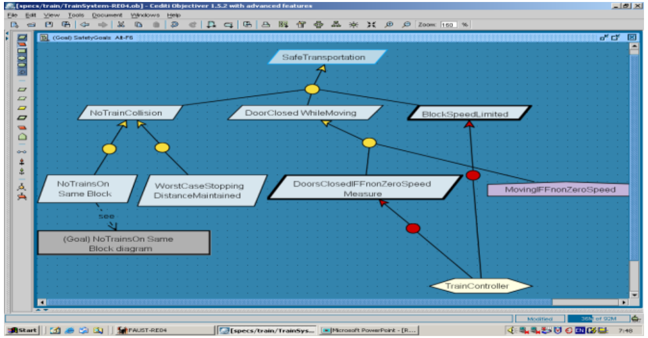
# Modeling System Operations

Mariano Ceccato

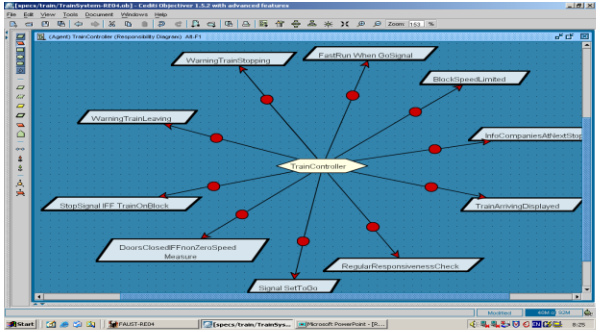
mariano.ceccato@univr.it

# Building models for RE

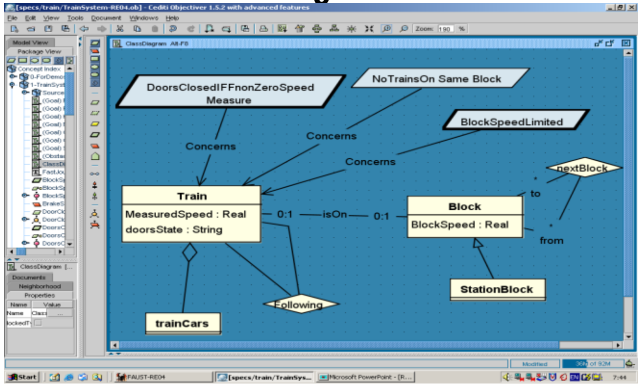
## Goals


[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
(Goal) SafetyGoals Alt-F6
SafeTransportation
NoTrainCollision
DoorClosed WhileMoving
BlockSpeedLimited
NoTrainsOn Same Block
WorstCaseStopping DistanceMaintained
DoorsClosedIFFnonZeroSpeed Measure
MovingIFFnonZeroSpeed
see
(Goal) NoTrainsOn Same Block diagram
TrainController
Start
FAUST-RE04
[specs/train/TrainSys...
Microsoft PowerPoint - [R...

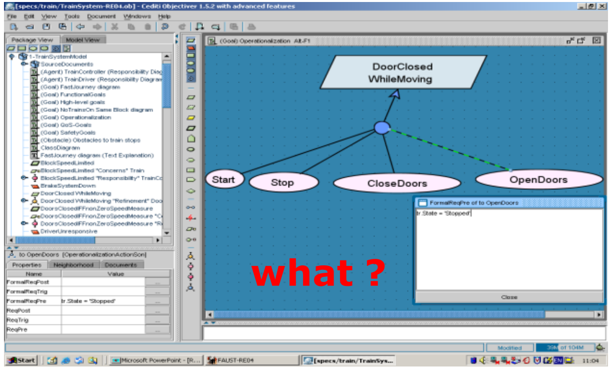
## Agents & responsibilities


[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
(Agent) TrainController (Responsibility Diagram) Alt-F1
WarningTrainStopping
FastRun When GoSignal
BlockSpeedLimited
WarningTrainLeaving
InfoCompaniesAtNextStop
TrainController
StopSignal IFF TrainOnBlock
TrainArrivingDisplayed
DoorsClosedIFFnonZeroSpeed Measure
RegularResponsivenessCheck
Signal SetToGo
Start
FAUST-RE04
[specs/train/TrainSys...
Microsoft PowerPoint - [R...

## Objects



## Operations


[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
Package View
Model View
(Goal) Operationalization Alt-F1
DoorClosed WhileMoving
Start
Stop
CloseDoors
OpenDoors
FormalReqPre of to OpenDoors
tr.State = 'Stopped'
Close
Start
Microsoft PowerPoint - [R...
FAUST-RE04
[specs/train/TrainSys...

1-TrainSystemModel
- SourceDocuments
- (Agent) TrainController (Responsibility Diag
- (Agent) TrainDriver (Responsibility Diagram
- (Goal) FastJourney diagram
- (Goal) FunctionalGoals
- (Goal) High-level goals
- (Goal) NoTrainsOn Same Block diagram
- (Goal) Operationalization
- (Goal) QoS-Goals
- (Goal) SafetyGoals
- (Obstacle) Obstacles to train stops
- ClassDiagram
- FastJourney diagram (Text Explanation)
- BlockSpeedLimited
- BlockSpeedLimited "Concerns" Train
- BlockSpeedLimited "Responsibility" TrainCo
- BrakeSystemDown
- DoorClosed WhileMoving
- DoorClosed WhileMoving "Refinement" Doo
- DoorsClosedIFFnonZeroSpeedMeasure
- DoorsClosedIFFnonZeroSpeedMeasure "C
- DoorsClosedIFFnonZeroSpeedMeasure "R
- DriverUnresponsive

to OpenDoors [OperationalizationActionSon]

Properties | Neighborhood | Documents

| Name | Value |
|---|---|
| FormalReqPost | |
| FormalReqTrig | |
| FormalReqPre | tr.State = 'Stopped' |
| ReqPost | |
| ReqTrig | |
| ReqPre | |

what ?

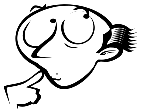
# The operation model

- Functional view of the system being modeled
  - **what services** are to be provided? (statics)
  - **under what conditions** for goal satisfaction?
- Represented by operationalization diagram, UML use cases
- Multiple uses:
  - software specifications --input for development team
  - description of environment tasks & procedures
  - basis for deriving:
    - black-box test data
    - executable specs for animation, prototyping
  - definition of function points (for size estimation), work units, user manual sections
  - satisfaction arguments, traceability management

# Modeling system operations:  outline

- What are operations?
- Characterizing system operations
  - Operation signature
  - Domain pre- and postconditions
  - Operation performer
- Goal operationalization
  - Required pre-, post-, trigger conditions for goal satisfaction
  - Agent commitments
  - Goal operationalization and satisfaction arguments
- Goals, agents, objects & operations: the semantic picture
- Representing operation models
  - Operationalization diagrams
  - UML use case diagrams
- Building operation models:  heuristics & derivation rules

# What are operations?

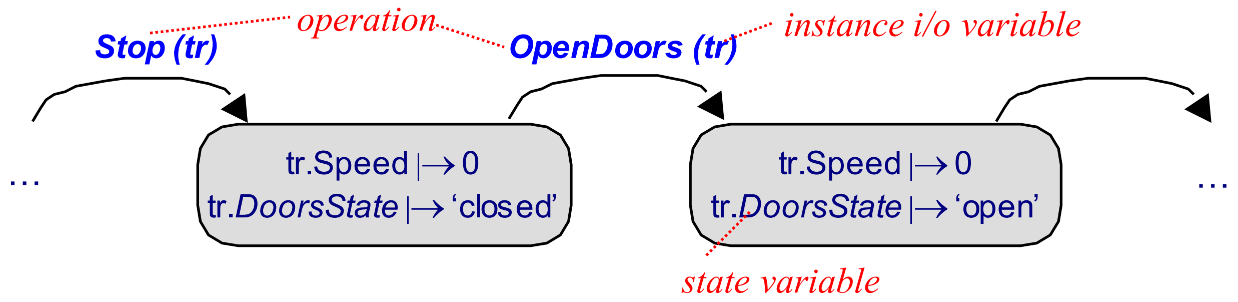


- **Operation** Op = set of input-output state pairs (binary relation)

  $$Op \subseteq \text{InputState} \times \text{OutputState}$$

  - <u>state</u> = tuple of functional pairs $\mathbf{x}_i \mapsto \mathbf{v}_i$ (cf. conceptual object lecture)

    $\mathbf{x}_i$ : variable, $\mathbf{v}_i$ : corresponding value
  - <u>input variables</u>: object instances to which *Op* applies
  - <u>output variables</u>: object instances upon which *Op* acts
  - <u>attributes</u> of i/o variables instantiated as state variables
- Operation **applications** yield state transitions (events)



***Stop (tr)***

*operation*

***OpenDoors (tr)***

*instance i/o variable*

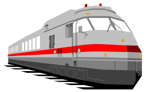
…

tr.Speed $\mapsto$ 0
tr.*DoorsState* $\mapsto$ ‘closed’

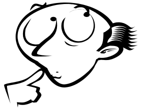
tr.Speed $\mapsto$ 0
tr.*DoorsState* $\mapsto$ ‘open’

…

*state variable*



# What are operations?



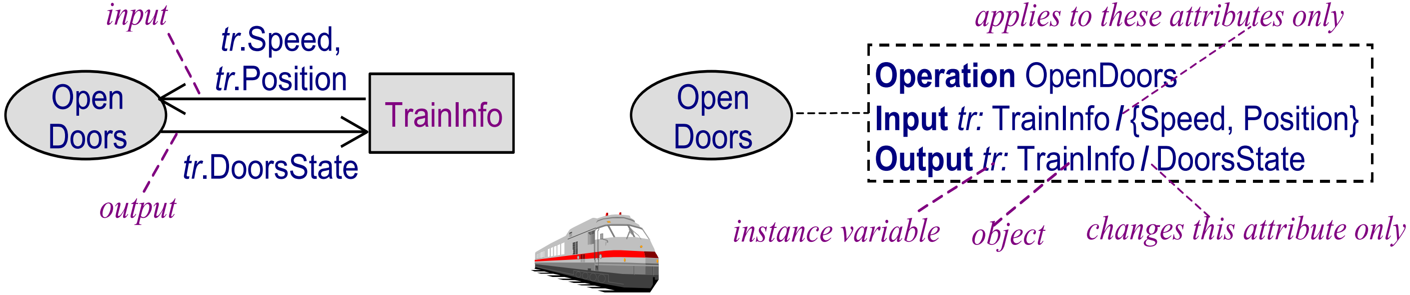
- Op must **operationalize** underlying goals from goal model
  - to make these satisfied => application under restricted conditions
- Generally deterministic:   relation over states is a function
  - no multiple alternative outputs from same input
- **Atomic**:  map input state to state at next smallest time unit
  - not decomposable into finer-grained operations

    ☞ decompose underlying goals, not operations !   (semantically simpler)
  - for operations lasting some duration:  use **startOp/endOp** events
- May be applied concurrently with others
  - intra-agent concurrency  (beside inter-agent concurrency)
  - e.g.   OpenDoors || DisplayWhichPlatform

- Software operations, environment operations (tasks)
  - e.g.   PlanMeeting ,  SendConstraints

# Characterizing system operations

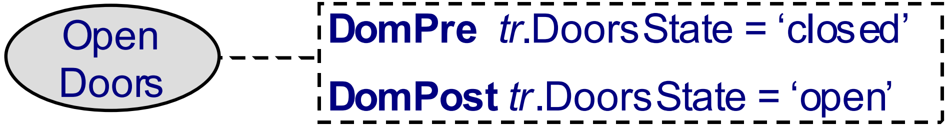
- Basic features: Name, Def, Category
- Signature
    - declares the input-output relation over states
        - input/output variables & their type (object from object model)
        - scope may be restricted to specific attributes (nothing else changes)
        - used in pre-, post-conditions
    - graphical or textual annotation



**Operation** OpenDoors
**Input** *tr*: TrainInfo / {Speed, Position}
**Output** *tr*: TrainInfo / DoorsState

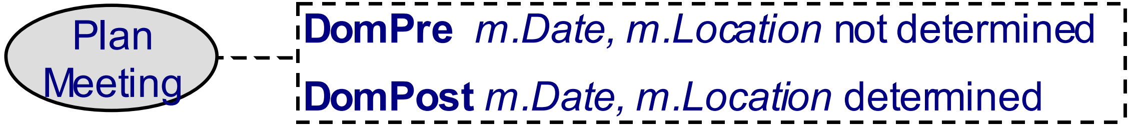
# Characterizing system operations: domain pre- and post-conditions

- Conditions capturing the class of state transitions that intrinsically defines the operation
- **DomPre**: condition characterizing class of input states in domain
  - <u>descriptive</u>, not prescriptive, for some goal
- **DomPost**: condition characterizing class of output states in domain
  - <u>descriptive</u>, not prescriptive, for some goal



**DomPre** *tr*.DoorsState = 'closed'

**DomPost** *tr*.DoorsState = 'open'



**DomPre** *m.Date, m.Location* not determined

**DomPost** *m.Date, m.Location* determined

# Characterizing system operations: operation performer

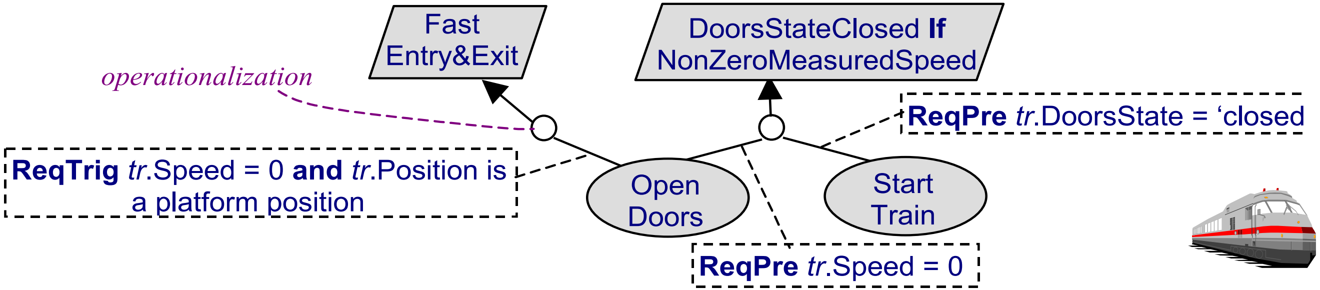
- An agent **performs** an operation if the applications of this operation are activated by instances of this agent (cf. agent responsibility lecture)
- Consistency rules between *operation* model & *agent* model:
  - every *input/output* state variable in signature of operation performed by an agent must be *monitored/controlled* by it in the agent model
  - every operation is performed by exactly one agent
    - cf. *Unique Controller* constraint in agent model

# Modeling system operations: outline

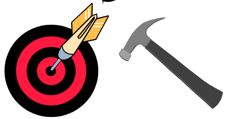
- What are operations?
- Characterizing system operations
  - Operation signature
  - Domain pre- and postconditions
  - Operation performer
- Goal operationalization
  - Required pre-, post-, trigger conditions for goal satisfaction
  - Agent commitments
  - Goal operationalization and satisfaction arguments
- Goals, agents, objects & operations: the semantic picture
- Representing operation models
  - Operationalization diagrams
  - UML use case diagrams
- Building operation models: heuristics & derivation rules

# Goal operationalization

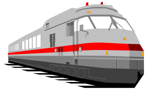
- A set of operations **operationalizes** a leaf goal if their spec ensures that the goal is satisfied
- These specs are **prescriptive** conditions on the operations:
  - **ReqPre**: *necessary* condition on Op's input states to ensure G:
    - when DomPre *true*, Op **may** be applied **only if** ReqPre *true* (permission)
  - **ReqTrig**: *sufficient* condition on Op's input states to ensure G:
    - when DomPre *true*, Op **must** be applied **as soon as** ReqTrig *true* (obligation)
  - **ReqPost**: condition on Op's output states to ensure G


Fast Entry&Exit
DoorsStateClosed **If** NonZeroMeasuredSpeed
*operationalization*
**ReqPre** *tr*.DoorsState = 'closed
**ReqTrig** *tr*.Speed = 0 **and** *tr*.Position is a platform position
Open Doors
Start Train
**ReqPre** *tr*.Speed = 0

# Specifying operations textually: example

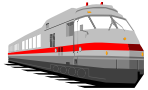


- **Operation** OpenDoors
  - **Def** *Operation controlling the opening of all train doors*
  - **Input** *tr*: Train / {Speed, Position}
  - **Output** *tr*: Train / DoorsState
  - **DomPre** The doors of train *tr* are closed
  - **DomPost** The doors of train *tr* are open
  - **ReqPre** For DoorsClosedWhileNonzeroSpeed
    - The speed of train *tr* is 0
  - **ReqPre** For SafeEntry&Exit
    - Train *tr* is at a platform
  - **ReqTrig** For FastEntry&Exit
    - Train *tr* has just stopped at a platform



# Specifying operations textually: another example

- **Operation** SendAccelerationCommand
  - **Def** *Operation of sending an acceleration command to a train*
  - **Input** *tr*: Train, cm: CommandMsg;
  - **Output** cm: Sent     % association instance %
  - **DomPre** not Sent (cm, tr)
  - **DomPost** Sent (cm, tr)
  - **ReqPost** For SafeAccelerationCommand
    - The commanded acceleration sent to *tr* is within safe range with respect to the preceding train’s position and speed
  - **ReqTrig** For CommandMsgSentInTime
    - No acceleration command has been sent to *tr* since 3 seconds



# Goal operationalization

- A leaf goal is generally operationalized by <u>multiple operations</u>
- An operation generally operationalizes <u>multiple goals</u>
  - all ReqPre/ReqPost are implicitly conjoined with DomPre/DomPost
  - if DomPre *true*, **must** be applied as soon as **one** ReqTrig *true*
    (not applied if one or more ReqTrig *true* with DomPre *false*)
  - if DomPre *true*, **may** be applied provided **all** ReqPre *true*
    (not applicable if all ReqPre *true* with DomPre *false*)

- <u>Consistency</u> constraint on obligations & permissions:

  **if** DomPre and (ReqTrig$_1$ *or* … *or* ReqTrig$_M$) *obligation*
  **then** (ReqPre$_1$ *and* … *and* ReqPre$_N$) *permission*

# Agent commitments

- For every goal G under responsibility of agent ag,
  for every operation Op operationalizing G,
  ag must guarantee that Op is applied:
  **when** Op's DomPre holds,
  **as soon as** one of Op's ReqTrig holds
  **only if** all Op's ReqPre hold,
  **so as to** establish Op's DomPost together with all Op's ReqPost

- Extra consistency rules between *operation* and *agent* models:
  - *ag* responsible for *G* must perform all operations operationalizing *G*
  - if these operations operationalize other goals, *ag* must be responsible for these goals too

# Agent commitments

- Agent non-determinism: eager vs. lazy behavior on ReqPre’s
  - **eager**: agent instance applies operation as soon as all ReqPre *true* (maximal progress)
  - **lazy**: agent instance applies operation only when obliged, due to one ReqTrig *true*

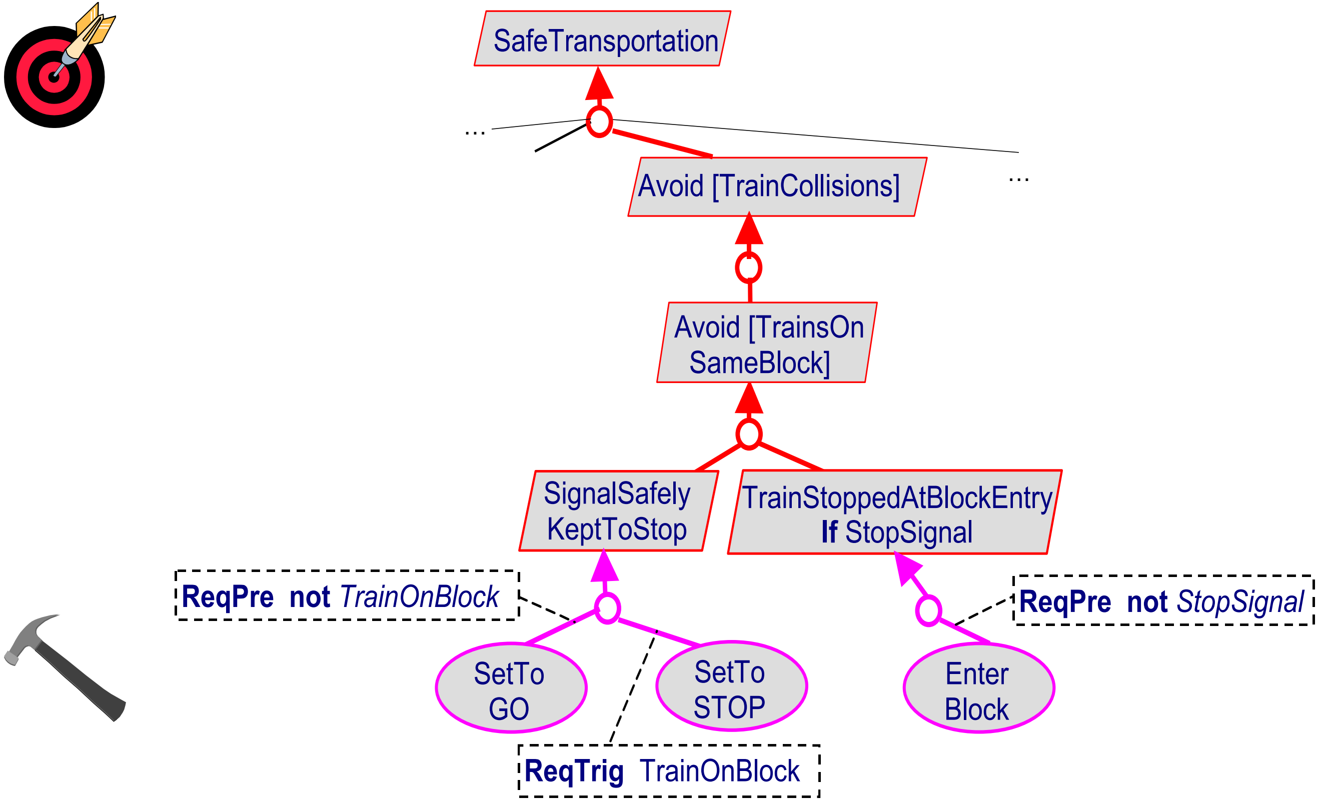
- Agent concurrency:
  - ReqTrig’s on multiple operations *true* in same state
  - true parallelism, intra-agent or inter-agent

# Goal operationalization and satisfaction arguments

- The *goal* and *operation* models may be used to argue that operational requirements ensure higher-level objectives
  - bottom-up chaining of *operationalization* & *refinement* links
    - {Spec($\text{Op}_1$), …, Spec($\text{Op}_M$)} |= OperationalizedGoal
    - {$\text{Subgoal}_1$ …, $\text{Subgoal}_N$ , DOM} |= ParentGoal (cf. goal diagram lecture)

- Yield derivational traceability links for free
  - **backwards**: why this operational spec, for what goals?
  - **forwards**: how is this goal ensured?

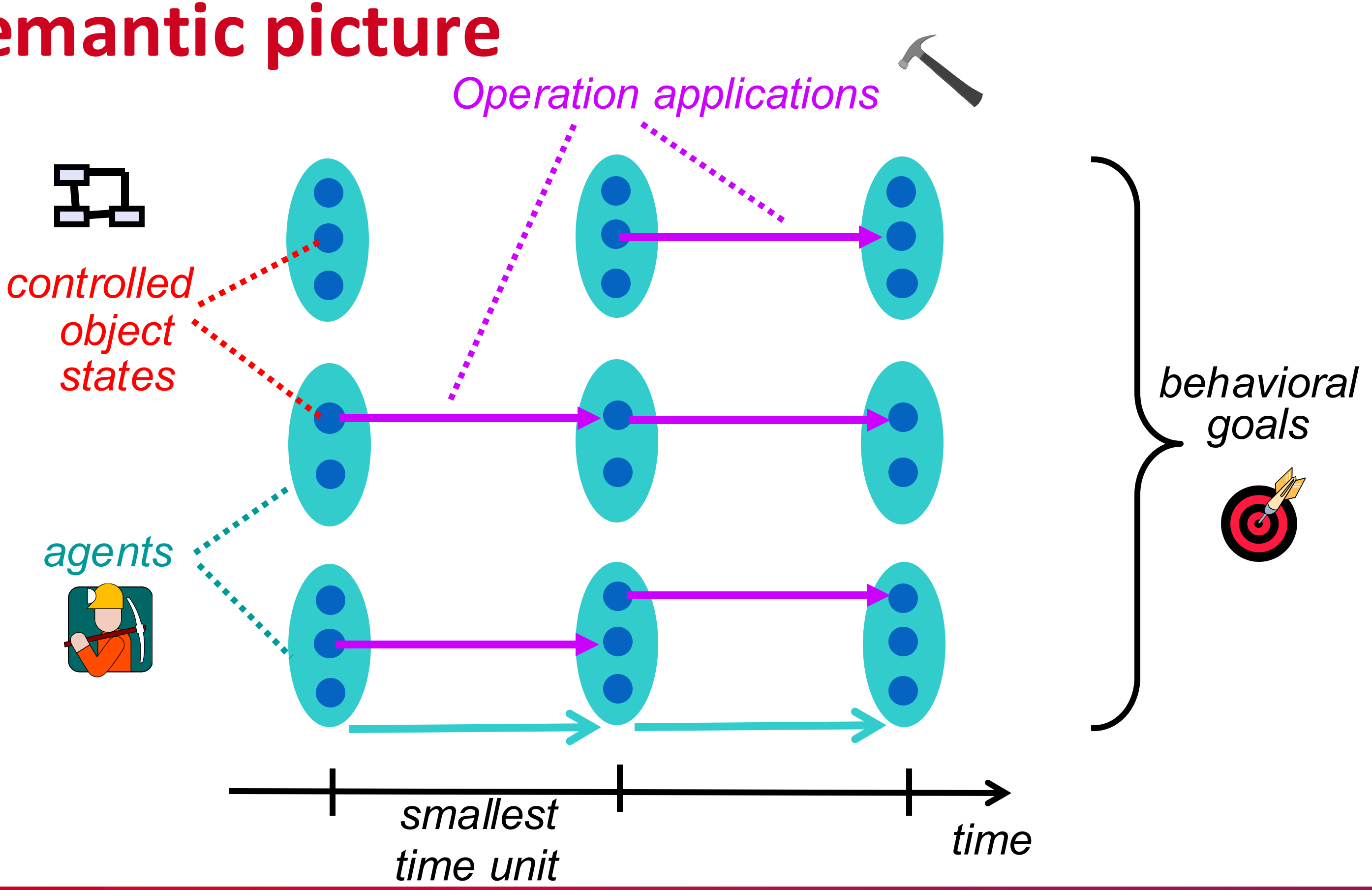
# Satisfaction arguments & derivational traceability: example


SafeTransportation
…
Avoid [TrainCollisions]
…
Avoid [TrainsOn SameBlock]
SignalSafely KeptToStop
TrainStoppedAtBlockEntry **If** StopSignal
**ReqPre not** *TrainOnBlock*
**ReqPre not** *StopSignal*
SetTo GO
SetTo STOP
Enter Block
**ReqTrig** TrainOnBlock

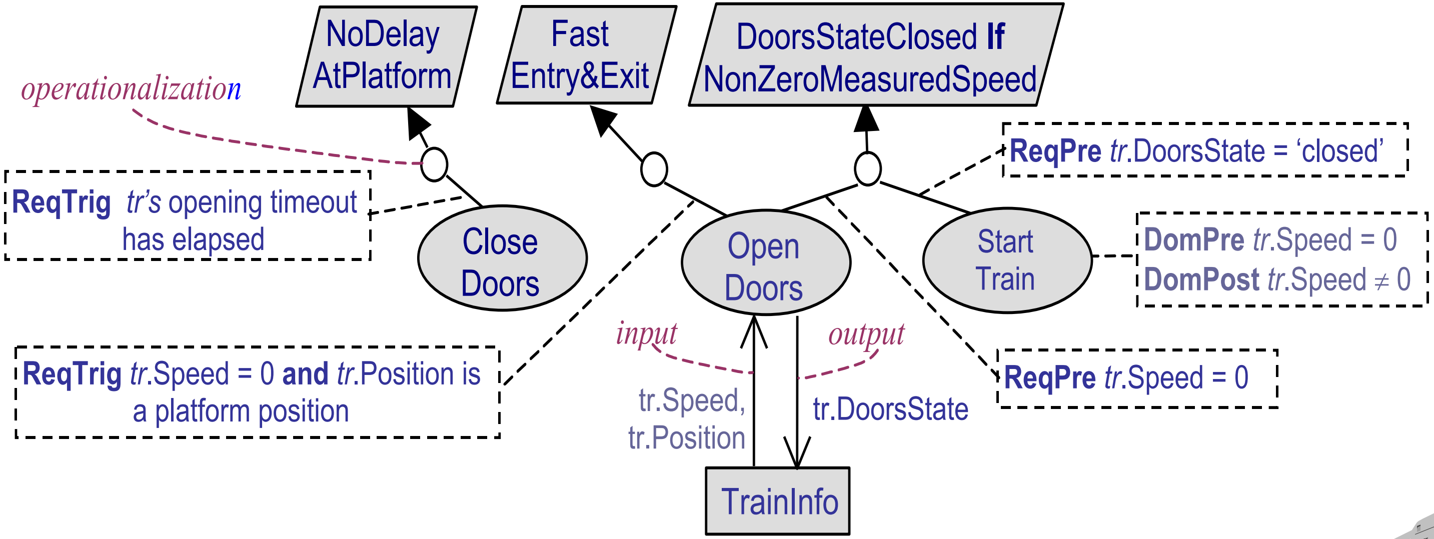
# Modeling system operations:  outline

- What are operations?
- Characterizing system operations
  - Operation signature
  - Domain pre- and postconditions
  - Operation performer
- Goal operationalization
  - Required pre-, post-, trigger conditions for goal satisfaction
  - Agent commitments
  - Goal operationalization and satisfaction arguments
- Goals, agents, objects & operations: the semantic picture
- Representing operation models
  - Operationalization diagrams
  - UML use case diagrams
- Building operation models:  heuristics & derivation rules

# Goals, objects, agents, operations: the semantic picture



# Representing operation models: operationalization diagrams


operationalization
NoDelay AtPlatform
Fast Entry&Exit
DoorsStateClosed **If** NonZeroMeasuredSpeed
**ReqPre** *tr*.DoorsState = ‘closed’
**ReqTrig** *tr’s* opening timeout has elapsed
Close Doors
Open Doors
Start Train
**DomPre** *tr*.Speed = 0
**DomPost** *tr*.Speed ≠ 0
input
output
**ReqTrig** *tr*.Speed = 0 **and** *tr*.Position is a platform position
tr.Speed, tr.Position
tr.DoorsState
**ReqPre** *tr*.Speed = 0
TrainInfo

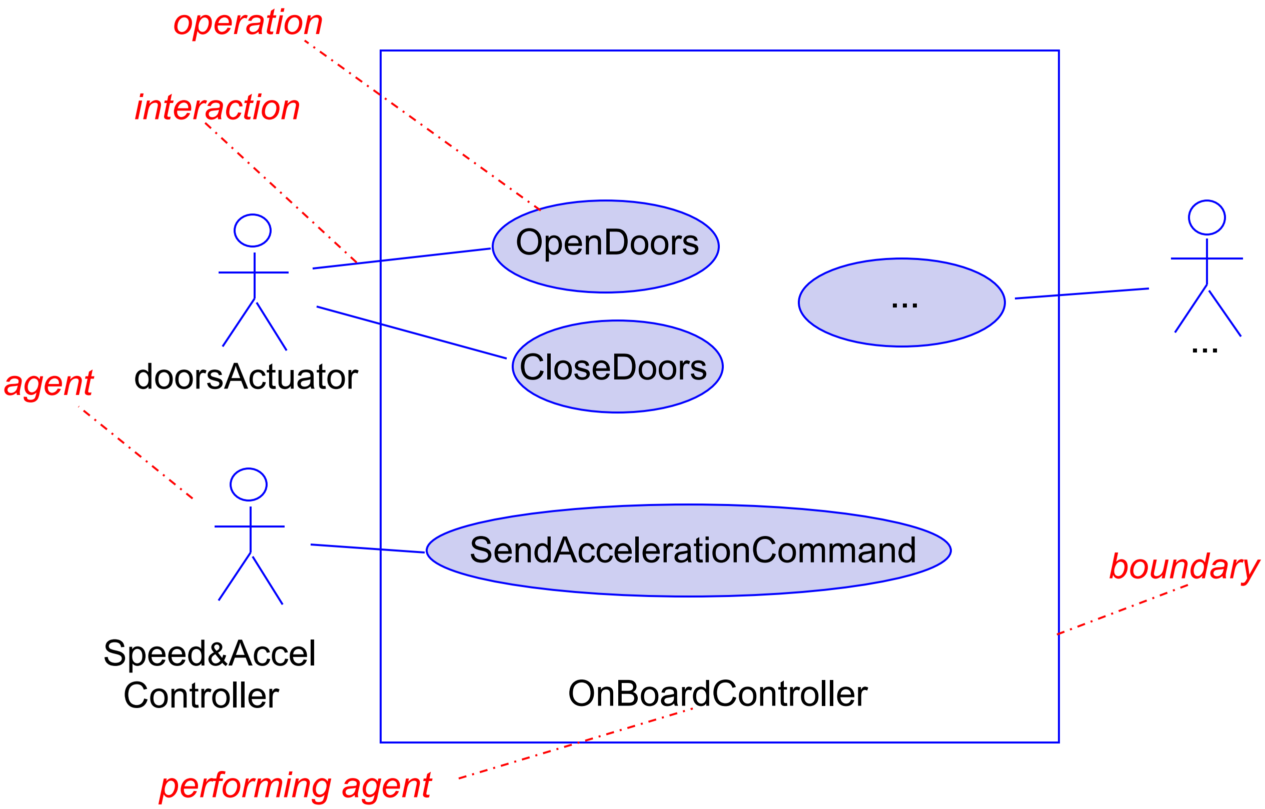
# Representing operation models: UML use case diagrams

- A **use case** outlines the operations an agent has to perform

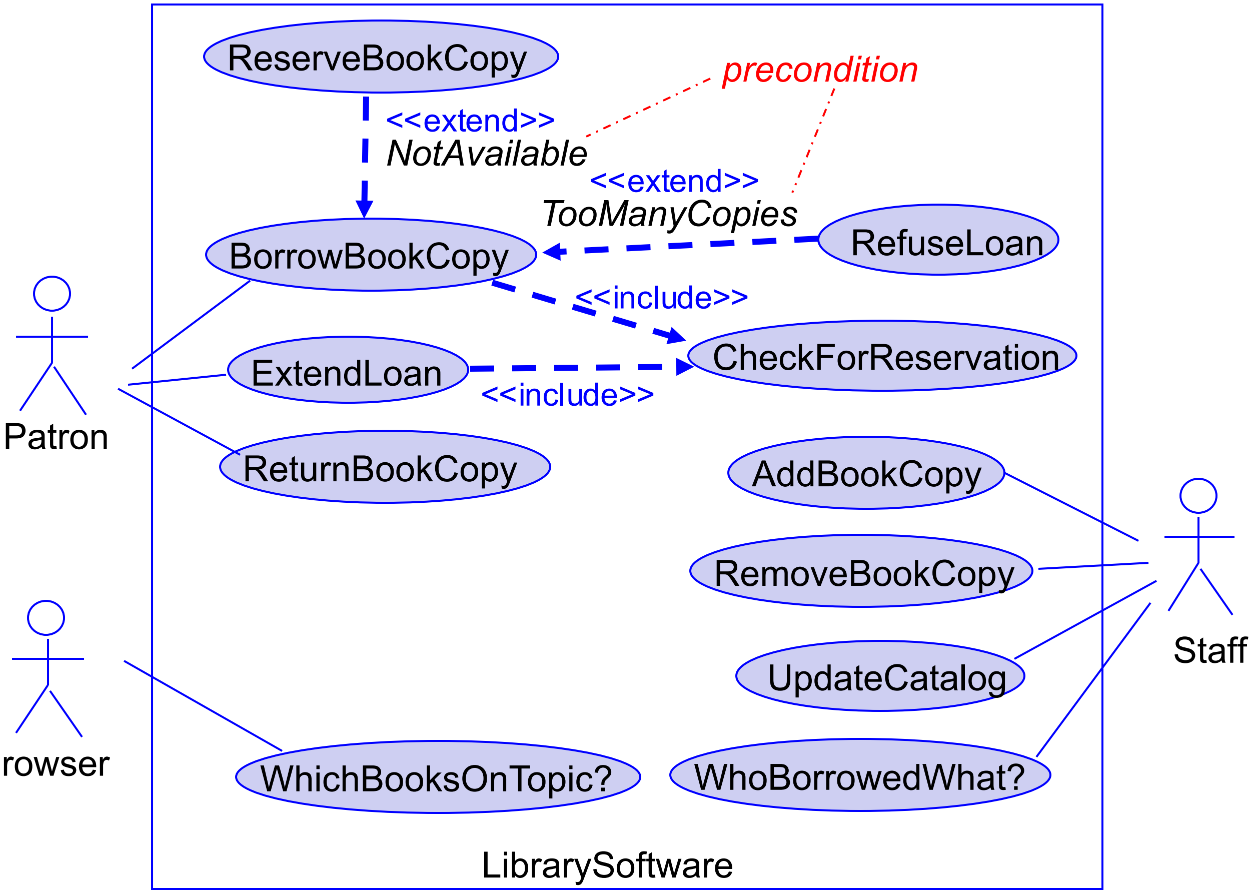
  +: interactions with:
  - the agents controlling operation inputs
  - the agents monitoring operation outputs

  +: optional (ill-defined) links:
  - to exception operations with preconditions ("**extend**")
  - to sub-operations ("**include**")
- A use case should operationalize the leaf goals underlying the operations in it
- <u>Decompose goals, NOT operations</u>!! (=> precise semantics)
- Generation of use cases from the operation & agent models is straightforward (see hereafter)

# UML use case diagrams:  example



# UML use case diagrams:  another example



# Modeling system operations:  outline

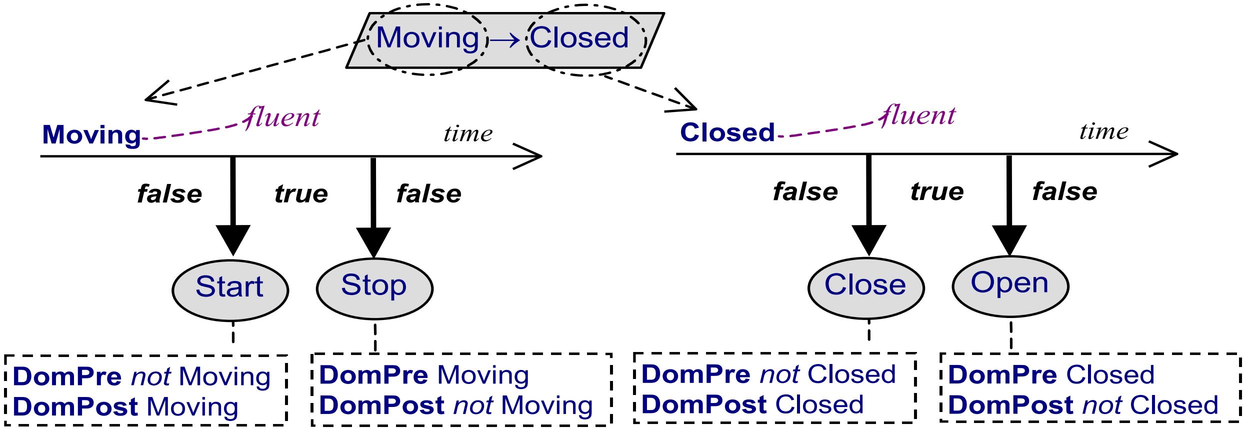
- What are operations?
- Characterizing system operations
  - Operation signature
  - Domain pre- and postconditions
  - Operation performer
- Goal operationalization
  - Required pre-, post-, trigger conditions for goal satisfaction
  - Agent commitments
  - Goal operationalization and satisfaction arguments
- Goals, agents, objects & operations: the semantic picture
- Representing operation models
  - Operationalization diagrams
  - UML use case diagrams
- Building operation models:  heuristics & derivation rules

# Derive operations from goal fluents

- Conditions defined by initiating and termination operations are called **fluents**
- For each behavioral leaf goal:  list atomic conditions *F* in its specification
- For each *F*,  look for:
  - **initiating operation**:  makes *F* **true** when *F* was **false**

    => DomPre =  not *F*,   DomPost =  *F*
  - **terminating operation**:  makes *F* false when *F* was true

    => DomPre =  *F*,   DomPost = not *F*



**DomPre** *not* Moving
**DomPost** Moving

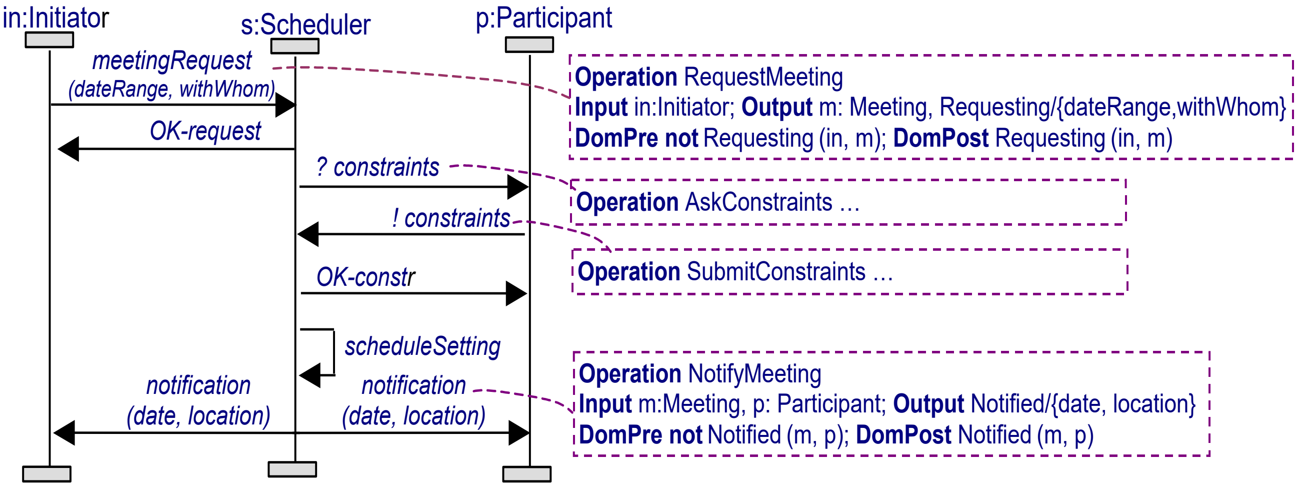
**DomPre** Moving
**DomPost** *not* Moving

**DomPre** *not* Closed
**DomPost** Closed

**DomPre** Closed
**DomPost** *not* Closed

# Identify operations from interaction scenarios

- For each interaction event in a scenario:
  - is this an operation application by the source agent with output monitored by the target agent?
  - what is the atomic condition characterizing the interaction on the source agent timeline?
    - right before interaction => DomPre
    - right after interaction => DomPost



**Operation** RequestMeeting
**Input** in:Initiator; **Output** m: Meeting, Requesting/{dateRange,withWhom}
**DomPre not** Requesting (in, m); **DomPost** Requesting (in, m)

**Operation** AskConstraints …

**Operation** SubmitConstraints …

**Operation** NotifyMeeting
**Input** m:Meeting, p: Participant; **Output** Notified/{date, location}
**DomPre not** Notified (m, p); **DomPost** Notified (m, p)

# Strengthen DomPre, DomPost with conditions required by goals

- **Identify required permissions**: if an operation's DomPost effect can violate a goal *G* under condition *C*

  => **ReqPre** for G: not C

  e.g. OpenDoors: ReqPre for "Moving → Closed": not Moving

- **Identify required obligations**: if an operation's DomPost effect is prescribed by a goal *G* to hold whenever condition *C* gets *true*

  => **ReqTrig** for G: C

  e.g. OpenDoors: ReqTrig for "StoppedAtPlatform → Open": StoppedAtPlatform

- **Identify required additional effects**: if an operation's DomPost is not sufficient to ensure the target condition *T* of goal *G* ...

  => **ReqPost** for G: missing subcondition from T

  e.g. PlanMeeting: ReqPost for ConvenientMeeting: date not in excluded dates

# Generating use case diagrams from operationalization diagrams

```
For each agent ag in agent diagram:
  enclose all operations performed by ag in a rectangle labelled ag;
  for each such operation op in corresponding operationalization diagram:
      for each other agent ag-env in the agent diagram:
          if ag-env controls one of op's input object attribute/association
                                 or monitors one of op's output object attribute/association
          then  include ag-env around ag's rectangle
                             and draw an interaction link between op and ag-env
        transfer op's DomPre, DomPost, ReqPres, ReqTrigs, ReqPosts
```