling **mobility** in UML-PMS
- Physical mobility: locations and moving locations
- Virtual mobility: mobile software components
- Dynamics of mobility: <<MobilityManager>>

Modeling the missing information about **performance**
- Annotated UML-PMS models: mobility models + SPT

Representing mobile code paradigms in UML-PMS
- Code On Demand (COD)
- Remote Evaluation (REV)
- Mobile Agent (MA)

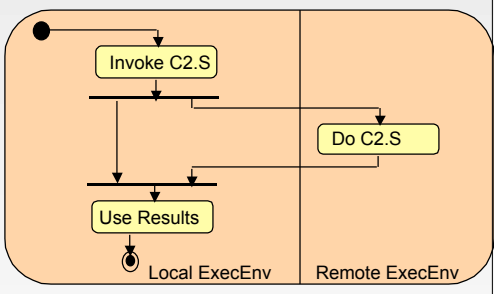
Performance Analysis based on UML-PMS models: an example

UML-PMS + UML-SPT: virtual mobility - modeling of mobile code paradigms

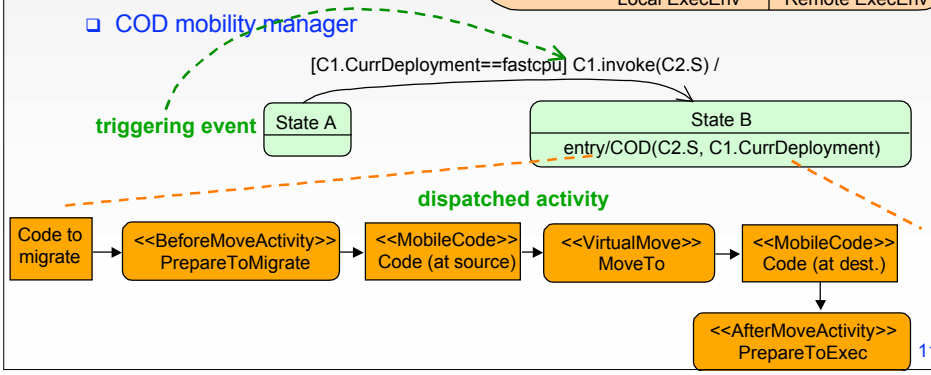
- code only mobility
  - *code on demand* (COD)
  - *remote evaluation* (REV)
    - both can be seen as “location aware” extensions of the “location unaware” client-server (CS) paradigm
- code+state mobility
  - *mobile agent* (MA)
- how to model them using UML-PMS + UML-SPT
  - definition of a suitable (fragment of) mobility manager
    - event triggering a state transition
    - dispatched activity
    - UML-SPT annotations for mobility “costs”

Plugging code mobility into an application: COD case (1)

- fragment of a "static" application
  - C1 invokes a service S implemented by C2

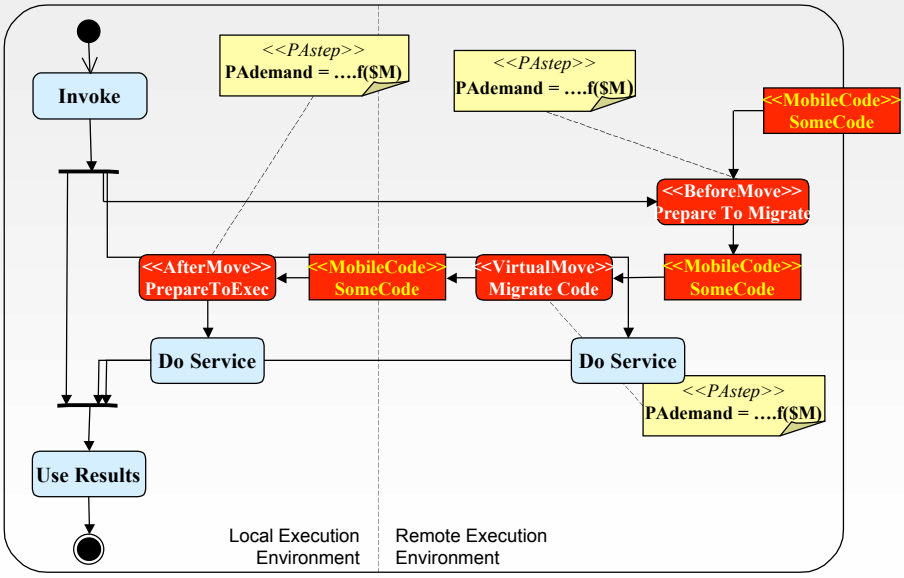


- COD mobility manager



117

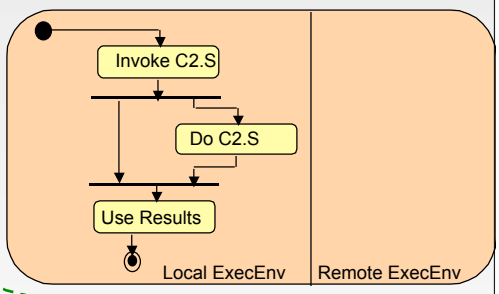
Plugging code mobility into an application: COD case (2)



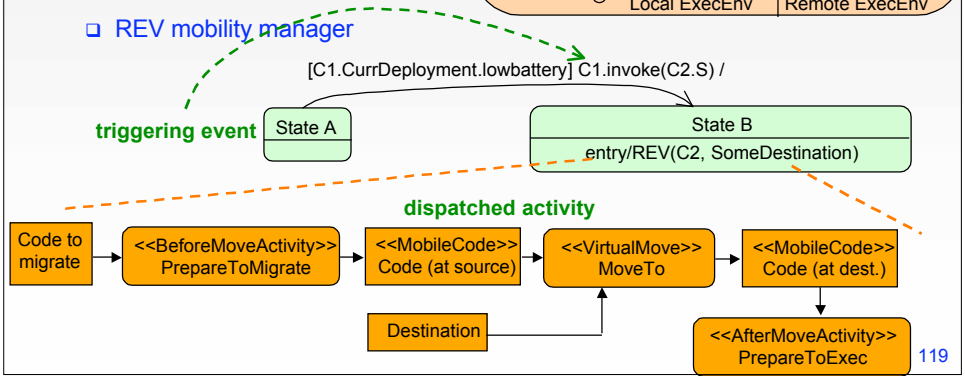
118

Plugging code mobility into an application: REV case (1)

- fragment of a "static" application
  - C1 invokes a service S implemented by C2

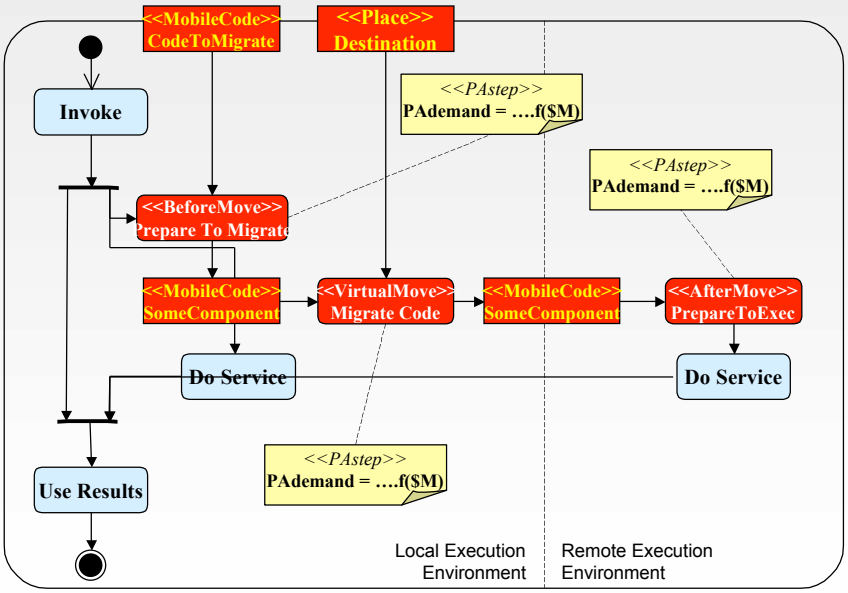


- REV mobility manager



119

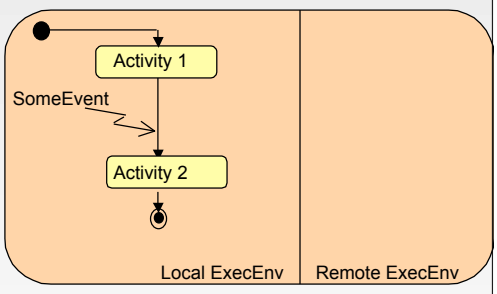
Plugging code mobility into an application: REV case (2)



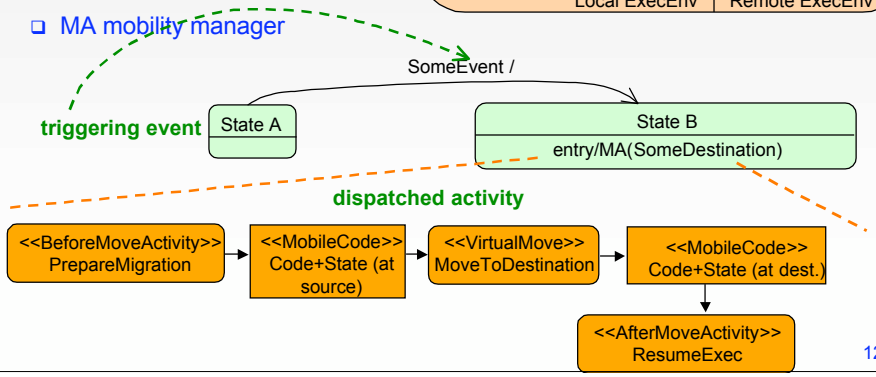
120

Plugging code mobility into an application: MA case (1)

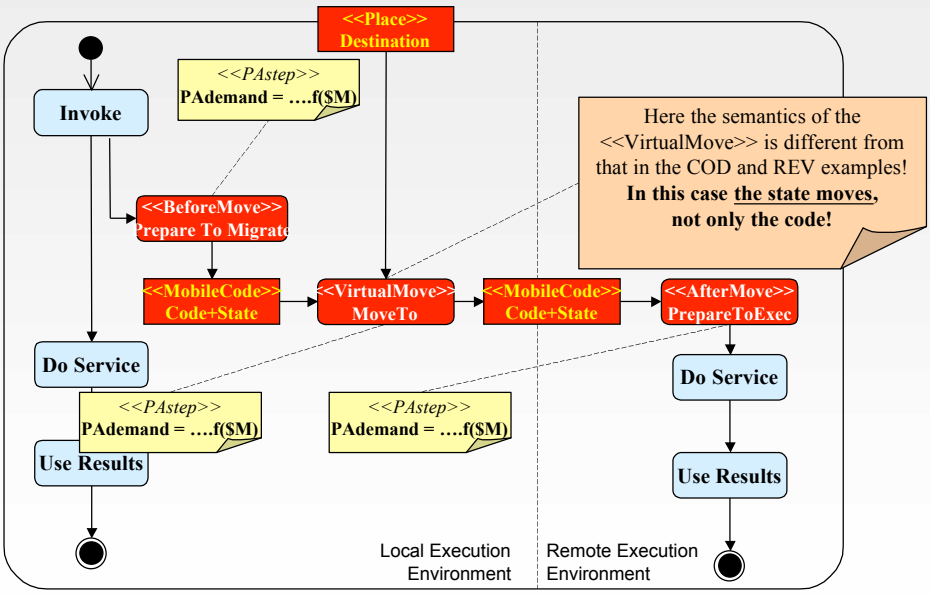
- fragment of a "static" application
  - C1 behavior



- MA mobility manager



Plugging code mobility into an application: MA case (2)



UML-PMS : outline

Modeling **mobility** in UML-PMS

- Physical mobility: locations and moving locations
- Virtual mobility: mobile software components
- Dynamics of mobility: <<MobilityManager>>

Modeling the missing information about **performance**

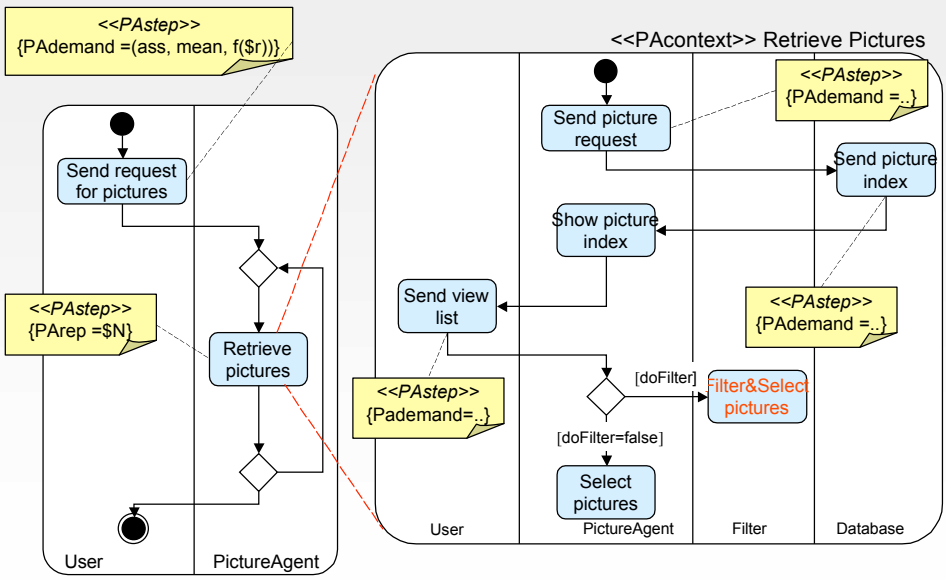
- Annotated UML-PMS models: mobility models + SPT

Representing mobile code paradigms in UML-PMS

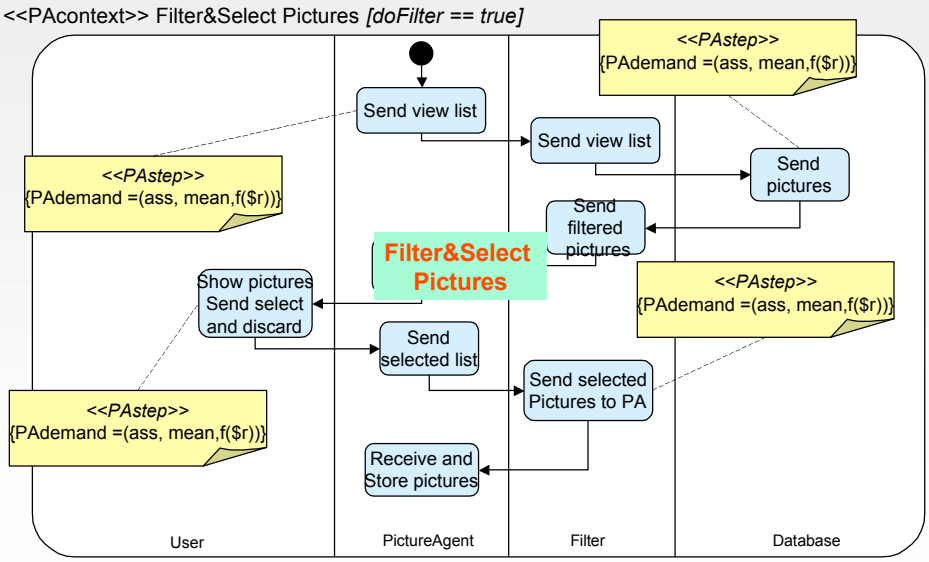
- Code On Demand (COD)
- Remote Evaluation (REV)
- Mobile Agent (MA)

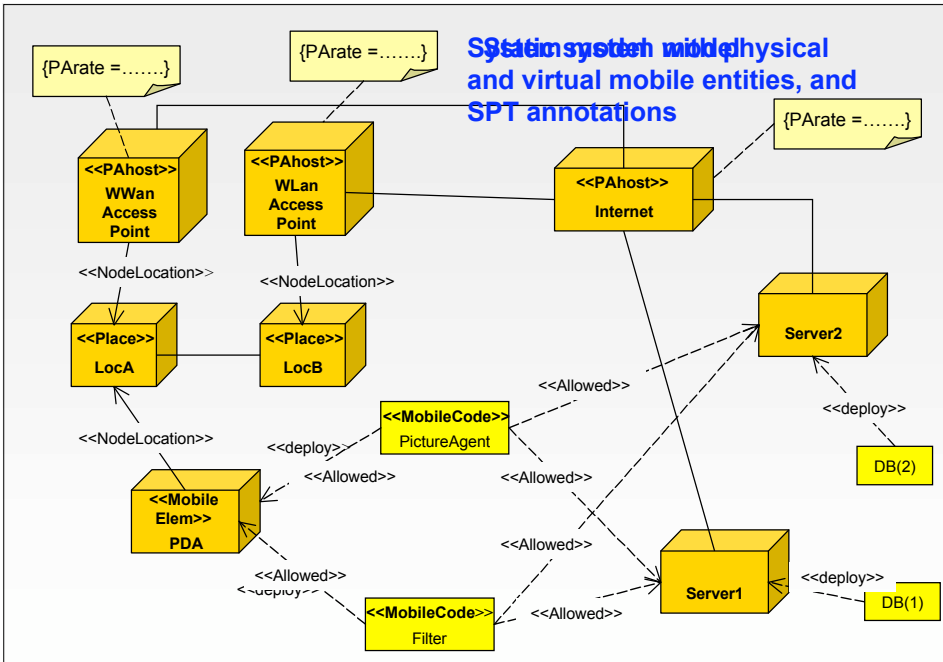
Performance Analysis based on UML-PMS models: an example

A picture retrieval system (1)

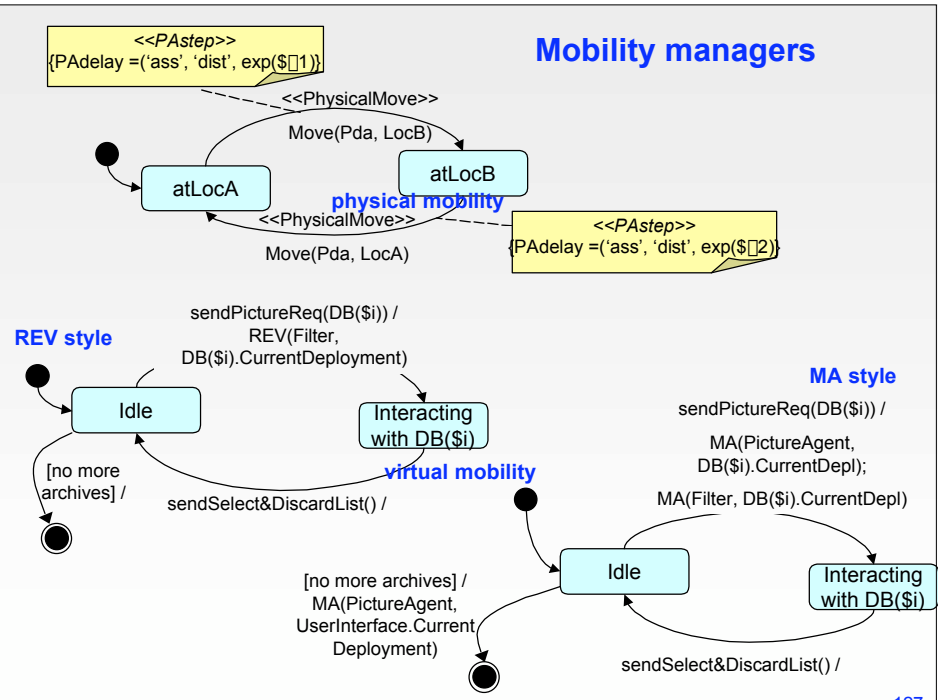


A picture retrieval system (2)





Mobility managers



Performance results: total network traffic (1)

average picture size (high quality): H
average picture size (low quality): h
average size of the pictures textual description: t
PictureAgent size (excluded the stored pictures): SPA
Filter size: SF
average size of request messages: r
average number of contacted picture archive sites: N (≥ 1)
average number of initially selected pictures per site: $Sinit$
average number of finally selected pictures per site: $p \cdot Sinit$ ($0<p\leq 1$)
discard percentage of previously loaded pictures, after a visit to a new site: $c \cdot 100\%$ ($(0<c\leq 1)$)

no virtual mobility : $T = 2 \cdot N \cdot [S_{init} * H + t + 2r]$

REV style : $TREV = 2 \cdot N \cdot [t + 3r + S_F + S_{init} * h + p * S_{init} * H]$

MA style : $TMA = 2 * N \cdot [t + 2r + S_{init} * h] + 2 \cdot (S_F + S_{PA}) + p * \cdot_{N-1}(c) + (N-1) \cdot [S_F + S_{PA}] + p * \sum_{i=1}^{N-1} \cdot_i(c)$

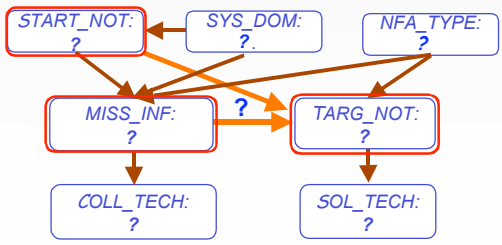
Performance results: total network traffic (2)

average picture size (high quality): H
average picture size (low quality): h
average size of the pictures textual description: t
PictureAgent size (excluded the stored pictures): SPA
Filter size: SF
average size of request messages: r
average number of contacted picture archive sites: N (≥ 1)
average number of initially selected pictures per site: $Sinit$
average number of finally selected pictures per site: p $Sinit$ ($0<p\leq 1$)
discard percentage of previously loaded pictures, after a visit to a new site: c 100% ($(0<c\leq 1)$)

$p=1$, $c=1$:	TMA > TREV > T
$p=1$, $c=0$:	TREV > T > TMA
$p=1$, $0<c<1$:	TREV > T (TMA ?)
$p<1$, $c=1$:	TMA > T > TREV
$p<1$, $c=0$:	T > TREV > TMA
$p<1$, $0<c<1$:	T > TREV (TMA ?)

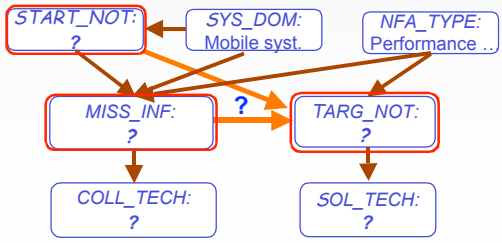
Conclusions

- A general goal: from NFR validation as an art to NFR validation as a routine process
  - systematic view of NFR validation, by identifying some crucial issues and their relationships
  - focus on the "translation" process from START_NOT (+ MISS_INF) to TARG_NOT



Conclusions

- A general goal: from NFR validation as an art to NFR validation as a routine process
  - systematic view of NFR validation, by identifying some crucial issues and their relationships
  - focus on the "translation" process from START_NOT (+ MISS_INF) to TARG_NOT
  - considered scenario : performance validation of mobile systems



Conclusions

- START_NOTs for mobile systems modeling
  - semi-formal notations: UML (with, possibly, "mobility-oriented" extensions)
  - formal notations: "mobility-oriented" Process Algebras
- UML
  - pros: graphical notation, extensibility, *de-facto* standard in industrial software production, "design-oriented features"
  - cons: imprecise semantics
  - translation to TARG_NOT requires further investigation
- "mobility-oriented" Process Algebras
  - pros: unambiguous, compositional features, well defined translation process to TARG_NOT (MRP)
  - cons: state space explosion, high number of parameters, non widely accepted formalism, lack of "design-oriented features"

Conclusions

UML

- expressive power
  - limited support to mobility modeling offered by the “original” UML
  - UML customization through its *extension* mechanisms
    - lightweight extensions (“**profiles**”)
    - heavyweight extensions
- usability
  - de facto standard notation for software design
  - promotes separation of concerns through different system views (“diagrams“)
  - translation methodology to some TARG_NOT
    - semantic problems
    - no “universal” methodology

Process Algebras

- expressive power
  - lack of explicit “primitives” to model mobile code style (COD, REV, MA, …)
  - two ways of modeling locations (and mobility)
    - explicitly
    - implicitly
      - » could be somewhat less intuitive
- usability
  - lack of a clear separation of concerns between application logic and mobility
    - at least in "basic” process algebras for mobility
  - integration with existing design practices is still an issue
  - translation methodology to some TARG_NOT
    - “natural” extension of their syntax and semantics
      - » MISS_INF : expressed through *stochastic process algebras*
      - » TARG_NOT = Markov process

Conclusions (no more … :-))

- Bridging the gap between formal and semi-formal notations?
  - mapping of information extracted from (semi-formal) design artifacts to parameters of formal models
  - using formal models to give a sound semantics to UML-based models
  - …