Engineering Requirements for High-Assurance Applications

Axel van Lamsweerde

University of Louvain

B-1348 Louvain-la-Neuve

avl@info.ucl.ac.be

SecappDev '08, Leuven, March 2008

High-Assurance Applications (HA)

McLean'95

Applications where *compelling evidence* is required that the system delivers its services in a manner that satisfies certain critical properties such as:

safety,

security,

survivability,

fault tolerance

High-Assurance Applications (2)

Survivability: ability of a system to fulfill its mission in a timely manner even in the presence of (external) incidents or attacks

Fault tolerance: ability to avoid or mitigate failure even in case of fault

Fault: (internal) cause of failure

Failure: deviation between actual & specified behavior

HA applications: problems & challenges

- The later software defects are found,
  the more expensive & dangerous they are ...

  ⇓

  - Start caring for high assurance *early*, i.e. at requirements engineering time

  - Preserve high assurance at every transition to downstream artefacts (architecture, test data, code)

HA applications: problems & challenges (2)

- A posteriori detection & fix of software defects may endlessly generate other defects ...

  $\Downarrow$

  - Adopt a constructive approach where high assurance is granted *by construction*

HA applications: problems & challenges (3)

- High assurance requires much stronger level of confidence ...

  ⇓

  - *Stronger confidence* requires more formal elaboration & analysis (supported by tools)
  - *Usability* at requirements engineering time requires lightweight techniques

Requirements engineering for HA applications: problems & challenges

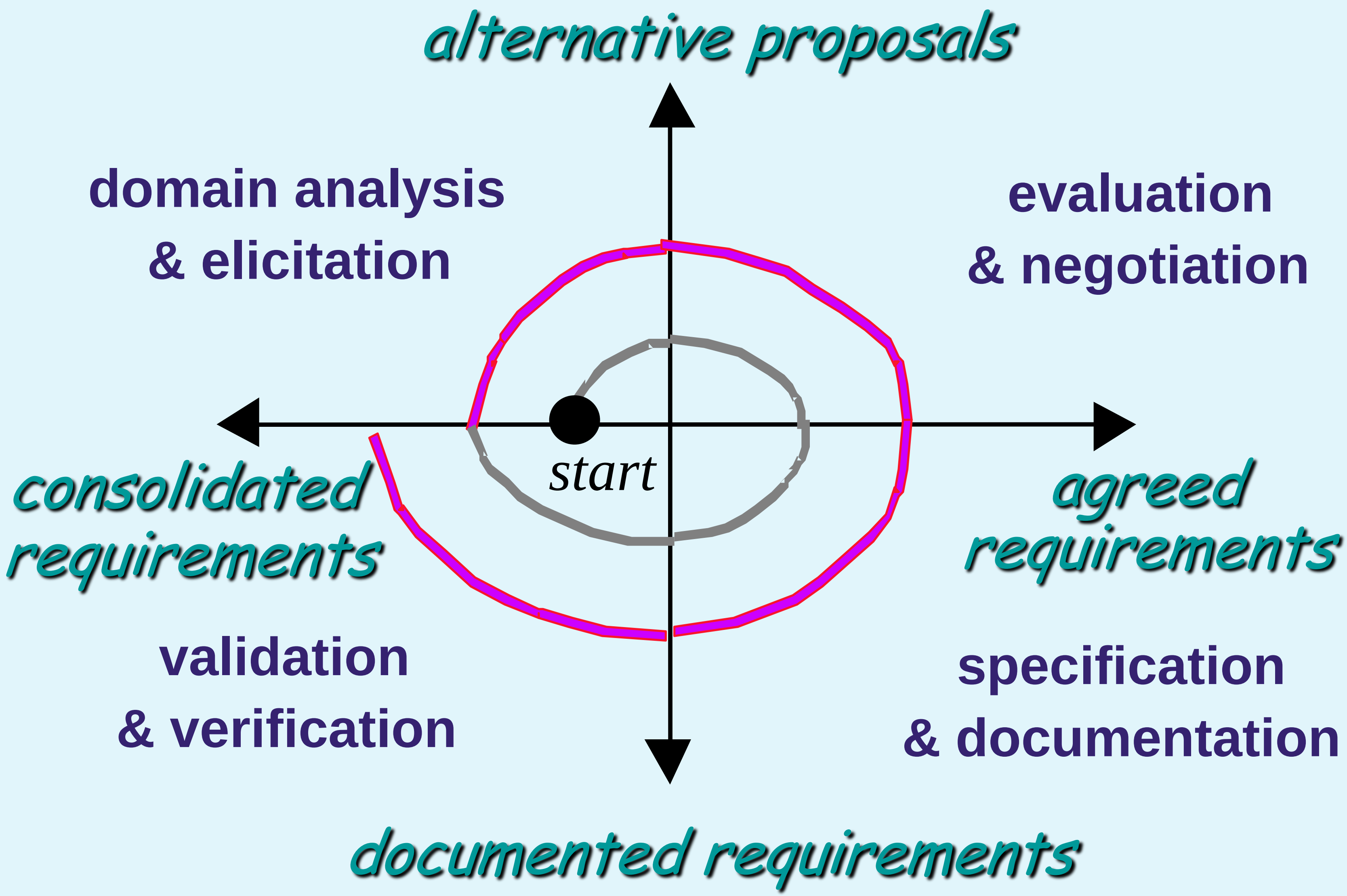
- Requirements Engineering (RE) ≠

  translating informal requirements into ± formal model

- Requirements are not there, you need to ...
  - elicit them,
  - evaluate them,
  - structure & document them,
  - analyze them,
  - modify them

RE: the WHY, WHAT, WHO dimensions

RE: an iterative process

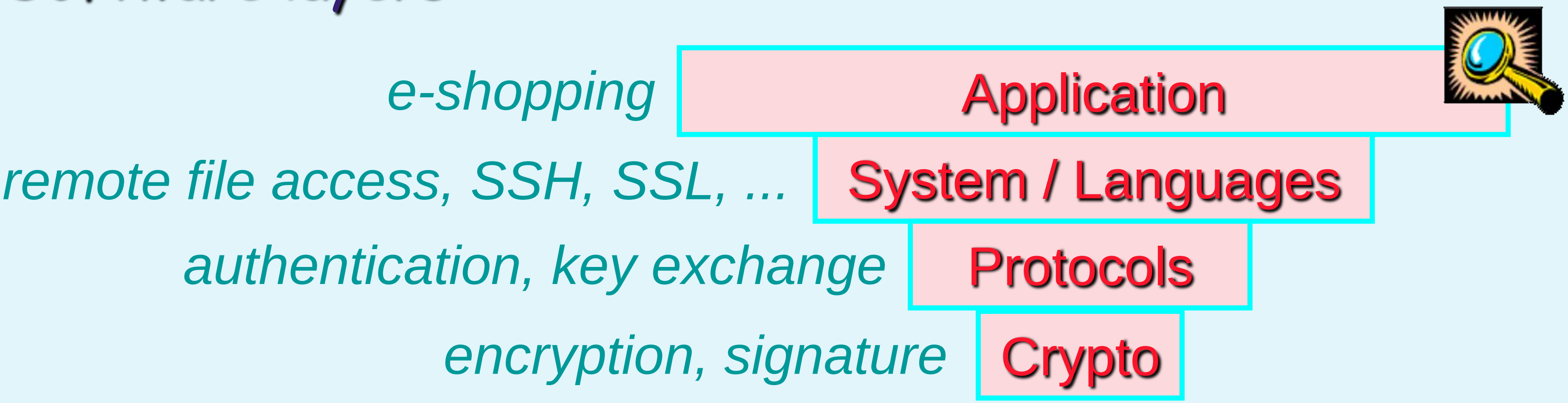


Requirements engineering is hard ...

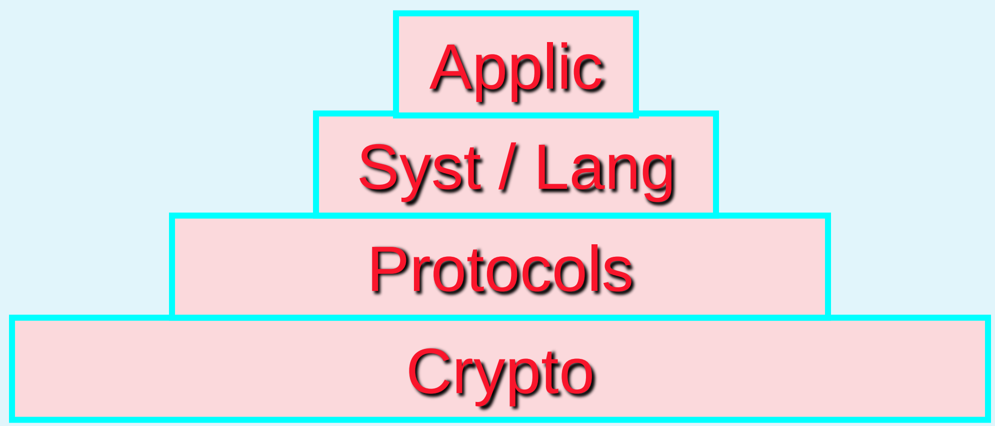
- System = software + environment (possibly malicious)
- Involves 2 systems: system-as-is, system-to-be
- Ranges from high-level, strategic objectives to detailed, technical requirements
- Requires evaluation of alternatives
- Raises conflicting concerns
- Requires anticipation of unexpected behaviors (for requirements completeness, system robustness)

Security engineering: problem space vs. solution space

Software layers



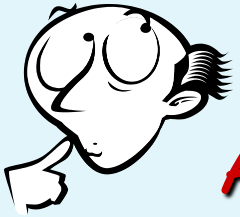
Security guarantee



Focus of these lectures

- Critical properties in HA systems
- *In particular:* security at upper, application layer
- Application is secure iff it satisfies a "complete" set of security goals
  - about confidentiality, integrity, availability, privacy, ...
- Necessary condition for application security:

  security goals must be made *explicit*, *precise*, *complete*, *adequate*, *non-conflicting*



A RE method for HA applications should be ...

- Goal-oriented: to ensure that requirements satisfy system objectives -- notably, *safety, security goals*
- Incremental: for *early* analysis of partial models
- Constructive: for analyst guidance & confidence
- Model-based: for abstraction & structure

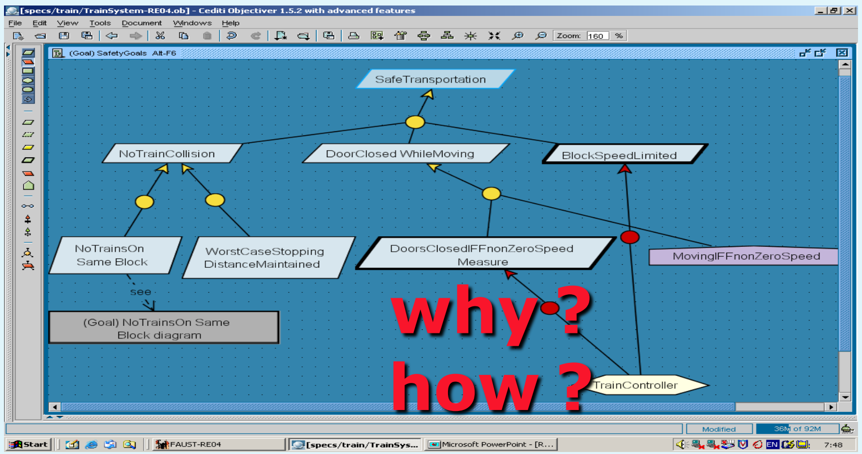
  Multiple models: for capturing multiple facets
- Mix declarative and operational styles as needed
- Formal when and where needed, but *lightweight*

Course outline

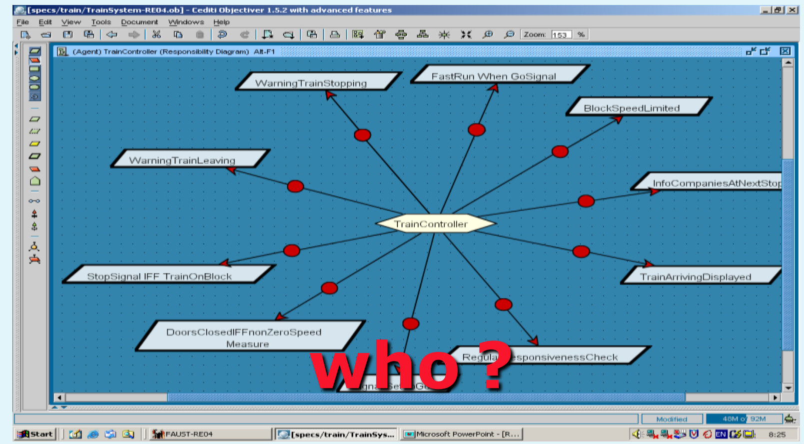
- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  - A goal-oriented model building method in action
  - Obstacle analysis for high assurance
  - Formal reasoning about models
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals
  - Model checking against confidentiality requirements

What models ?

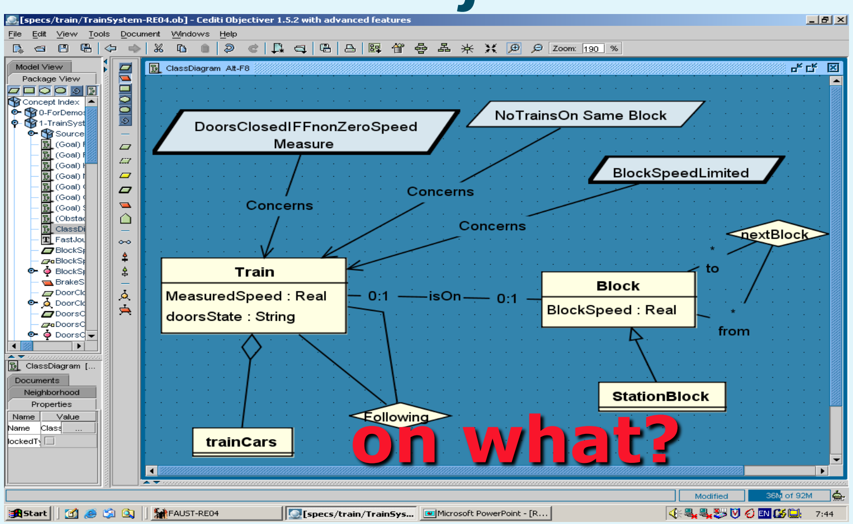
Goals



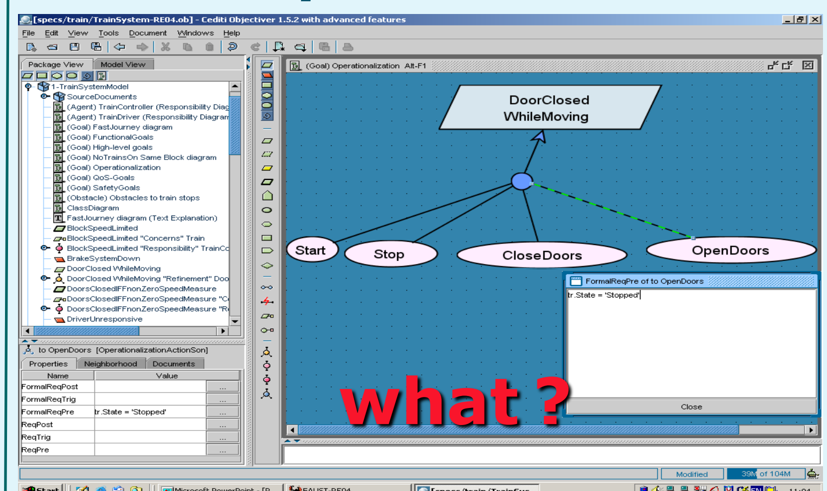
Agents, responsibilities



Objects

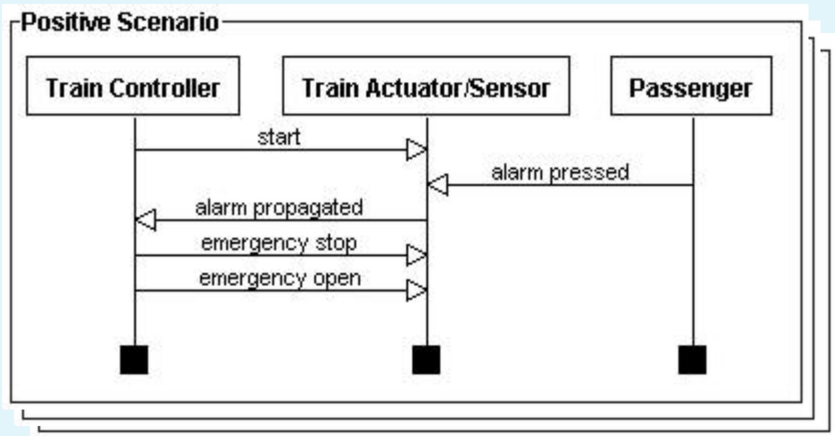


Operations

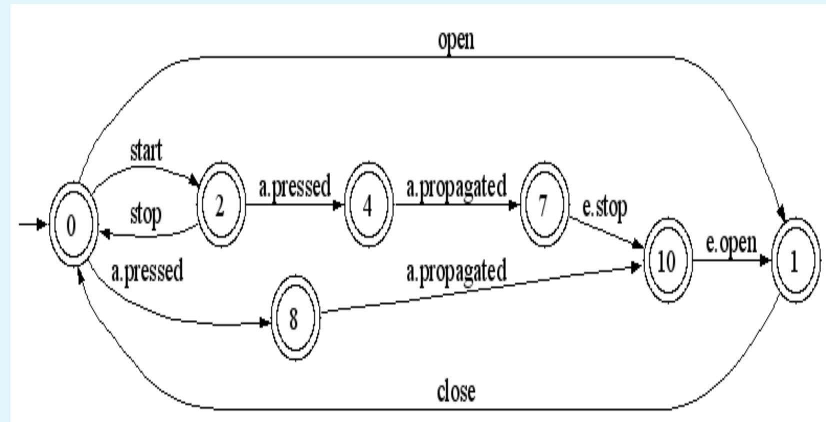


What models ? (2)

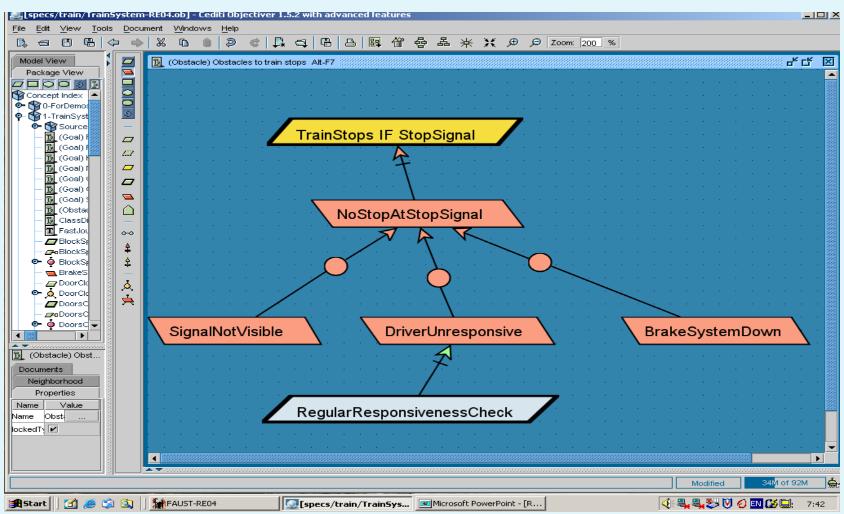
Interaction scenarios



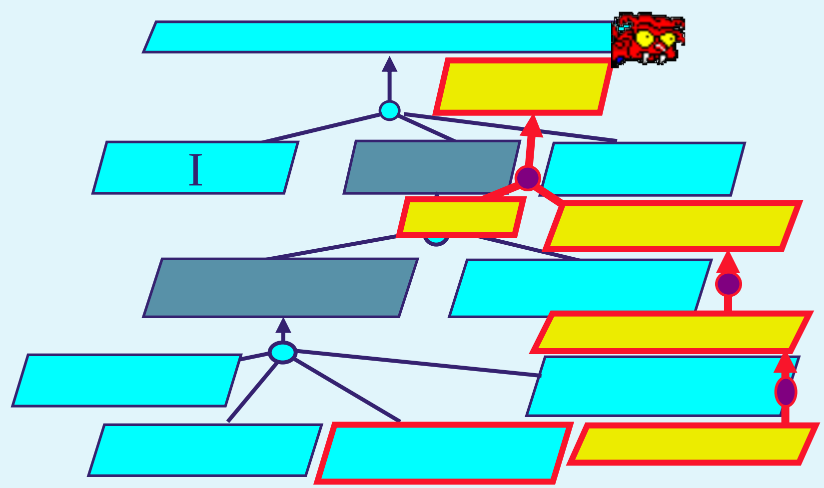
Behaviors

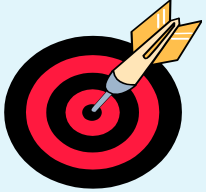


Hazards



Threats





The goal model

- Intentional view of the system being modeled
- Goal = objective to be achieved by *system* ...
  - *prescriptive* statement of intent about system
  - system (as-is, to-be) = software + environment

    "E-money shall be paid only if sufficient e-purse balance"
- ... unlike domain property ...
  - *descriptive* statement about environment

    "Paid money is no longer in purse"

Goals in a goal model have different granularities & abstraction levels

- Higher-level, coarser-grained goals ...

  strategic, global, business-specific

  "Cash-free payment supported anywhere anytime"

- Lower-level, finer-grained goals ...

  technical, local, design-specific

  "E-purses shall have a max capacity of *X* euros"

Goals in a goal model are of different types

- **Behavioral goals prescribe maximal sets of admissible system behaviors**

Achieve *[TargetCondition]:*

if CurrentCondition **then sooner-or-later** TargetCondition

Achieve [E-moneyMovedAsNeeded]

Maintain *[GoodCondition]:*

always (**if** CurrentCondition **then** GoodCondition)

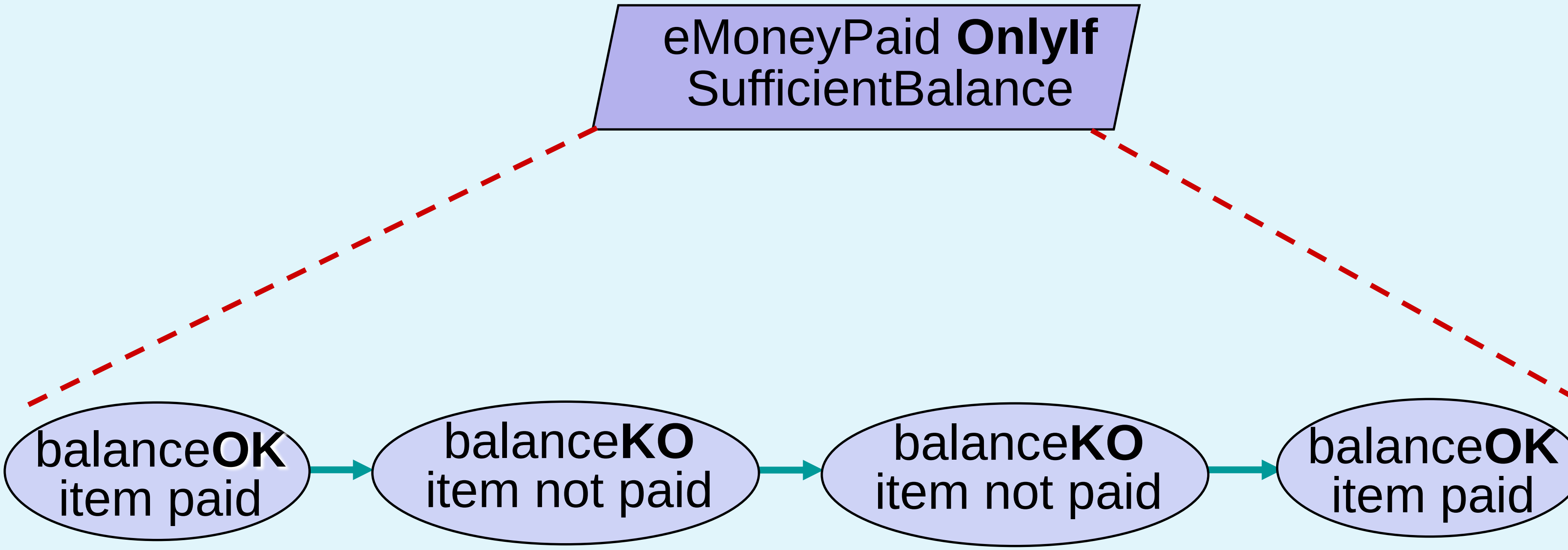
Maintain [E-moneyAccuratelyStored]

Avoid *[BadCondition]:*

never (**if** CurrentCondition **then** BadCondition)

Avoid [E-purseBalanceDisclosedToNonOwners]

Behavioral goals prescribe intended behaviors declaratively



Goals in a goal model are of different types (2)

- Soft goals prescribe preferences among alternative system behaviors
  - cannot be established in a clear-cut sense
  - used to *compare alternative* options

***Improve**, **Increase**, **Reduce**, … [TargetCondition]*

Reduce [BankClerkWorkload]

⇓

Achieve [ePurseLoadedAtATM]

preferred over

Achieve [ePurseLoadedAtBank]

Goals in a goal model are of different categories

- Functional goals state intent behind system services
  - Used to build operational models: use cases, state machines, task workflows, ...

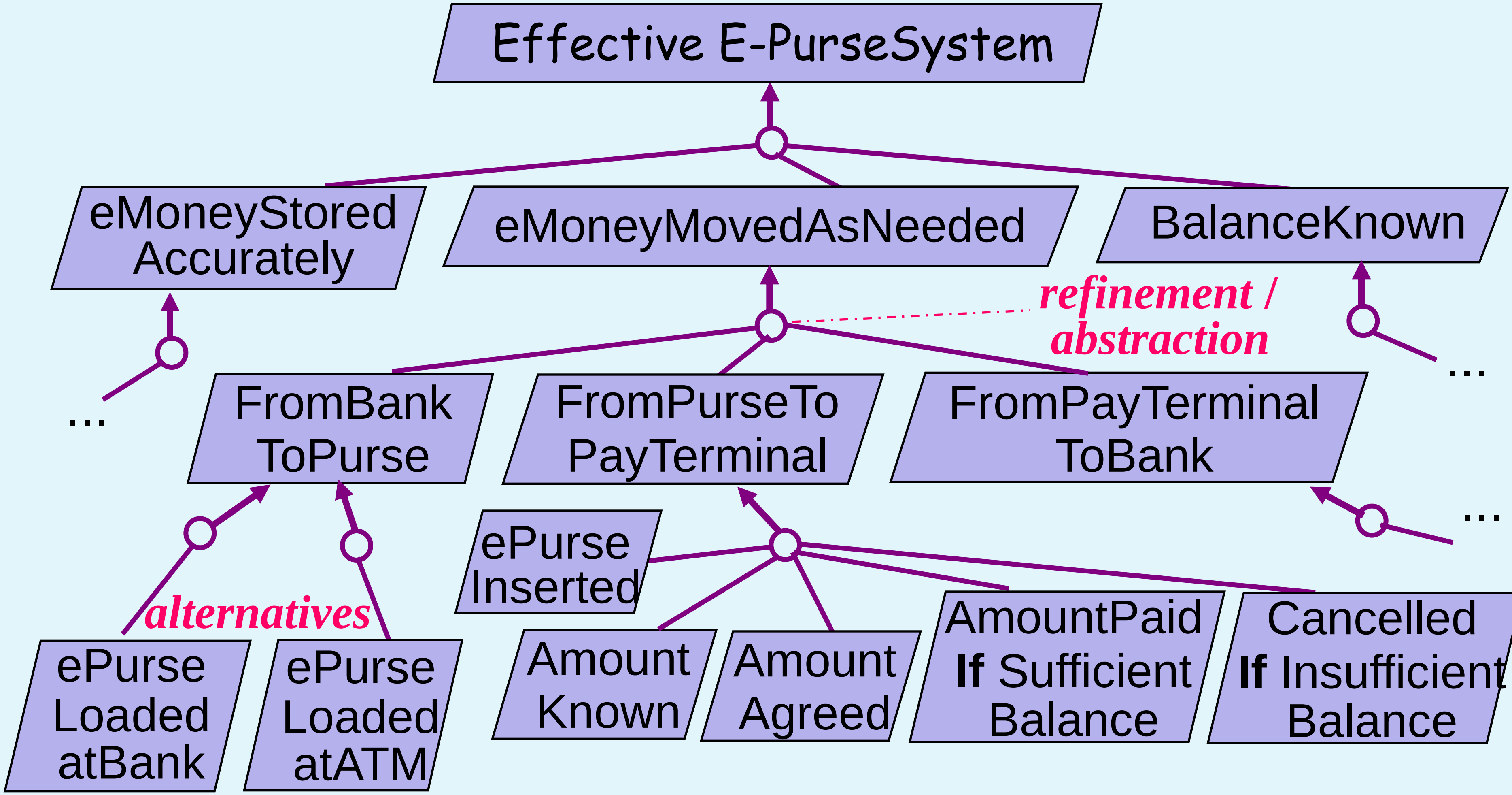
    EmoneyMovedFromEpurseToPayTerminal

- Quality goals constrain quality of service ("non-functional goals")
  - About security, safety, accuracy, usability, cost, performance, interoperability, etc.

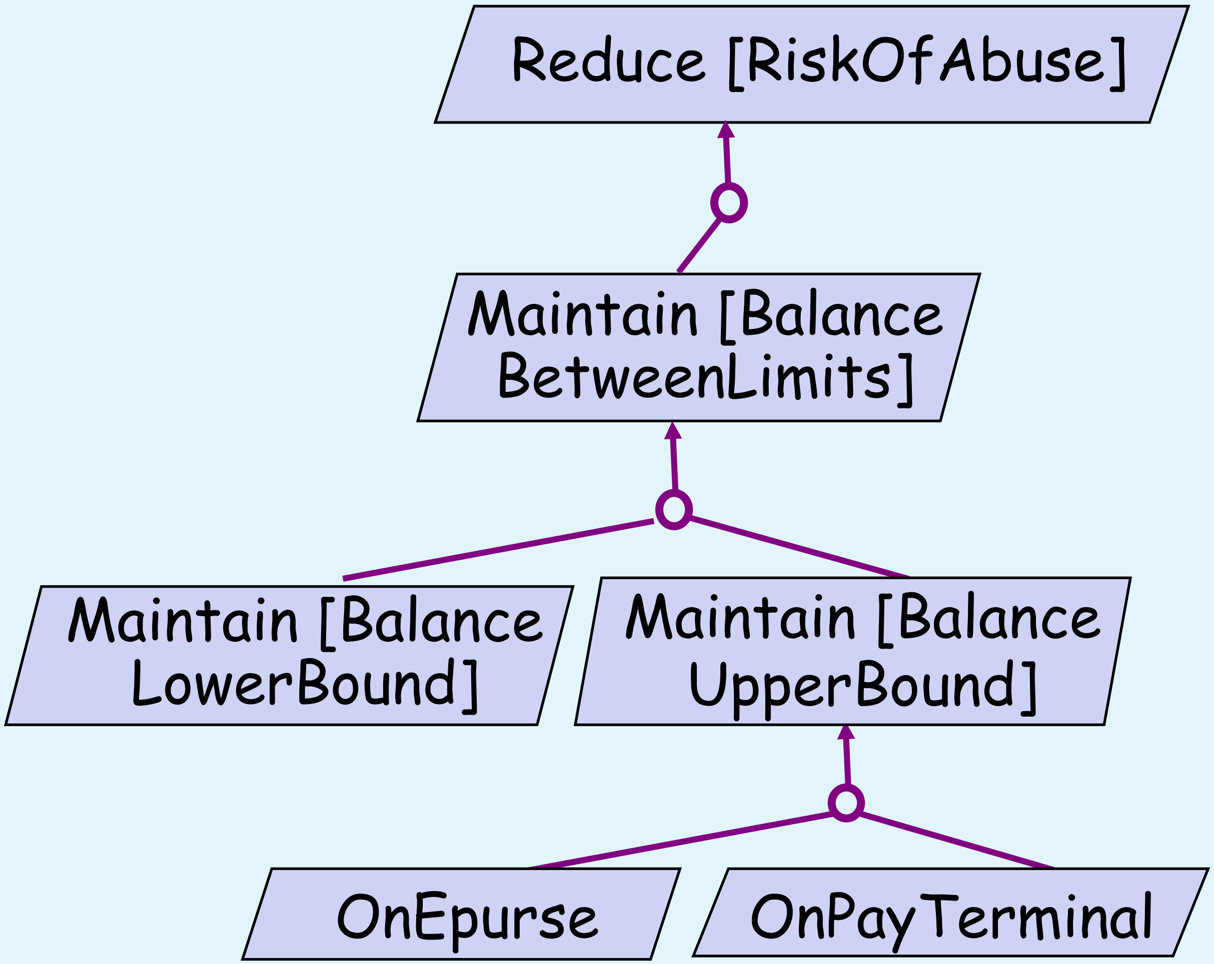
    Maintain [eMoneyStoredAccurately], Improve [PurseUsability]

  - Some are softgoals (e.g. "ility" goals)
  - Often conflicting

The goal model shows contribution links



Refining a security soft goal into behavioral goals



Goal specifications annotate the goal model

Goal *Achieve* [AmountPaid **If** SufficientBalance]

Def *A payment shall be done for some input amount through e-purse debit and pay terminal credit if the amount is OK-ed by the payer and the e-purse balance is higher or equal to this amount*

[**FormalSpec** $\forall$ ep: e-Purse, pterm: PayTerminal, p: Payer

Inserted (ep, p, pterm) $\wedge$ OK (pterm.InputAmount, p)

$\wedge$ pterm.InputAmount $\leq$ ep.Balance

$\Rightarrow \mathbf{o}$ (pterm.Balance = $\bullet$ pterm.Balance + pterm.InputAmount

$\wedge$ ep.Balance = $\bullet$ ep.Balance - pterm.InputAmount)]

[Priority Highest]

...

Optional formalization in *real-time temporal logic*

("next", "always", "sooner-or-later", ...) for formal reasoning

Goal satisfaction requires agent cooperation

- **Agent =** role (rather than individual)
  responsible for goal satisfaction

Achieve [eMoneyMovedFromPurseToPayTerminal] ↔

Payee, Payer, ePurse, PayTerminal

- **Agent types:**
  - software
    (software-to-be, legacy software, COTS components, foreign components)
  - devices (sensors, actuators, ...)
  - humans playing specific roles

Goal satisfaction requires agent cooperation (2)

- Finer-grained goal ⇒ fewer agents required for goal satisfaction
- Requirement = goal assigned to single agent in *software-to-be*

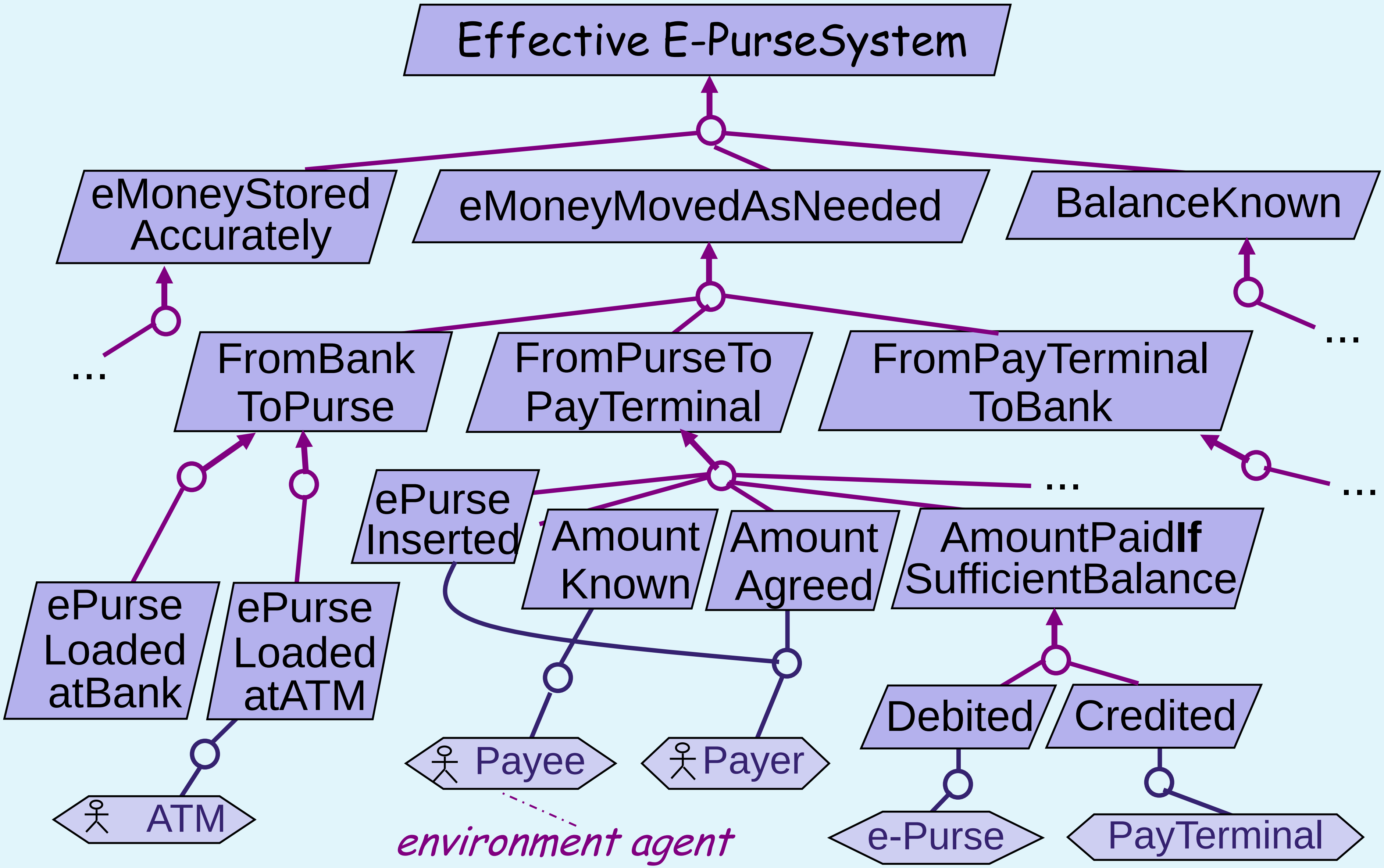
  Achieve [AmountDebitedIfSufficientBalance] ↔ ePurse

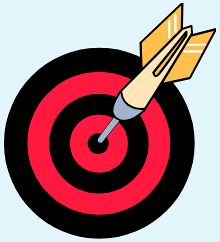
  Achieve [AmountCreditedIfDebited] ↔ PayTerminal

- Expectation = goal assigned to single agent in *environment*

  Achieve [AmountAgreed] ↔ Payer

Goals are refined until single responsibilities can be assigned





WHY are goals so important ?

- Abstraction level for strategic stakeholders (decision makers)
- Force environment assumptions to be made explicit
- Criterion for requirements completeness

  REQ is complete if for all G:

  $\{REQ, EXPECT, Dom\} \vdash G$

- Criterion for requirements relevance

  r in REQ is pertinent if for some G:

  r is used in $\{REQ, EXPECT, Dom\} \vdash G$

High assurance requires satisfaction arguments

- Informal argument at least, formal argument at best

  R, E, D |— ***G***

  " in view of properties *D* of the domain,
  the requirements *R* will achieve goals ***G***
  under expectations *E* "

  R1: amount debited from e-purse

  R2: same amount credited to pay terminal

  D: amount paid if debited from e-purse and credited to terminal

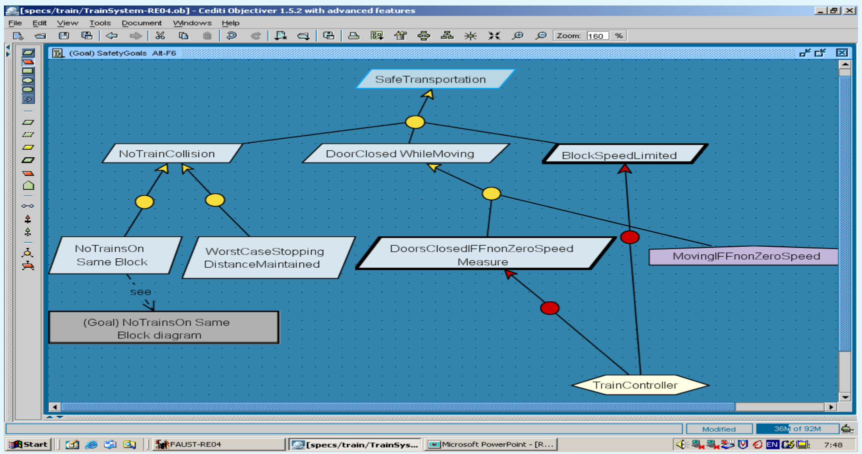
  E: amount agreed by payer

  |— ***G:*** *amount agreed and paid*

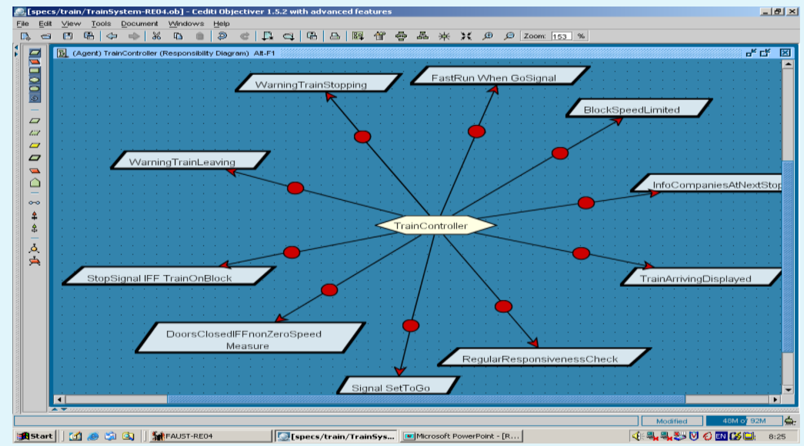
- A goal model supports satisfaction arguments & traceability links for free

What models ?

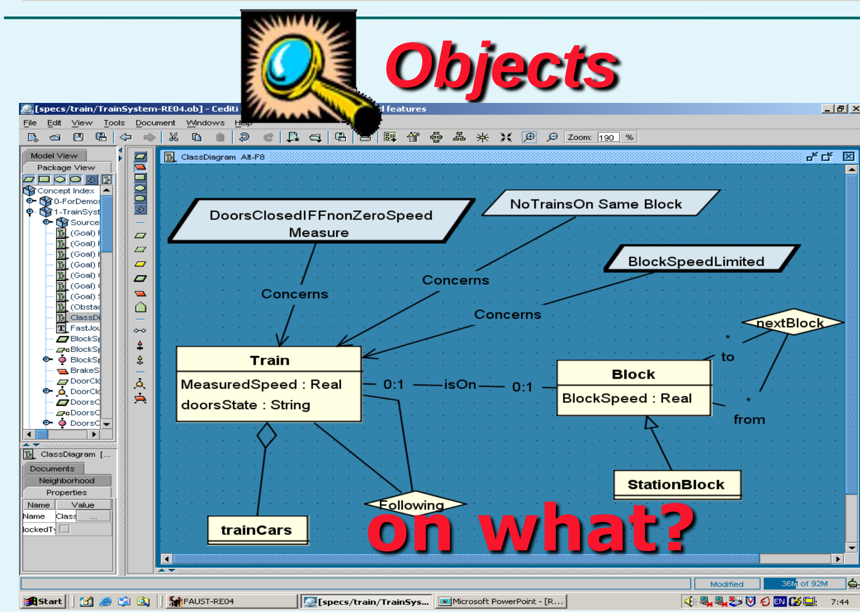
Goals



Agents & responsibilities

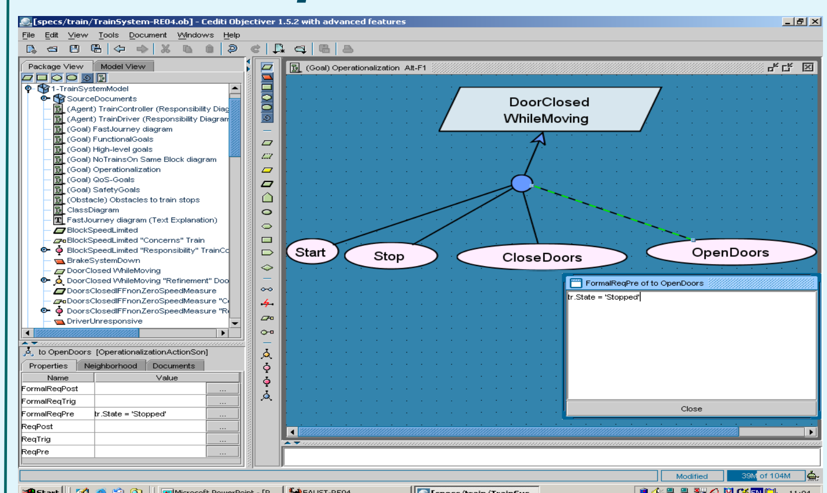


Objects



on what?

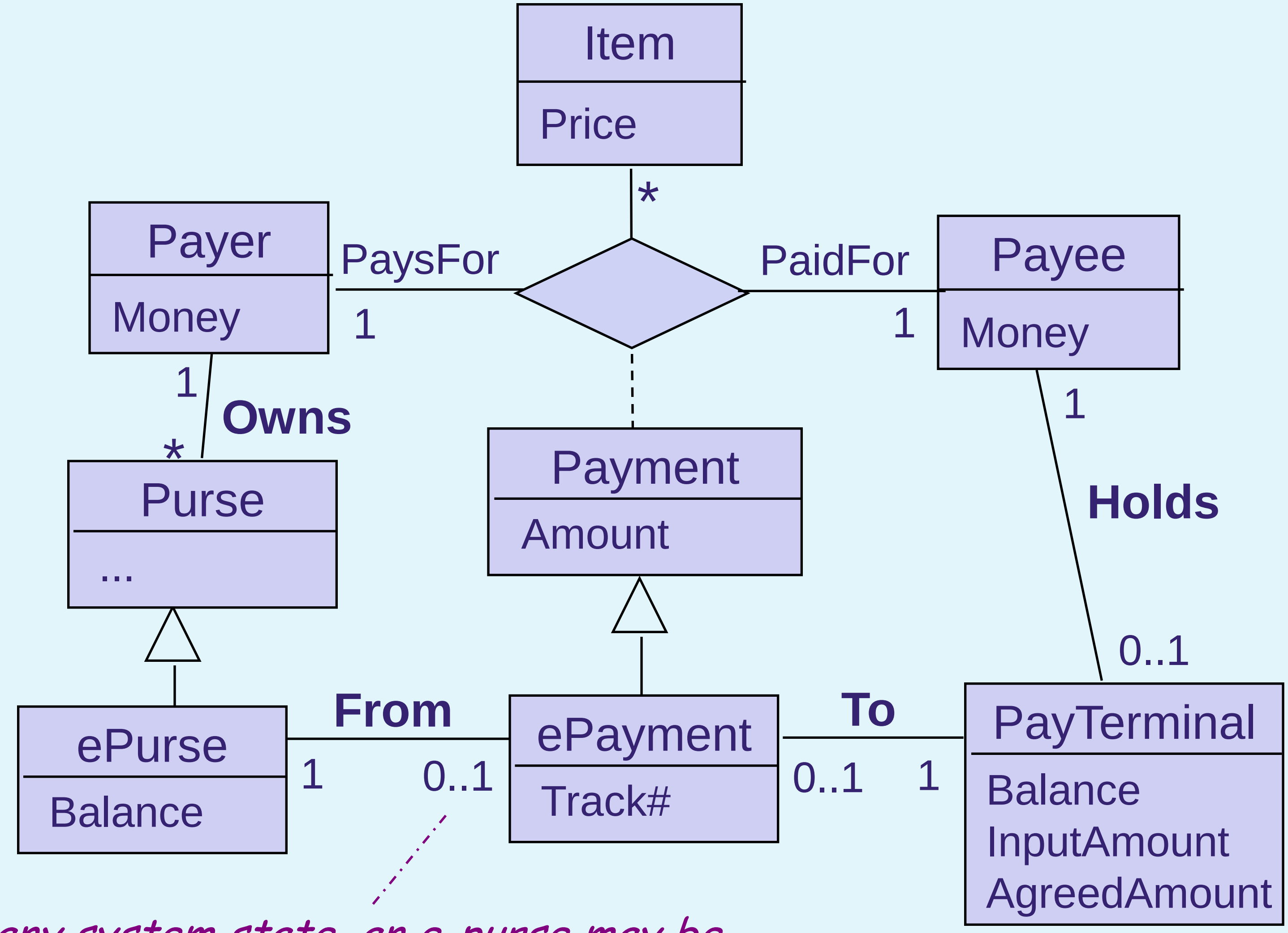
Operations



Modeling objects

- Structural view of the system being modeled
- Object = thing of interest in the system whose instances ...
  - share similar features (attributes, associations)
  - can be distinctly identified
  - have specific behavior from state to state
- Object attributes/associations yield state variables
- Object specializations (at meta level):
  - entity: autonomous object
  - association: object dependent on objects it links
  - event: instantaneous object
  - agent: active object, controls behaviors

The structure of objects is modeled using UML



in any system state, an e-purse may be involved in at least 0 and at most 1 e-payment

Object specifications annotate the object model

Relationship Payment

Def *Condition for an item to be sold by a payee to a payer*

DomInvar An item is paid if its price is debited from the payee and credited to the payer

[FormalSpec $\forall$ it: Item, pyr: Payer, pye: Payee

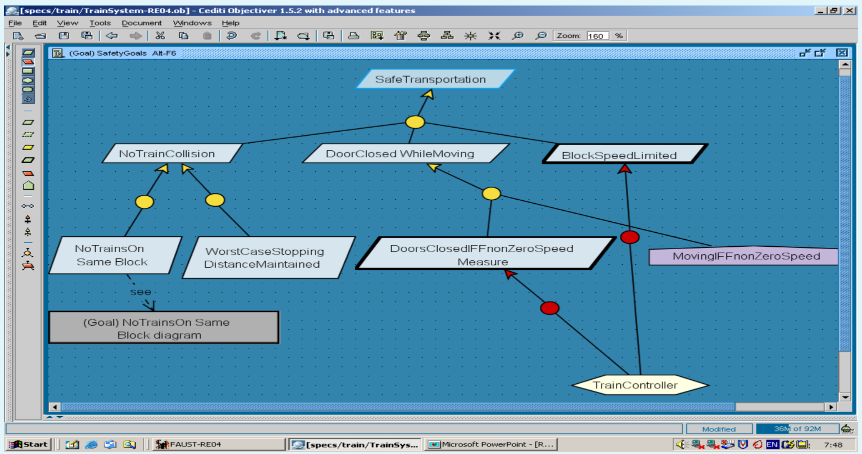
Payment (pyr, pye, it) $\Rightarrow$ pyr.Money = $\bullet$ pyr.Money - it.Price

$\wedge$ pye.Money = $\bullet$ pye. Money + it.Price]

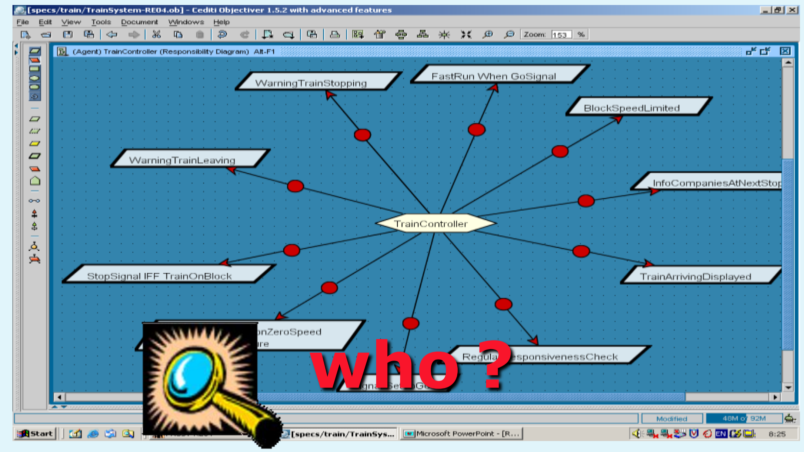
domain properties

What models ?

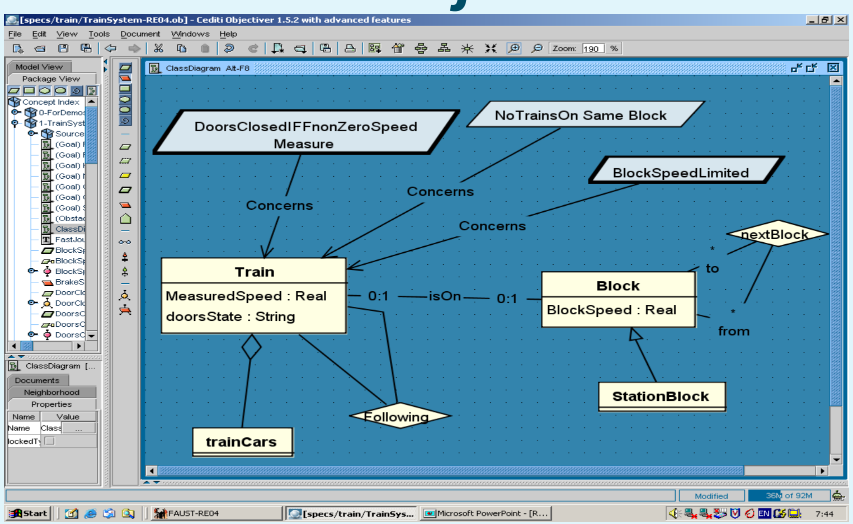
Goals



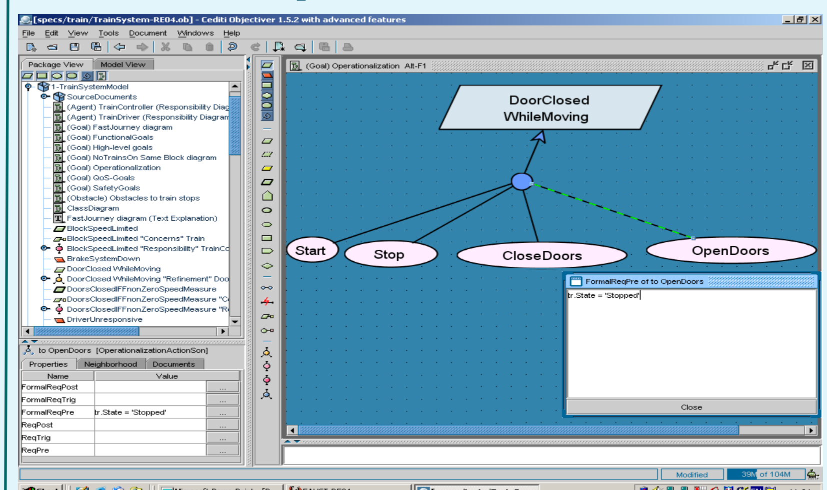
Agents & responsibilities



Objects



Operations



Modeling agents

- Responsibility view of the system being modeled:

  who is in charge of what

- Agent :
  - (Role rather than individual -- software, device, human)
  - Active object: *monitors* & *controls* state variables (through *operations* on attrib, assoc)
  - Runs concurrently with others
  - Agent responsible for goal ⇒

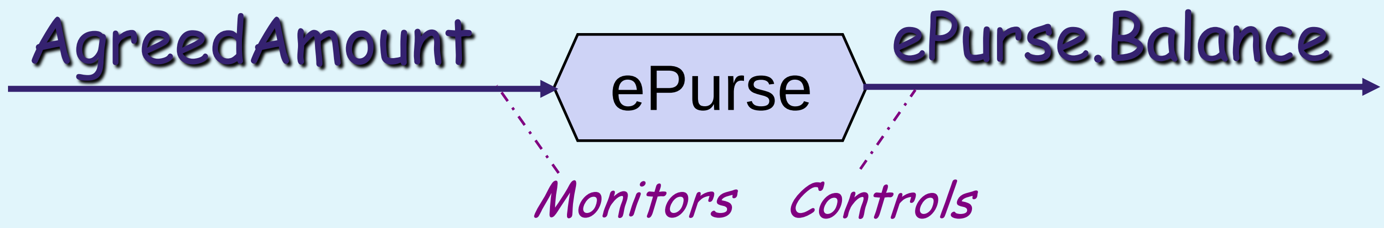
    must restrict system behaviors

    goal must be *realizable* by agent


Goal realizability by an agent

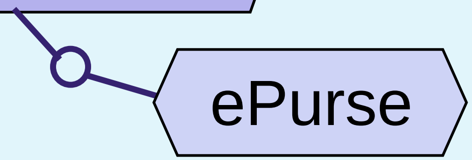
- A goal is *realizable* by an agent *iff* its monitoring & control capabilities enable it/her to satisfy the goal in view of known domain properties (without more restrictions than required by G)
- A goal is *unrealizable* by an agent because of ...
  - lack of monitorability of variables to be evaluated
  - lack of controllability of variables to be constrained
  - reference to future
  - conditional unsatisfiability (aka conflict with other goals)
  - unbounded achievement (liveness property)

Goal realizability & agent capabilities: examples



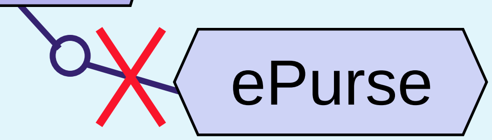
Example 1: Realizable by ePurse

AgreedAmount $\leq$ ep.Balance $\Rightarrow$ ep.Balance = ... **-** ...

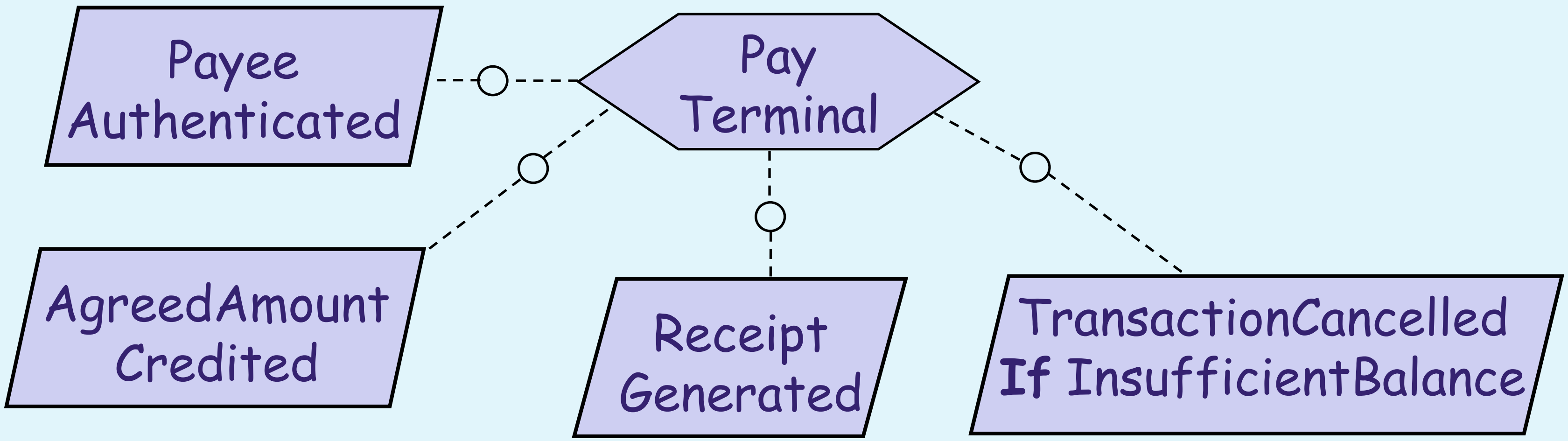


Example 2: Not realizable by ePurse

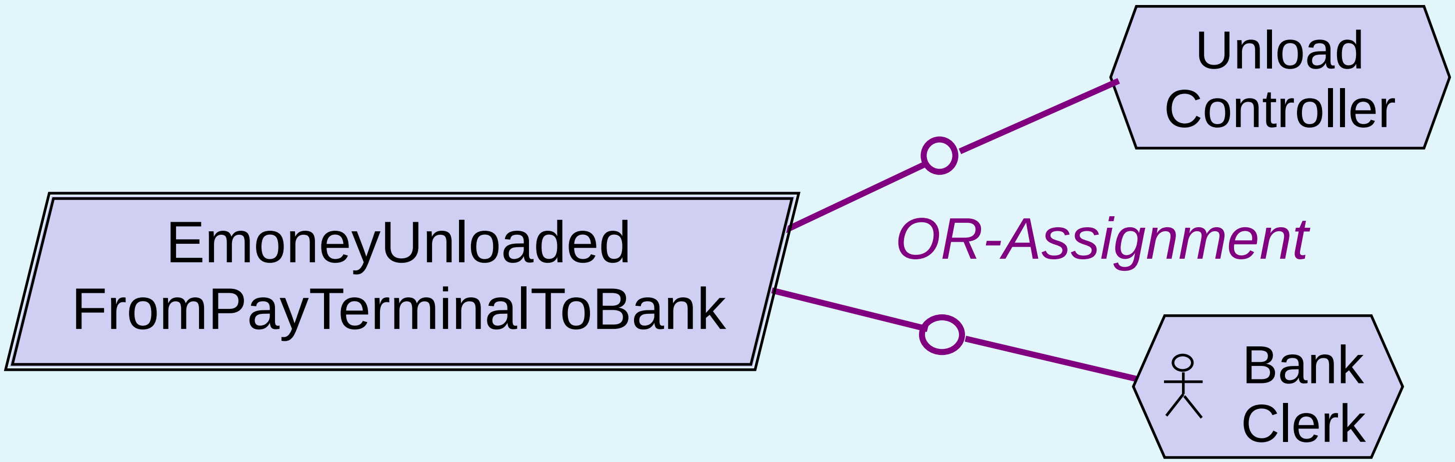
pterm.InputAmount $\leq$ ep.Balance
$\Rightarrow$ pterm.Balance = ... **+** ... $\wedge$ ep.Balance = ... **-** ...



Modeling agents: responsibility view

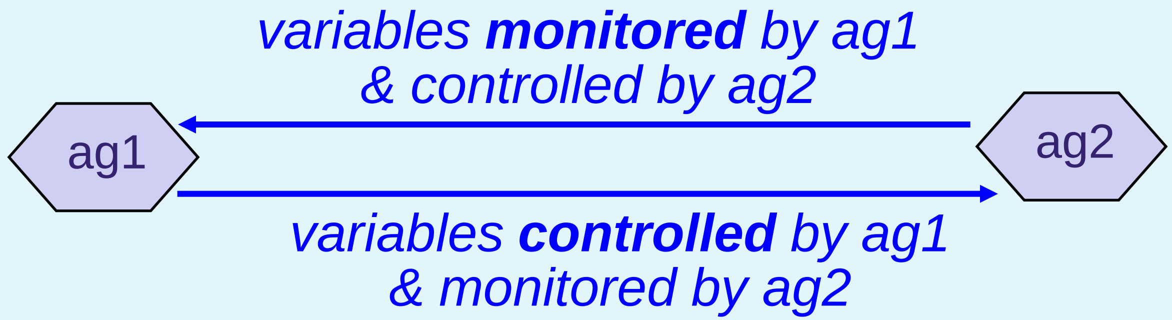


Alternative agent assignments define alternative system boundaries

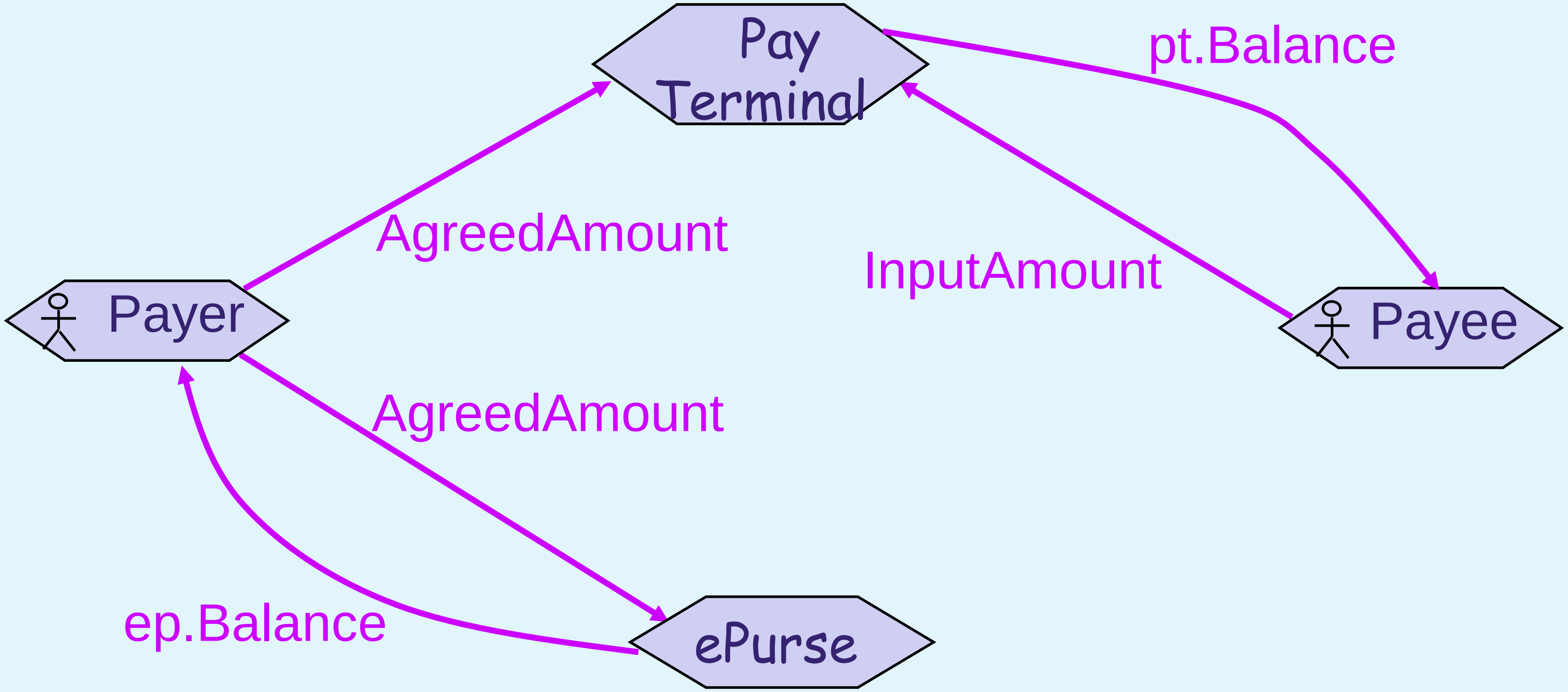


Modeling agents: interface view

- Interface among agents = monitored/controlled state variables
  (attributes/relationships from object model)
- Interface view = context diagram

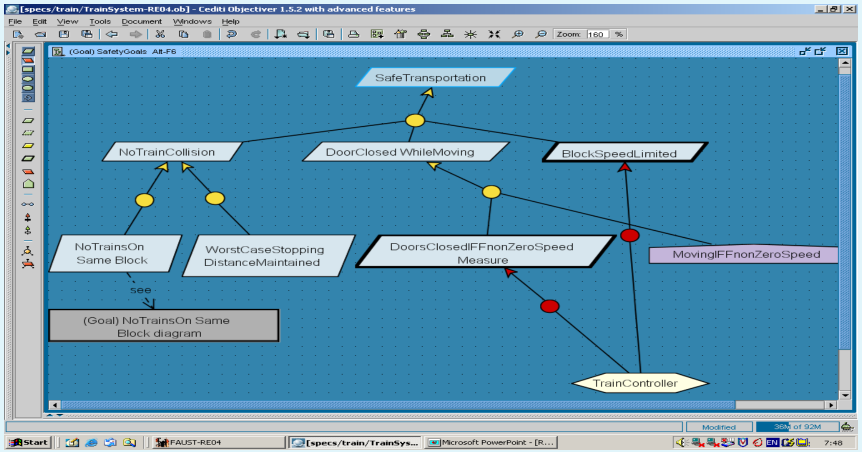


Context diagram: example

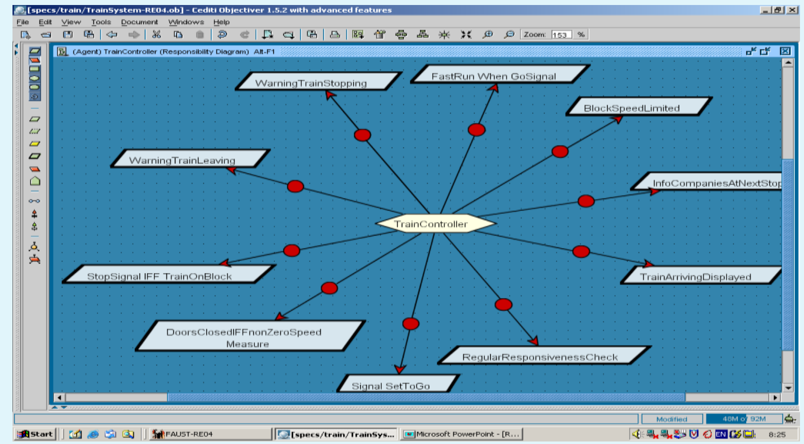


What models ?

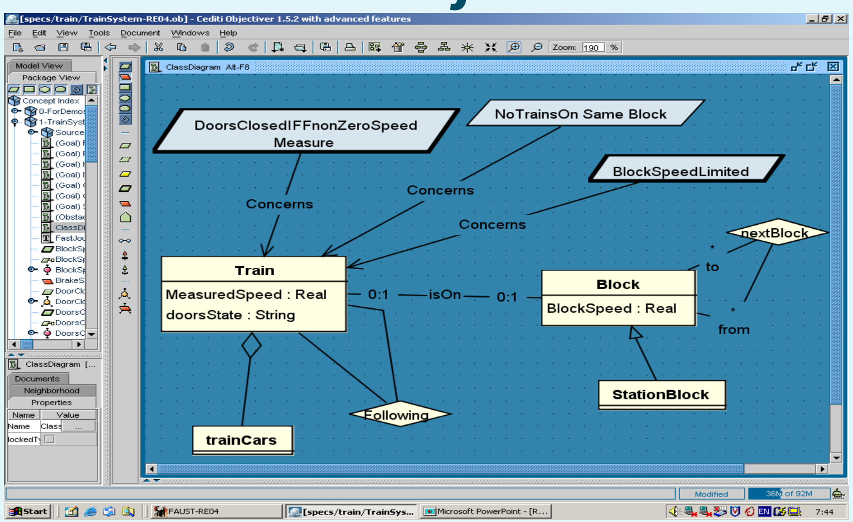
Goals



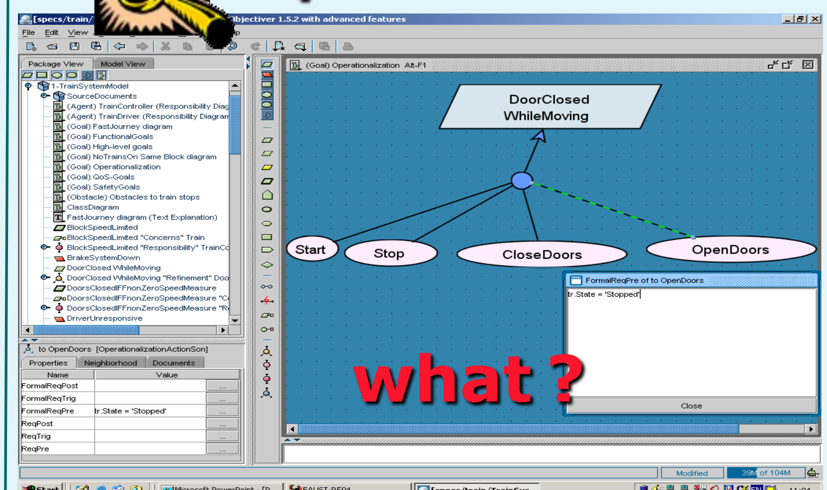
Agents & responsibilities



Objects



Operations



what ?

Modeling operations

- Functional view of the system being modeled:

  what services are to be provided? (statics)

- Operation Op:
  - relation Op $\subseteq$ InputState $\times$ OutputState
  - Op must operationalize underlying goals
  - Op applications define state transitions (events) in behavioral model
  - Op applications are concurrent with others
  - Op is atomic: maps to state at next smallest time unit

    (operations with duration: use start/end events)

Specifying operations

- Name, Def
- DomPre: condition characterizing the class of input states in the domain
- DomPost: condition characterizing the class of output states in the domain
- Links to other models:

  Operationalization (goals), Input/Output (objects), Performance (agent)

Specifying operationalizations

- An *operationalization* of G into Op is specified by:
  - ReqPre: *necessary* condition on Op's input states to ensure G (permission)
  - ReqTrig: *sufficient* condition on Op's input states to ensure G :

    requires immediate application of Op provided DomPre holds (obligation)
  - ReqPost: condition on Op's output states to ensure G
- Consistency rule: ReqTrig $\wedge$ DomPre $\Rightarrow$ ReqPre

Specifying operations: example

Operation ePay

Def *Operation controlling the e-payment for an item*

Input ep: ePurse, pt: PayTerminal; **Output** ePayment

DomPre There is no ePayment from *ep* to *pt*

DomPost There is an ePayment from *ep* to *pt*

ReqPre For *AgreedAmountPaidIfSufficientBalance:*

pt.AgreedAmount $\leq$ ep.Balance

ReqPost For *AgreedAmountPaidIfSufficientBalance:*

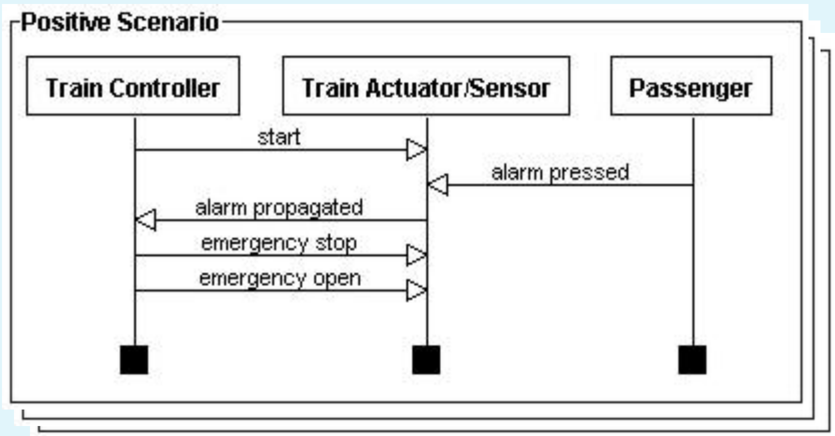
ep.Balance = $\bullet$ ep.Balance - pt.AgreedAmount

ReqTrig For *InstantPaymentUponAgreement:*

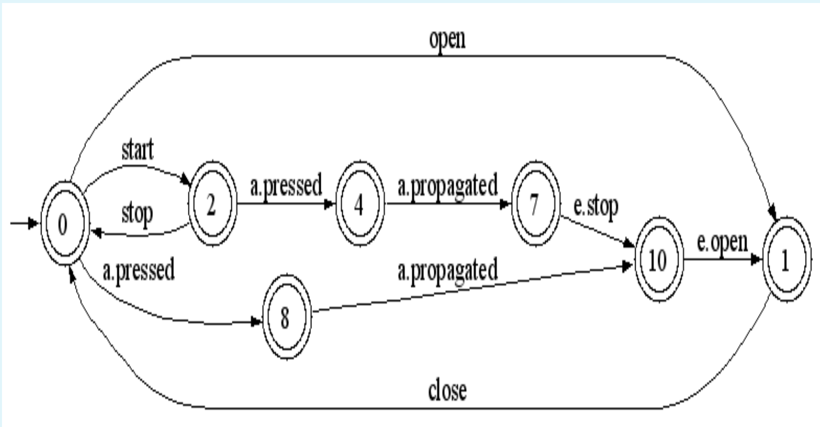
The amount is agreed and the balance is sufficient

What models ?

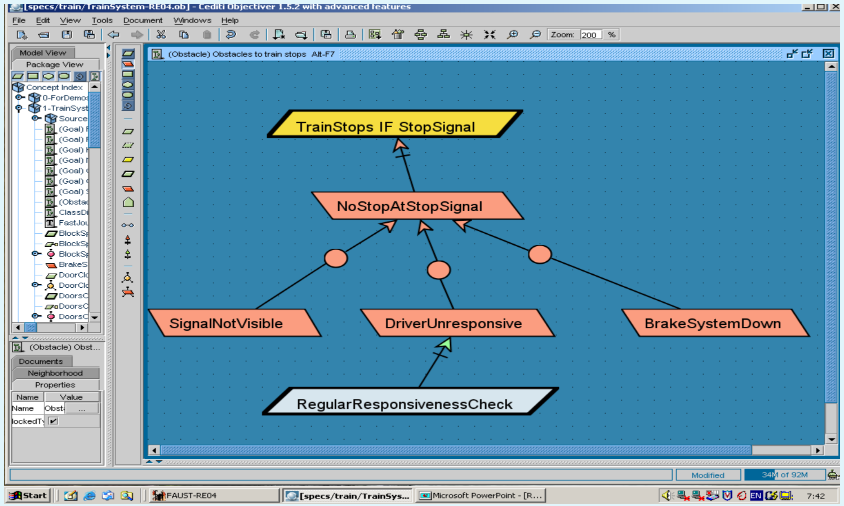
Interaction scenarios



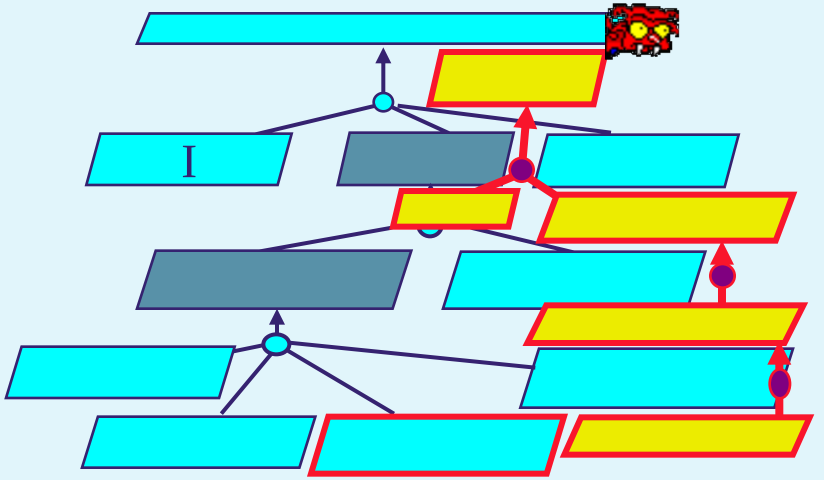
Behaviors



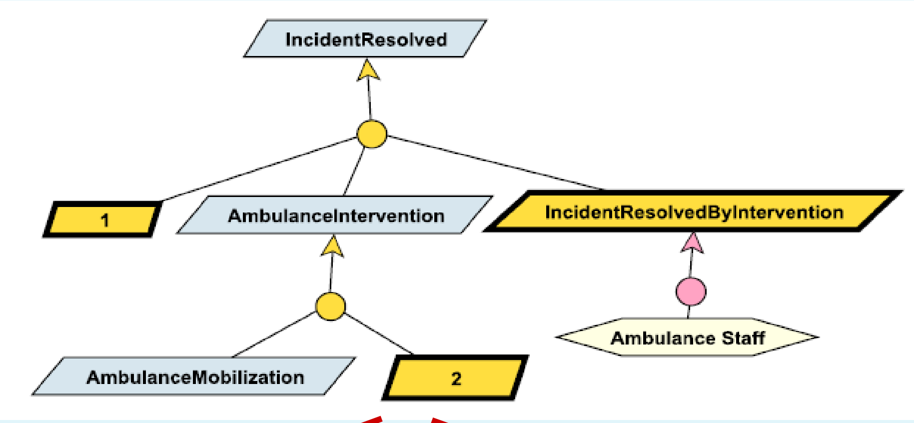
Hazards



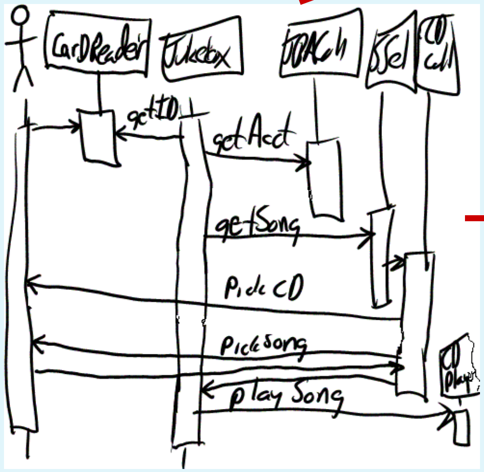
Threats

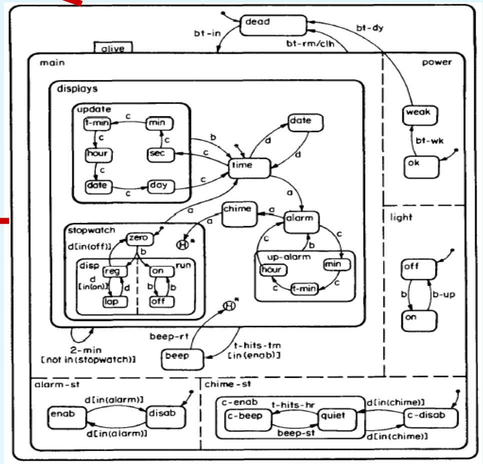


Goals, scenarios, state machines: win-win partners



+ declarative
+ many behaviors
- too abstract?

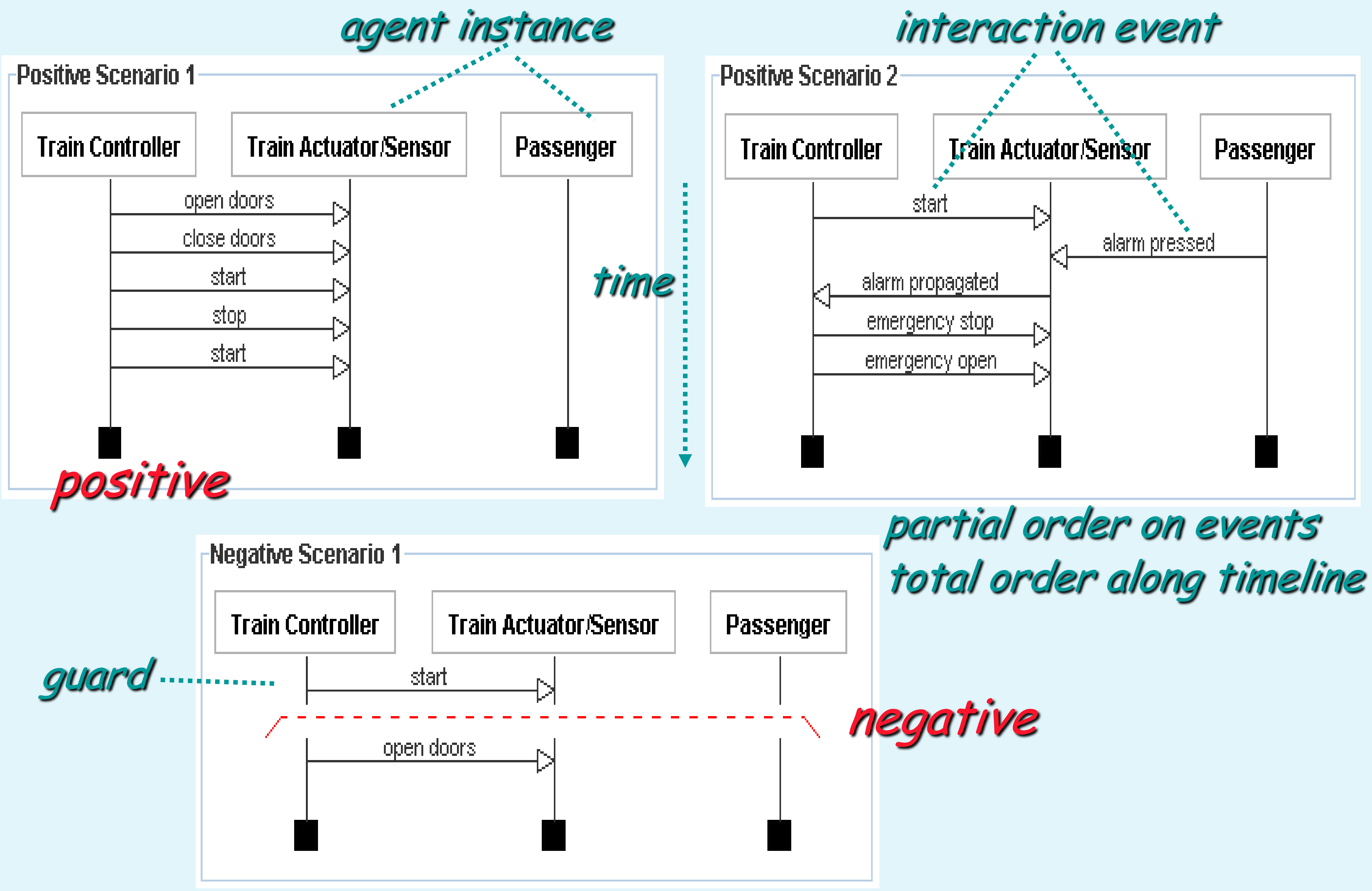




+ concrete examples
- partial, few behaviors

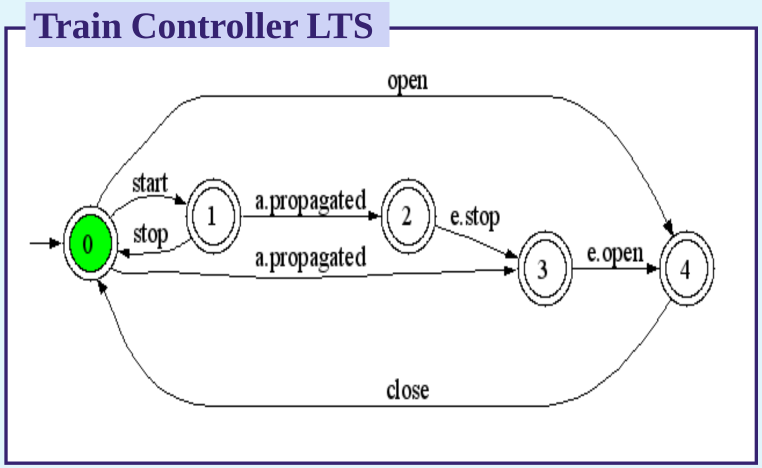
+ model-checkable, executable
- hard to build, understand

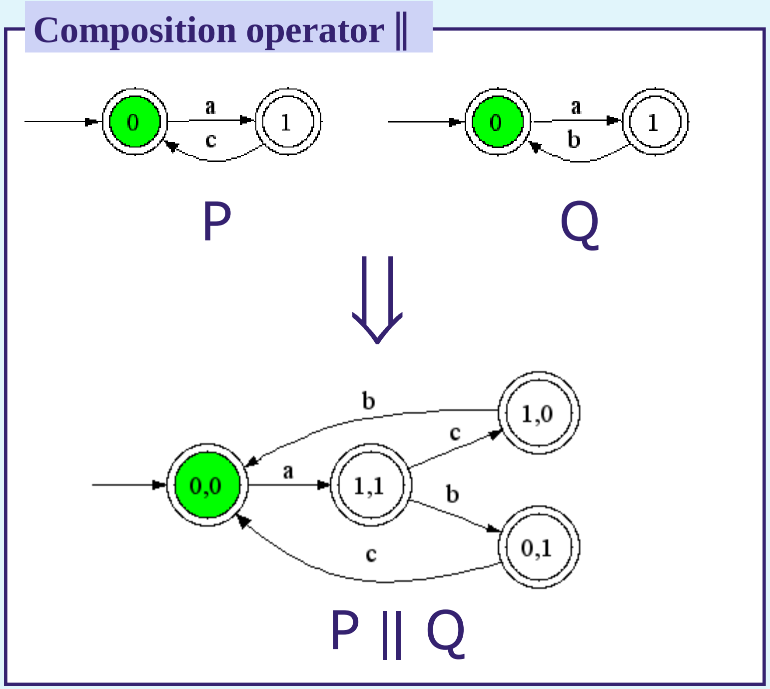
Scenarios as simple MSCs



Modeling behaviors with LTS

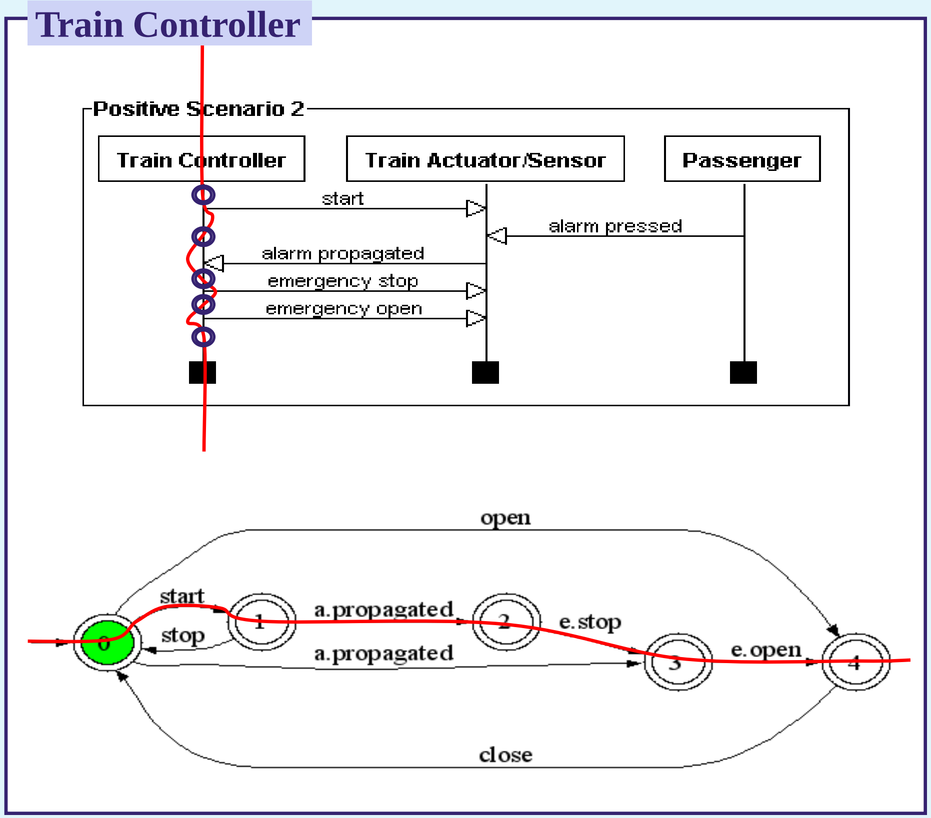
- An agent is modeled as a LTS
- System behavior = composition of agent behaviors
  - Agents behave asynchronously but synchronize on shared events
  - Composition: ||-operator

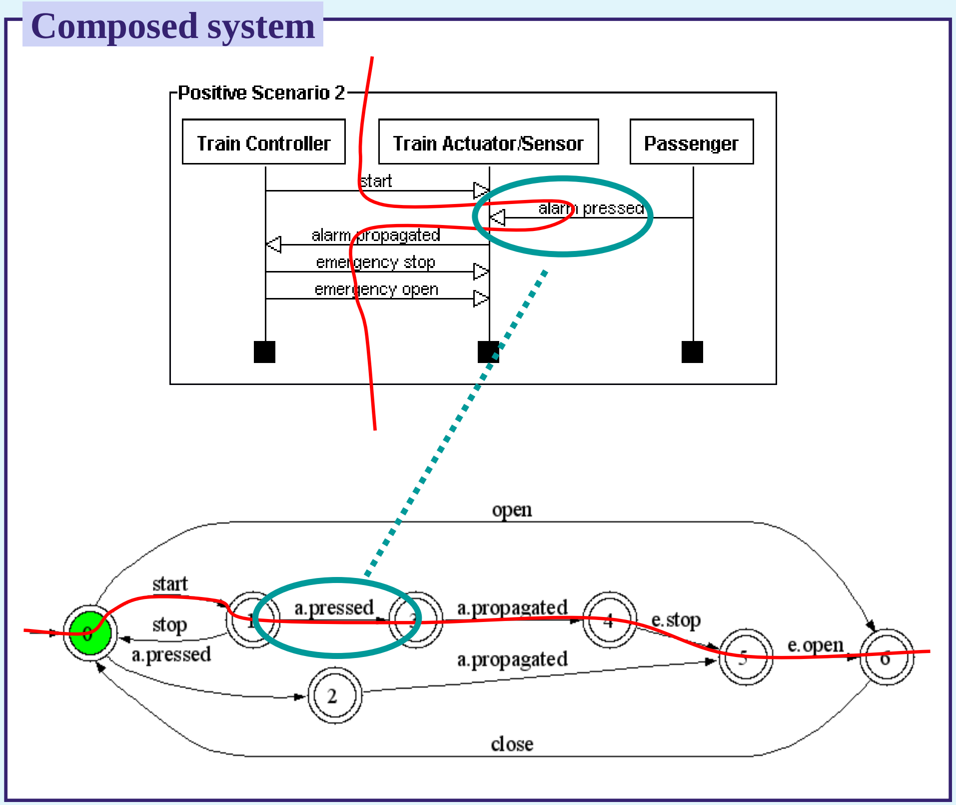




Scenarios vs. behavior models

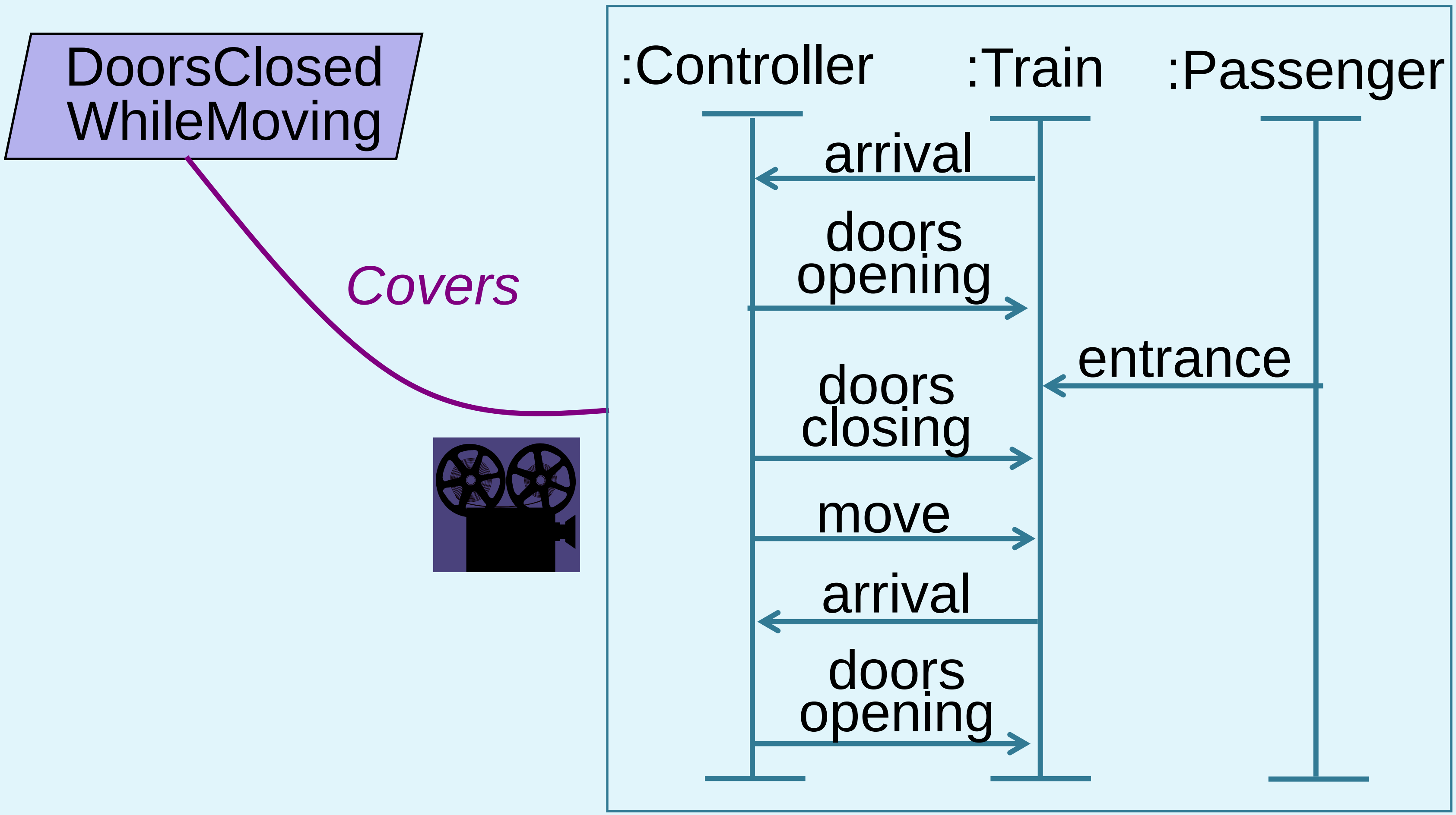
- A scenario defines paths in a behavior model
  - a path in each agent LTS
  - a path in the system's LTS (||)



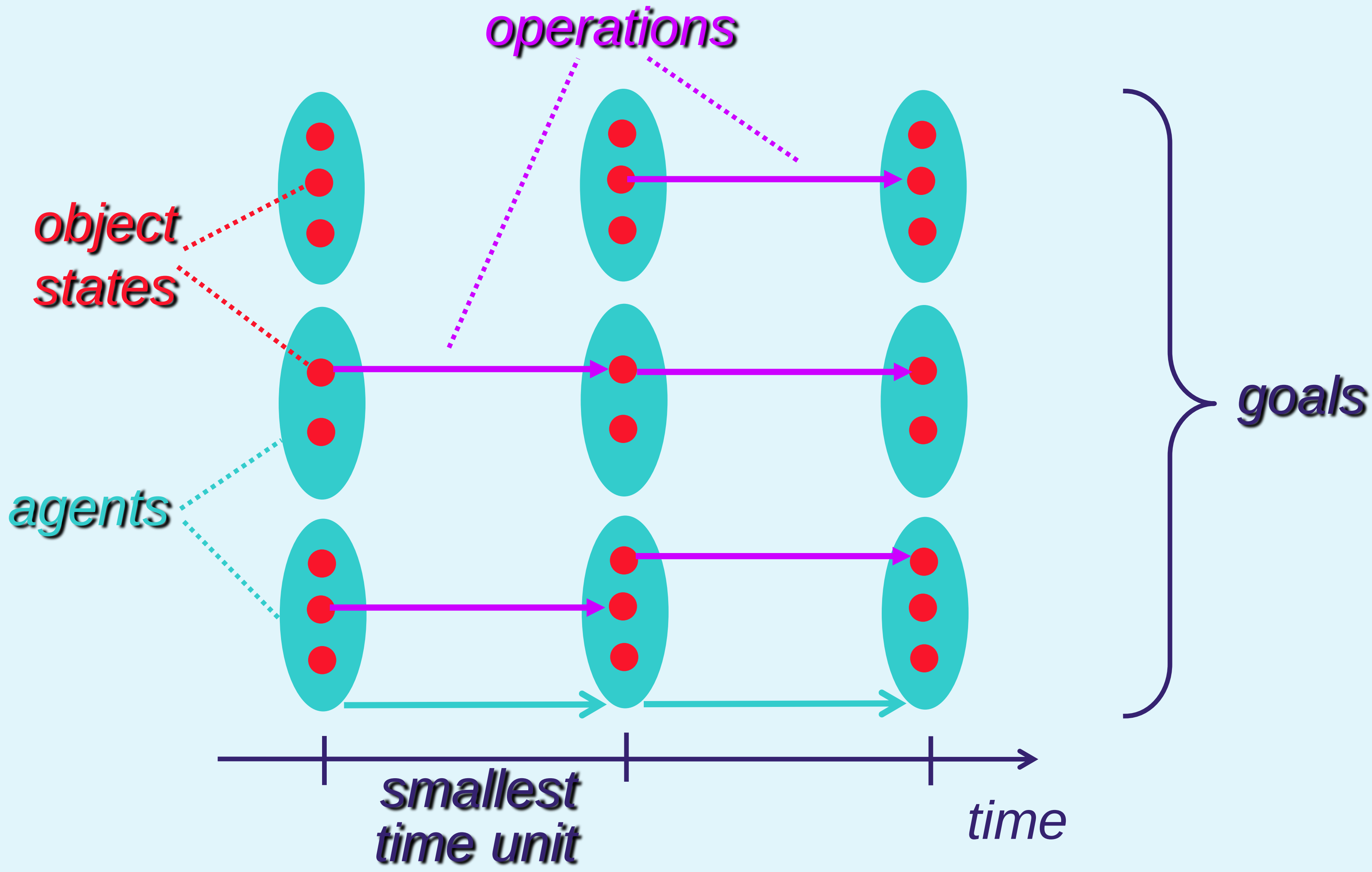


Goals vs. scenarios

- A behavioral goal prescribes a set of scenarios



Goals, objects, agents, operations: the semantic picture

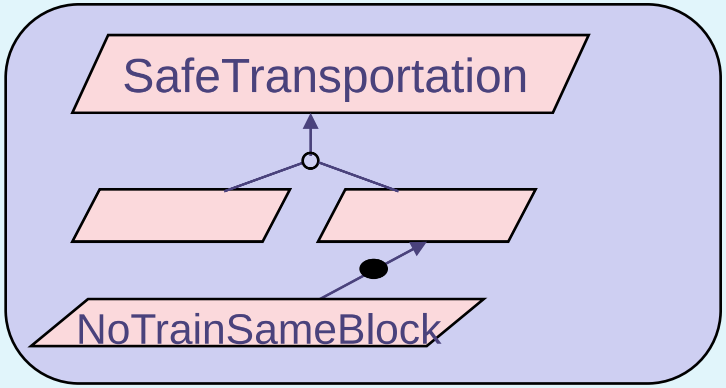


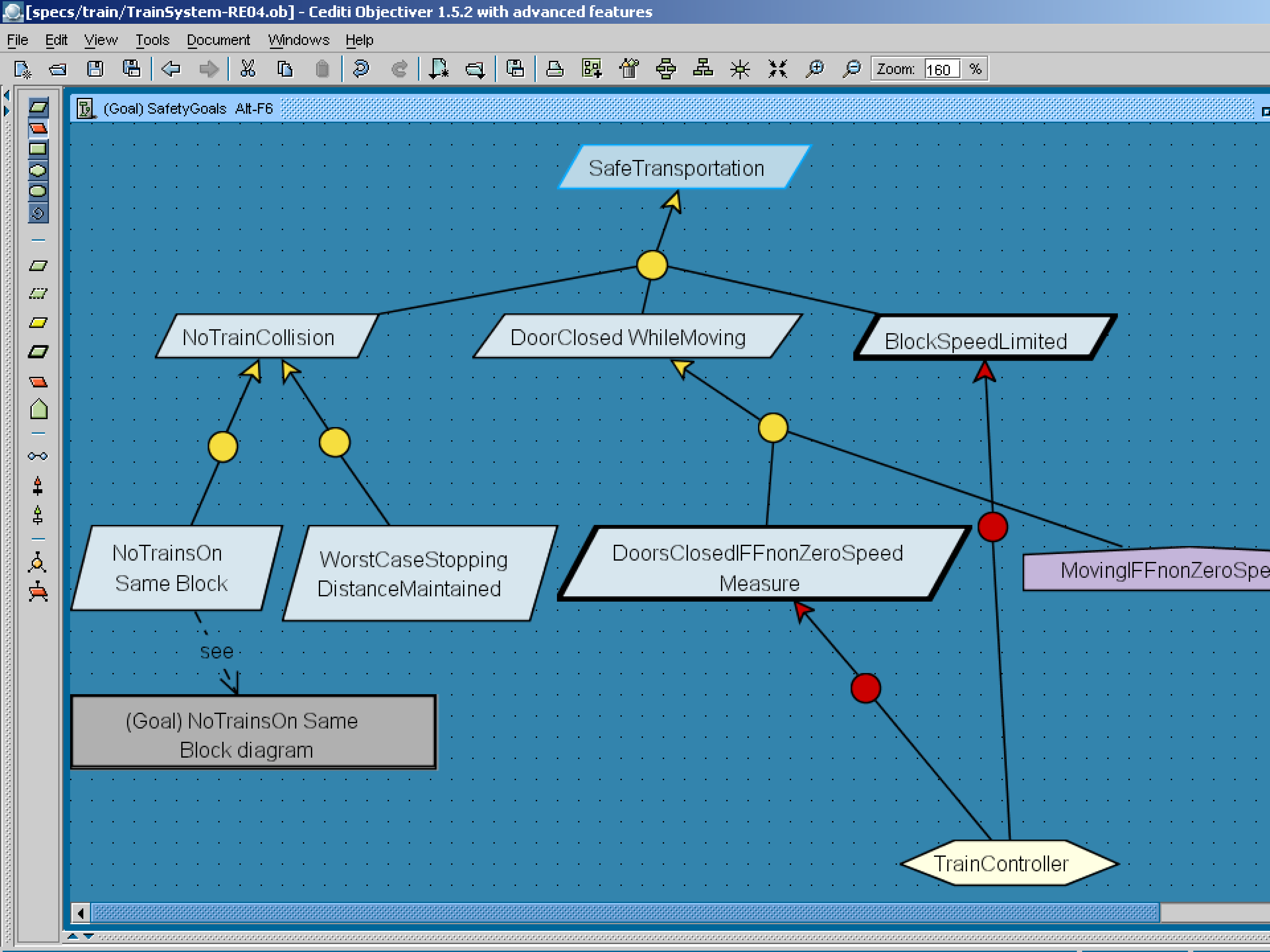
Course outline

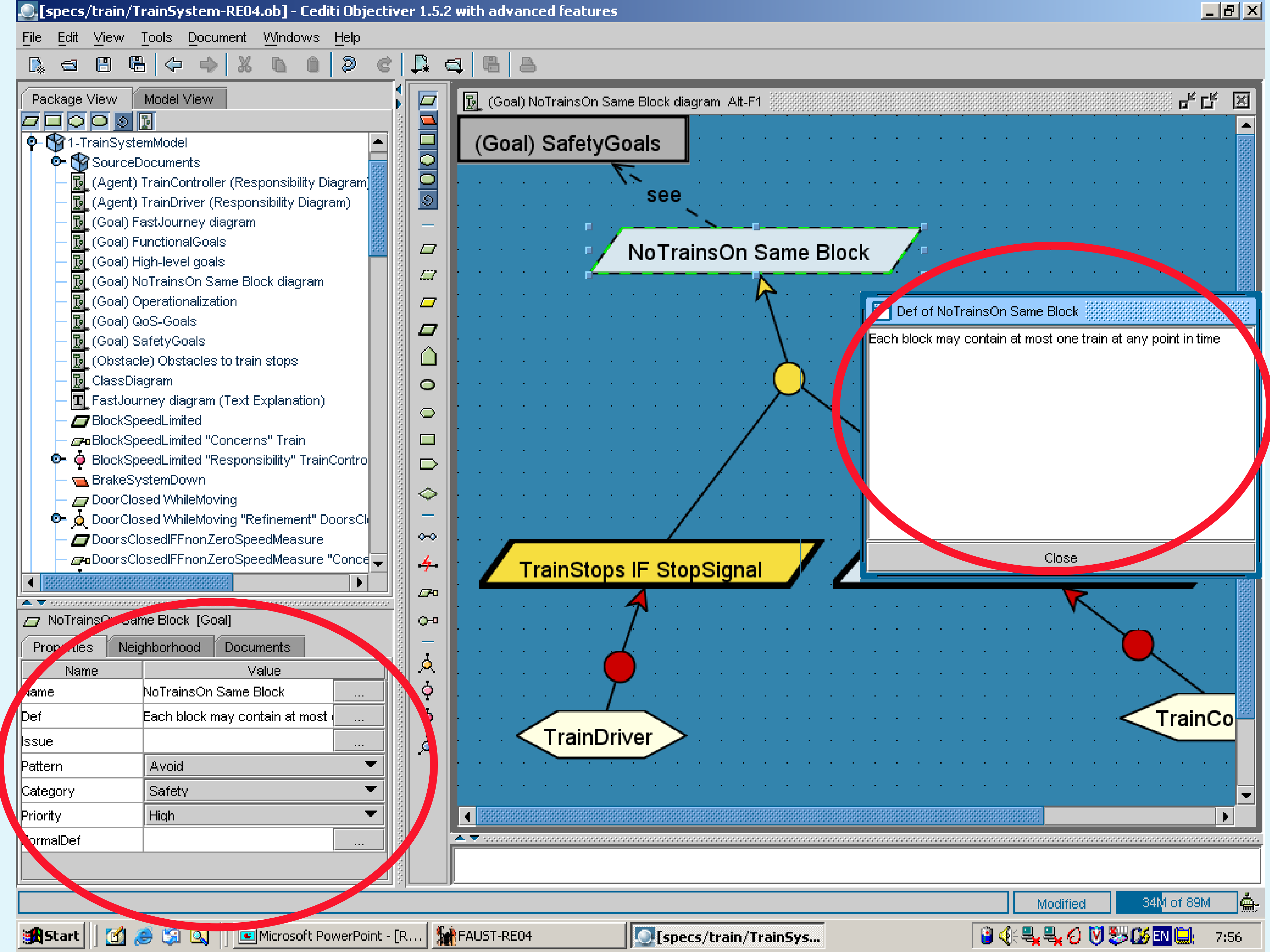
- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  -  A goal-oriented model building method in action
  - Obstacle analysis for high assurance
  - Formal reasoning about models
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals
  - Model checking against confidentiality requirements

Model building in KAOS

1. Domain analysis: refine/abstract goals




[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
Zoom: 160 %
(Goal) SafetyGoals Alt-F6
SafeTransportation
NoTrainCollision
DoorClosed WhileMoving
BlockSpeedLimited
NoTrainsOn Same Block
WorstCaseStopping DistanceMaintained
DoorsClosedIFFnonZeroSpeed Measure
MovingIFFnonZeroSpe
see
(Goal) NoTrainsOn Same Block diagram
TrainController


[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
Package View
Model View
1-TrainSystemModel
SourceDocuments
(Agent) TrainController (Responsibility Diagram)
(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" TrainContro
BrakeSystemDown
DoorClosed WhileMoving
DoorClosed WhileMoving "Refinement" DoorsCl
DoorsClosedIFFnonZeroSpeedMeasure
DoorsClosedIFFnonZeroSpeedMeasure "Conce
Same Block [Goal]
Neighborhood
Documents
Name
Value
NoTrainsOn Same Block
Def
Each block may contain at most
Issue
Pattern
Avoid
Category
Safety
Priority
High
(Goal) NoTrainsOn Same Block diagram Alt-F1
(Goal) SafetyGoals
see
NoTrainsOn Same Block
Def of NoTrainsOn Same Block
Each block may contain at most one train at any point in time
Close
TrainStops IF StopSignal
TrainDriver
TrainCo
Modified
34M of 89M
Start
Microsoft PowerPoint - [R...
FAUST-RE04
[specs/train/TrainSys...
EN
7:56

Building a goal model: heuristics & tips

- Early discovery of goals ...
  - Analysis of system-as-is
    - ⇒ problems, deficiencies, technology opportunities
    - ⇒ goals of S2B: Avoid / Reduce / Improve them
  - Search for *intentional & prescriptive* keywords in documents available, interview transcripts, etc.
    - in order to, so as to, so that, ...
    - has to, must, to be, must be, shall, ensure, want, motivate, expected to, ...
    - purpose, objective, aim, concern, ...

  refinement links: in order to *X* the system has to *Y*

Building a goal model: heuristics & tips (2)

- Later discovery of goals ...
  - by abstraction (bottom-up):

    asking WHY? questions about...

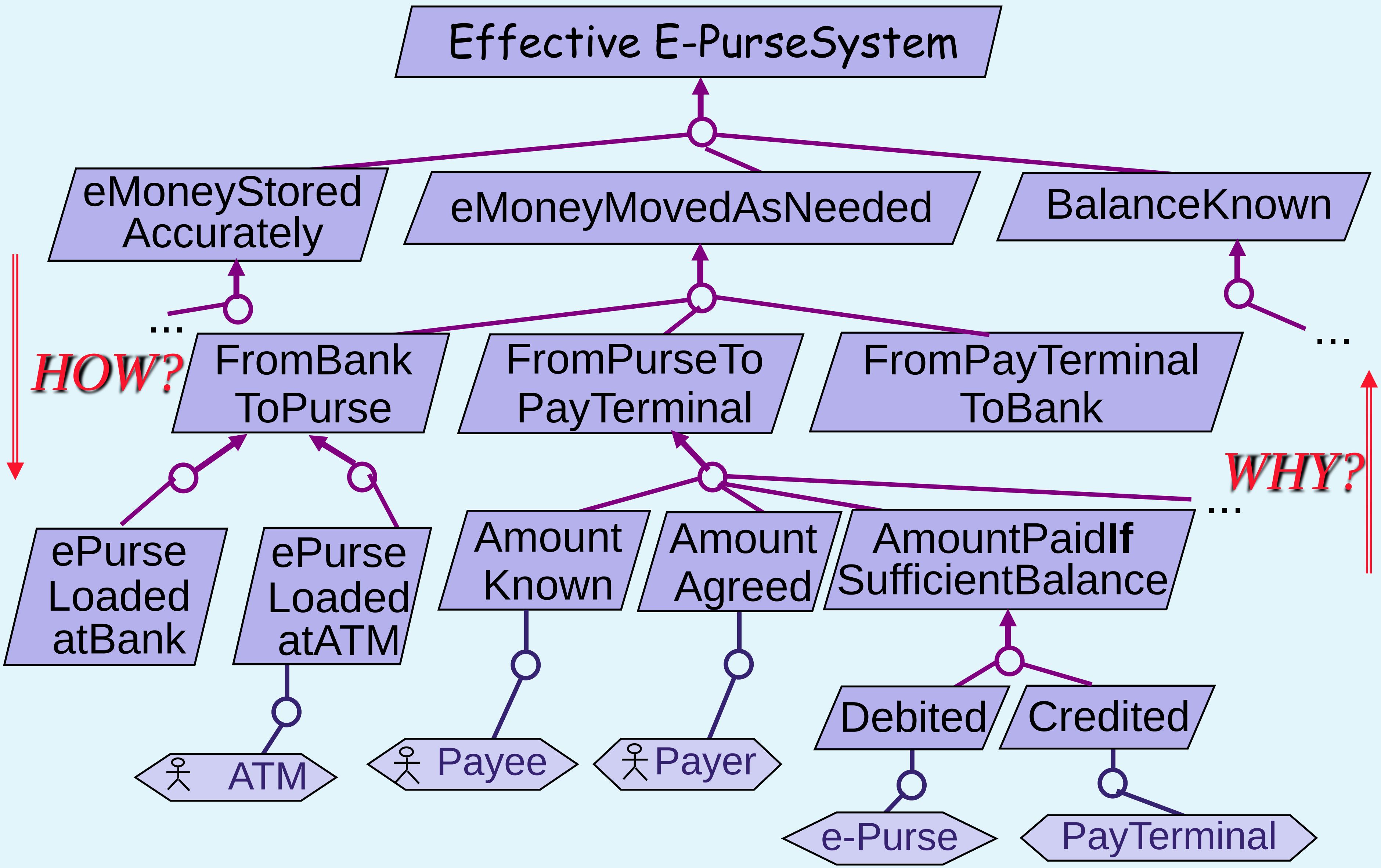
    lower-level goals

    interaction *scenarios*

    other operational material available
  - by refinement (top-down):

    asking HOW? questions about goals available
  - by use of refinement patterns (cf. below)
  - by resolution of obstacles, threats, conflicts (cf. below)

Building a goal model: HOW / WHY questions



Building a goal model: heuristics & tips (3)

- Abstract goals … until when ?

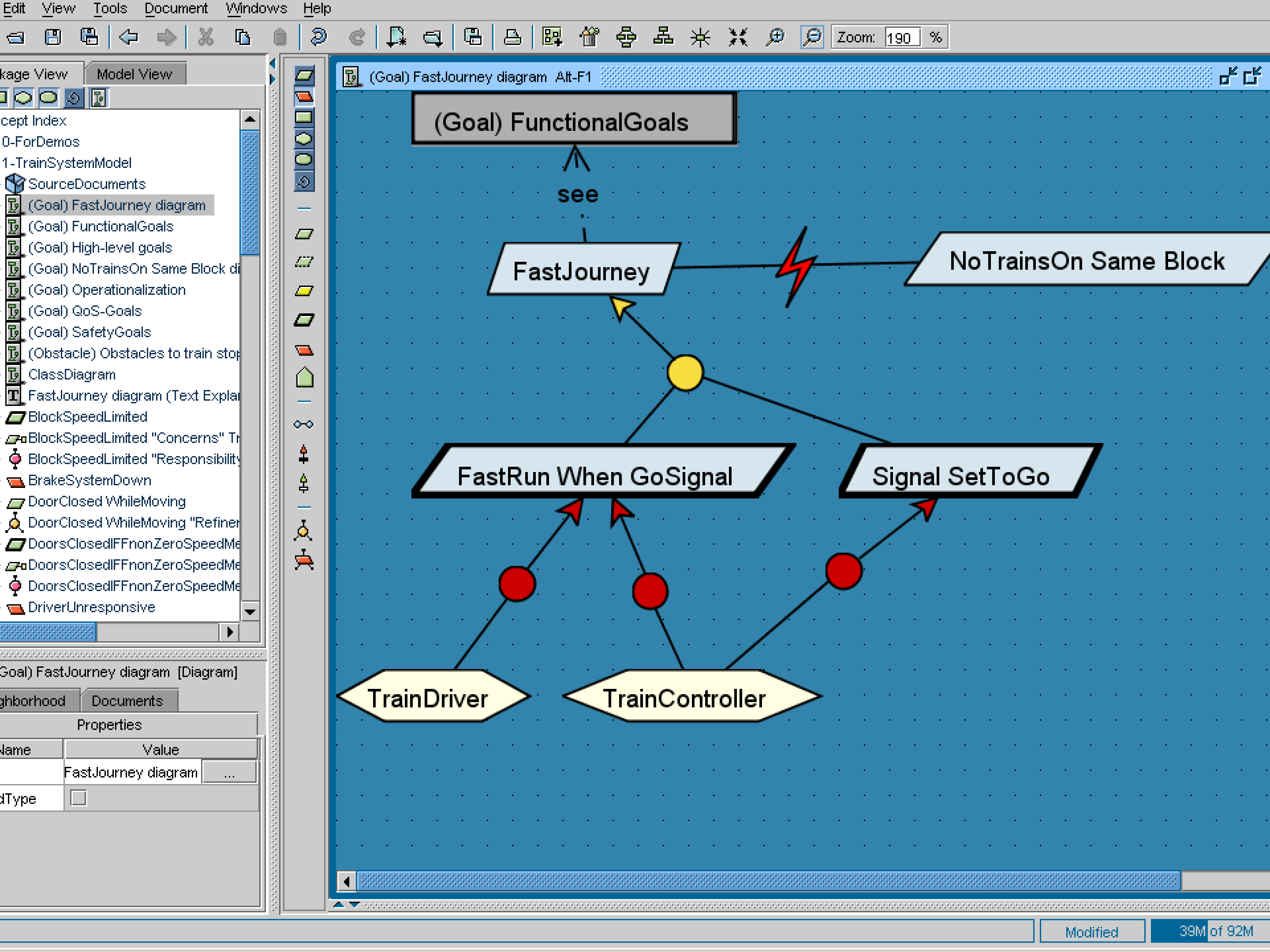
  … until boundary of system capabilities is reached

  e.g. *MakePeopleHappy* is beyond system's capabilities

- Refine goals … until when ?

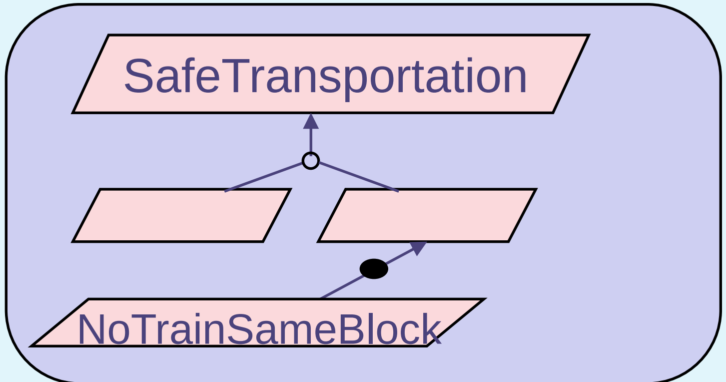
  … until assignable to single agents as …

  - requirement (software agent)
  - expectation (environment agent)

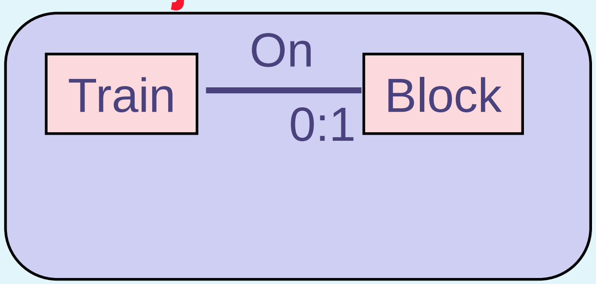

Edit View Tools Document Windows Help
Zoom: 190 %
kage View
Model View
cept Index
0-ForDemos
1-TrainSystemModel
SourceDocuments
(Goal) FastJourney diagram
(Goal) FunctionalGoals
(Goal) High-level goals
(Goal) NoTrainsOn Same Block di
(Goal) Operationalization
(Goal) QoS-Goals
(Goal) SafetyGoals
(Obstacle) Obstacles to train stop
ClassDiagram
FastJourney diagram (Text Explar
BlockSpeedLimited
BlockSpeedLimited "Concerns" Tr
BlockSpeedLimited "Responsibility
BrakeSystemDown
DoorClosed WhileMoving
DoorClosed WhileMoving "Refinem
DoorsClosedIFFnonZeroSpeedMe
DoorsClosedIFFnonZeroSpeedMe
DoorsClosedIFFnonZeroSpeedMe
DriverUnresponsive
Goal) FastJourney diagram [Diagram]
ghborhood
Documents
Properties
Name
Value
FastJourney diagram
...
dType
(Goal) FastJourney diagram Alt-F1
(Goal) FunctionalGoals
see
FastJourney
NoTrainsOn Same Block
FastRun When GoSignal
Signal SetToGo
TrainDriver
TrainController
Modified
39M of 92M

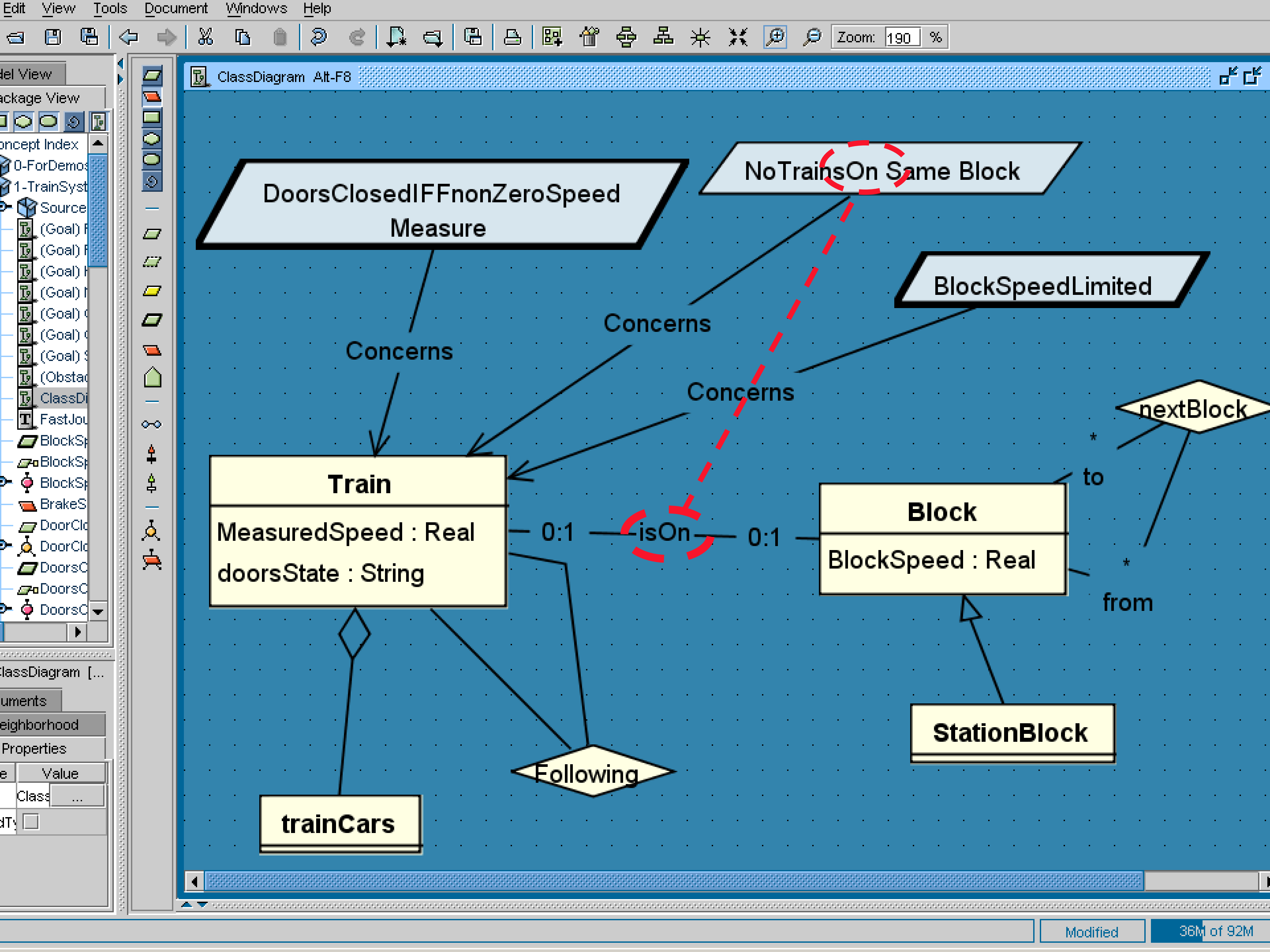
Model building in KAOS

1. Domain analysis: refine/abstract goals



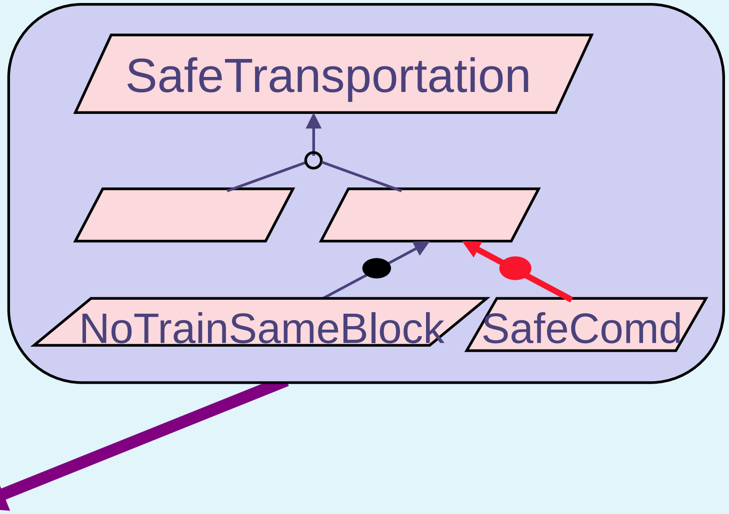
2. Domain analysis: derive/structure objects




Edit View Tools Document Windows Help
Zoom: 190 %
ClassDiagram Alt-F8
DoorsClosedIFFnonZeroSpeed Measure
NoTrainsOn Same Block
BlockSpeedLimited
Concerns
Concerns
Concerns
Train
MeasuredSpeed : Real
doorsState : String
0:1
isOn
0:1
Block
BlockSpeed : Real
nextBlock
*
to
*
from
StationBlock
trainCars
Following
Modified
36M of 92M

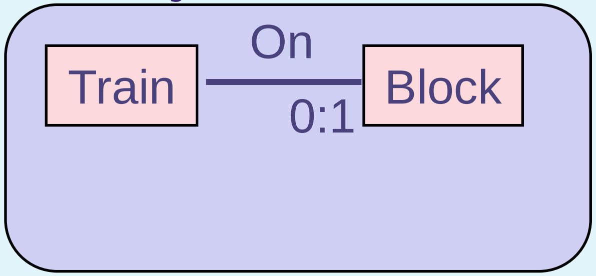
Model building in KAOS

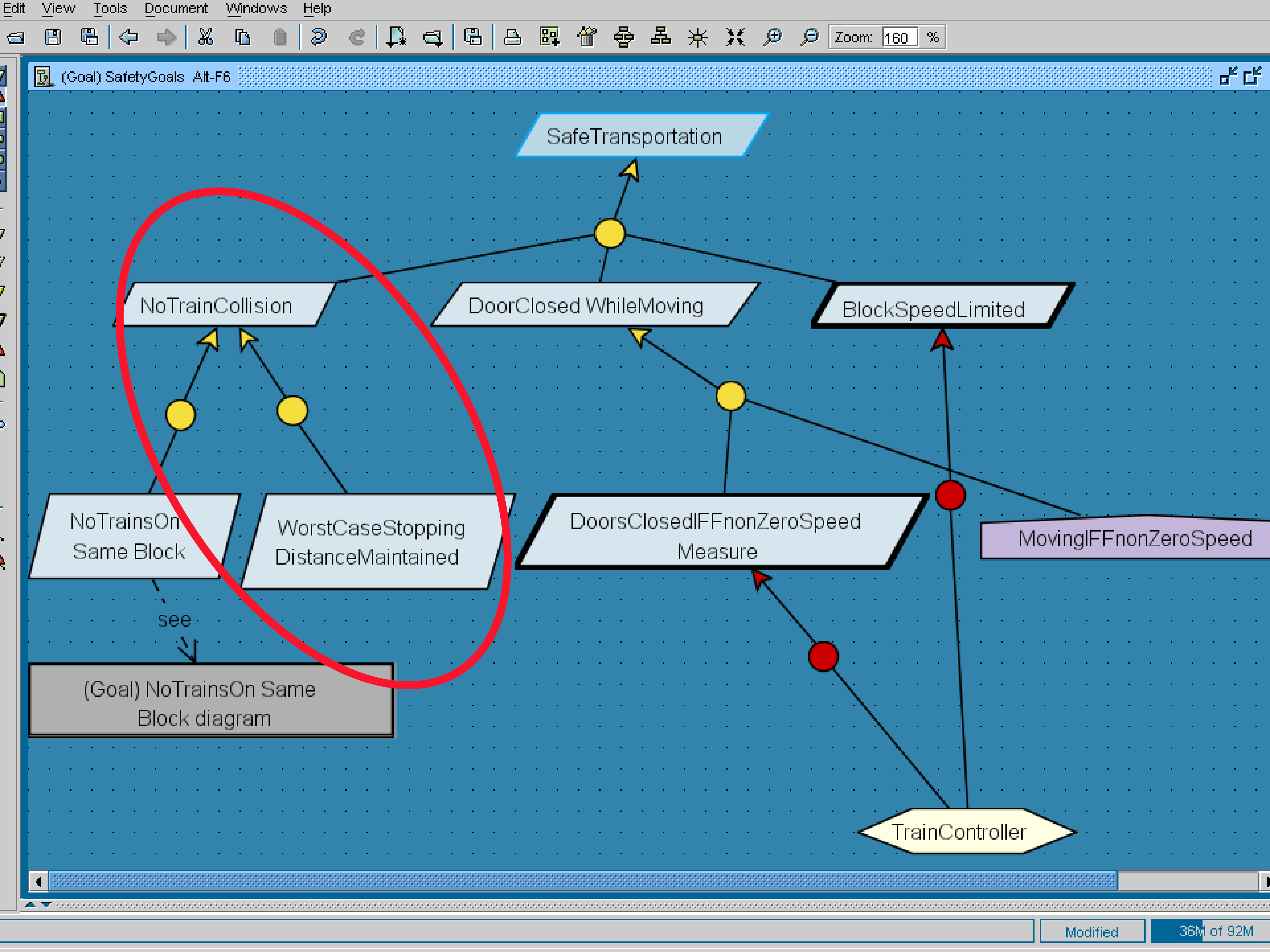
1. Domain analysis:
refine/abstract goals



3. S2B analysis:
enriched goals
(alternatives)

2. Domain analysis:
derive/structure
objects

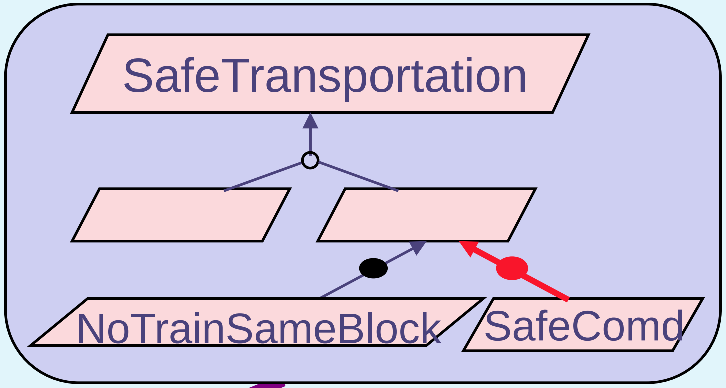




Edit View Tools Document Windows Help
Zoom: 160 %
(Goal) SafetyGoals Alt-F6
SafeTransportation
NoTrainCollision
DoorClosed WhileMoving
BlockSpeedLimited
NoTrainsOn Same Block
WorstCaseStopping DistanceMaintained
DoorsClosedIFFnonZeroSpeed Measure
MovingIFFnonZeroSpeed
see
(Goal) NoTrainsOn Same Block diagram
TrainController
Modified
36M of 92M

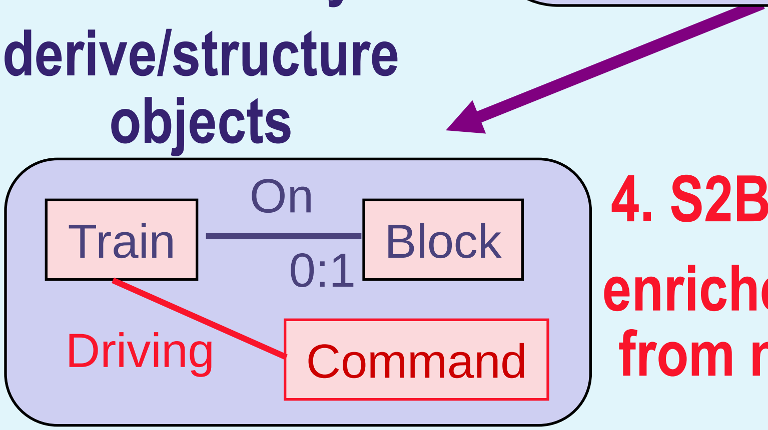
Model building in KAOS

1. Domain analysis:
refine/abstract goals



3. S2B analysis:
enriched goals
(alternatives)

2. Domain analysis:
derive/structure
objects



4. S2B analysis:
enriched objects
from new goals

The object model is derivable from the goal model

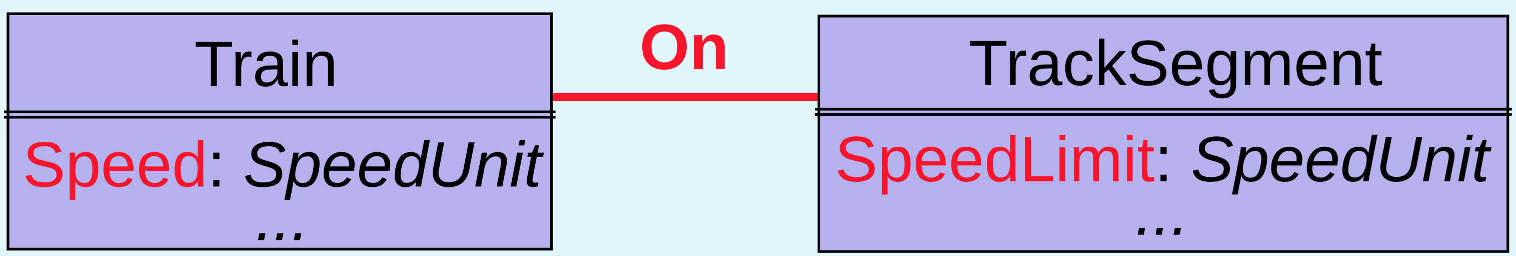
Goal Maintain [BlockSpeedLimited]

InformalDef *A Train should stay below the max speed the block can handle*

FormalDef $\forall$ tr: Train, ts: TrackSegment

On (tr, ts) $\Rightarrow$ tr.Speed $\leq$ ts.SpeedLimit

$\Downarrow$



Systematic, no "hocus pocus" (confessed by UML gurus)

$\Rightarrow$ completeness & pertinence of object model

Object model derivation: more formally ... (2)

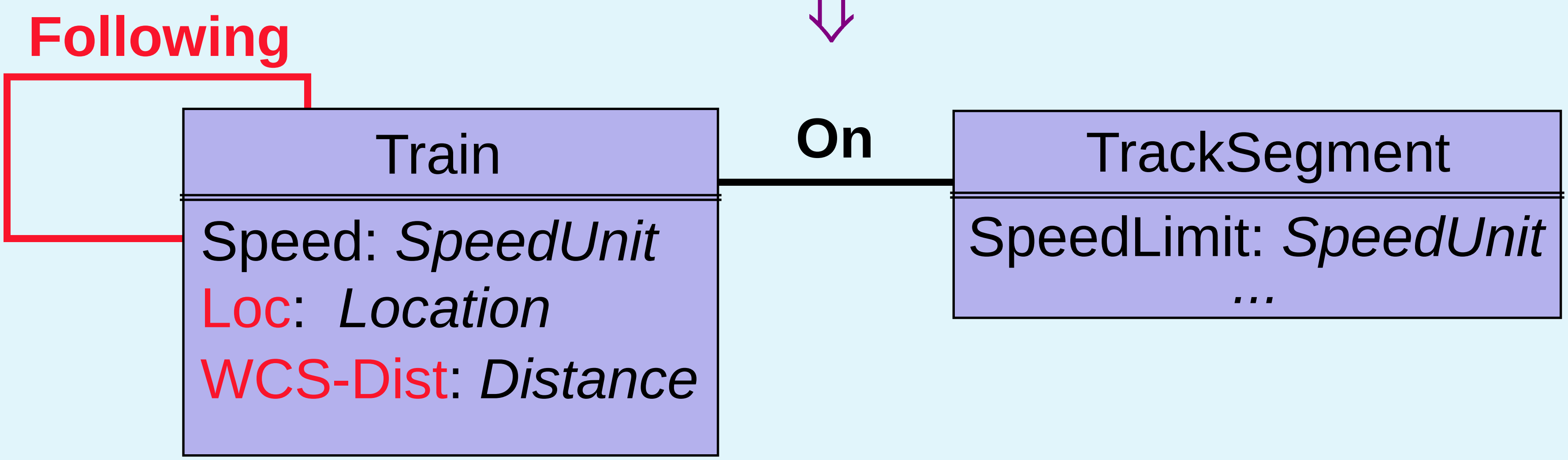
Goal Maintain [WC-SafeDistanceBetwTrains]

InformalDef *A Train should stay sufficiently far to avoid hitting the train in front in case of sudden stop*

FormalDef $\forall$ tr1, tr2: Train

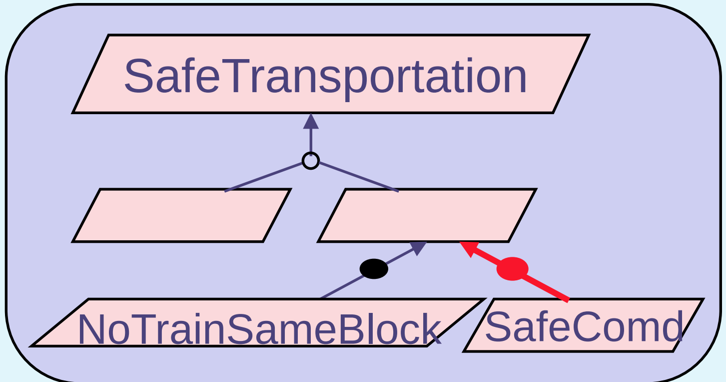
Following (tr1, tr2) $\Rightarrow$ tr1.Loc - tr2.Loc > tr1.WCS-Dist

$\Downarrow$



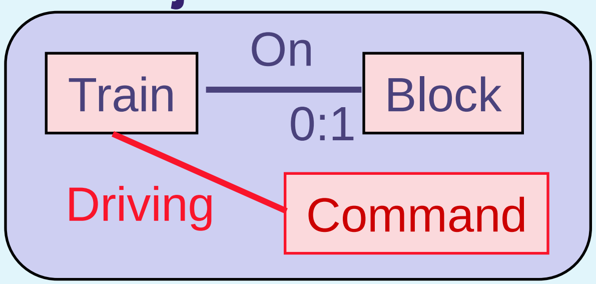
Model building in KAOS

1. Domain analysis:
refine/abstract goals

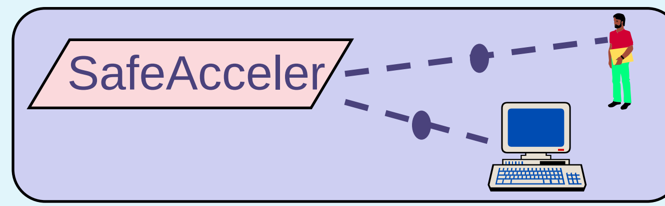


3. S2B analysis:
enriched goals (alternatives)

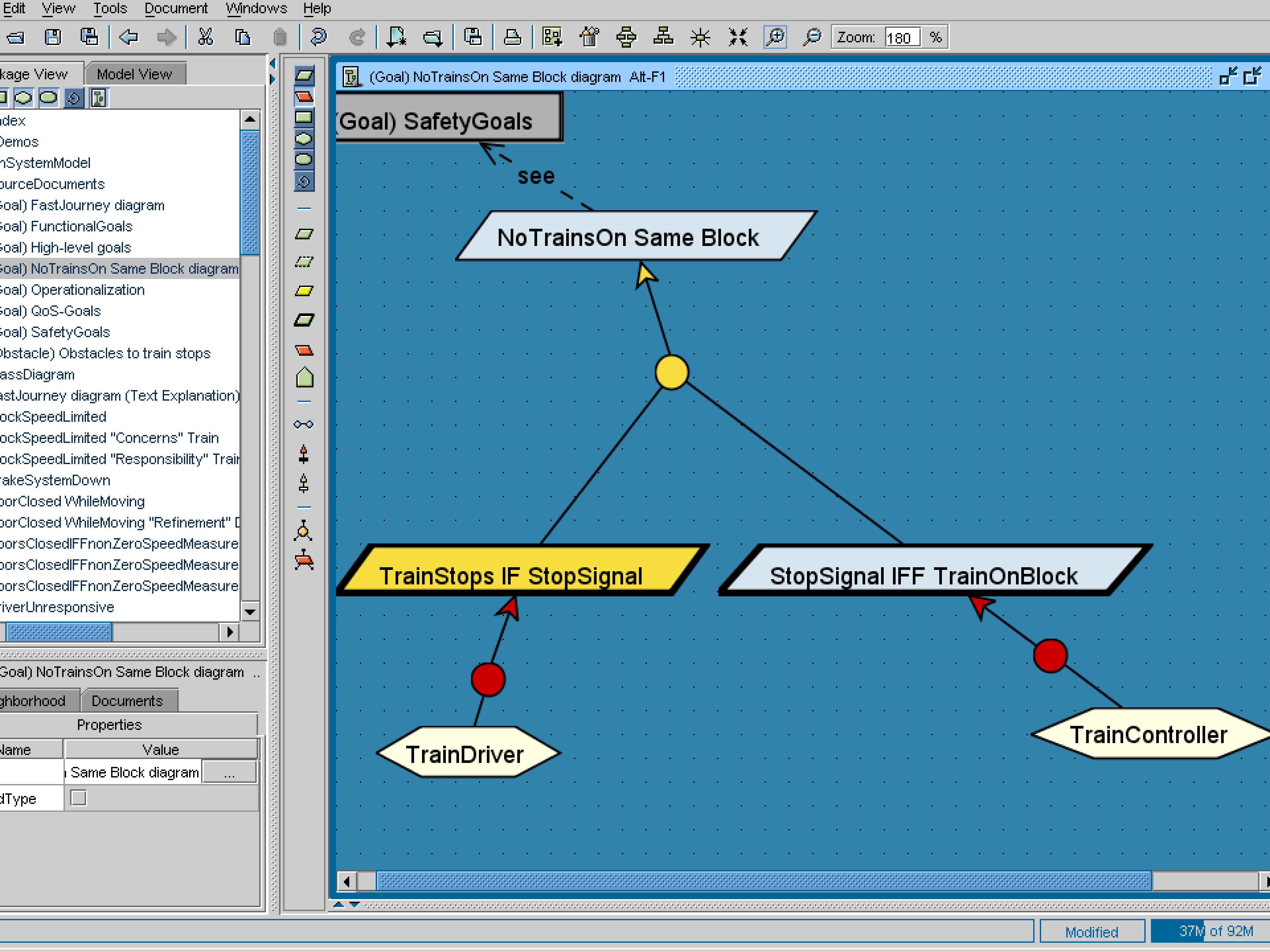
2. Domain analysis:
derive/structure objects



4. S2B analysis:
enriched objects from new goals



5. Responsibility analysis:
agent OR-assignment



Edit View Tools Document Windows Help
Zoom: 180 %
kage View
Model View
(Goal) NoTrainsOn Same Block diagram Alt-F1
dex
Demos
nSystemModel
ourceDocuments
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
assDiagram
astJourney diagram (Text Explanation)
ockSpeedLimited
ockSpeedLimited "Concerns" Train
ockSpeedLimited "Responsibility" Trai
rakeSystemDown
oorClosed WhileMoving
oorClosed WhileMoving "Refinement" D
oorsClosedIFFnonZeroSpeedMeasure
oorsClosedIFFnonZeroSpeedMeasure
oorsClosedIFFnonZeroSpeedMeasure
riverUnresponsive
Goal) NoTrainsOn Same Block diagram ..
ghborhood
Documents
Properties
Name
Value
Same Block diagram
...
dType
(Goal) SafetyGoals
see
NoTrainsOn Same Block
TrainStops IF StopSignal
StopSignal IFF TrainOnBlock
TrainDriver
TrainController
Modified
37M of 92M

Context diagrams can be derived from goals

Many behavioral goals take the form

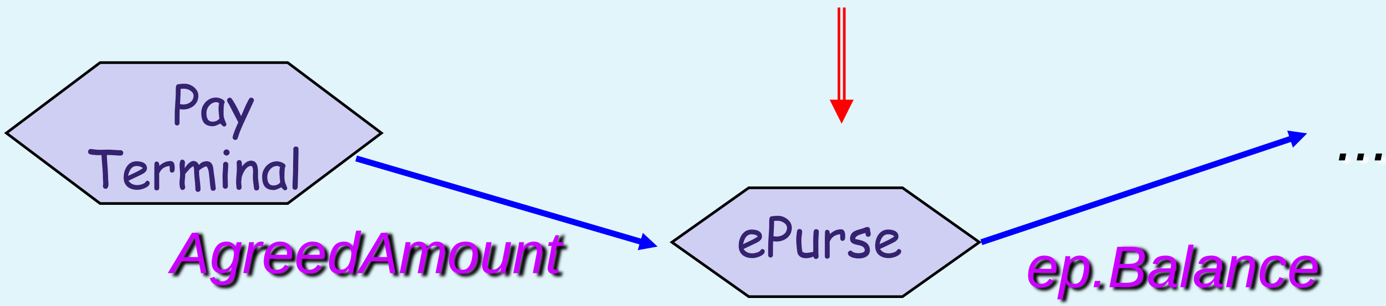
G: CurrentCondition [monitoredVariables]

$\Rightarrow$ [sooner-or-later/always]

TargetCondition [controlledVariables]

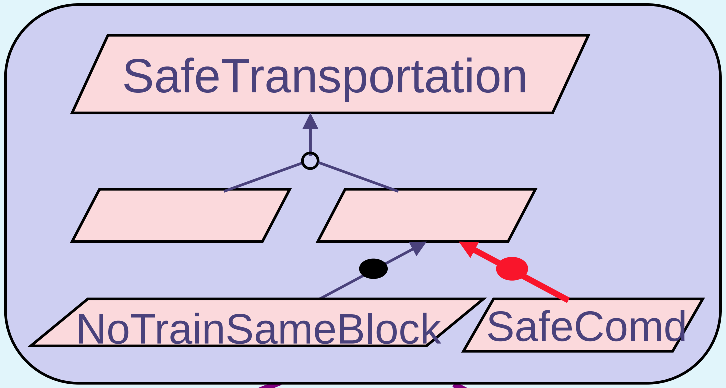
AgreedAmount $\leq$ ep.Balance

$\Rightarrow$ $\circ$ ep.Balance = ep.Balance - AgreedAmount



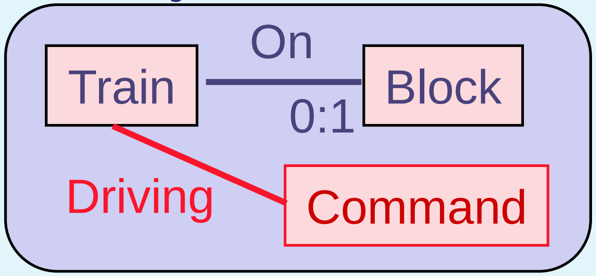
Model building in KAOS

1. Domain analysis: refine/abstract goals

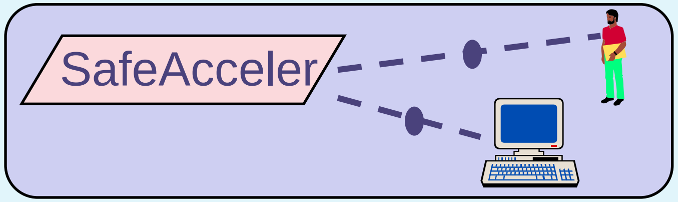


3. S2B analysis: enriched goals (alternatives)

2. Domain analysis: derive/structure objects

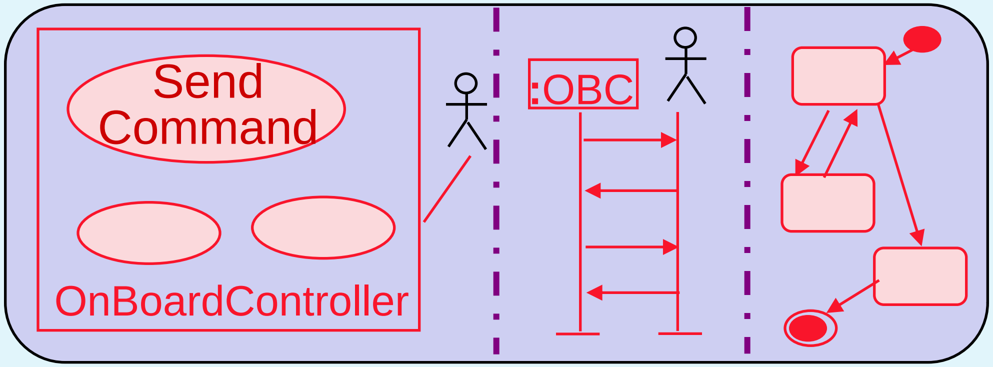


1-5. Obstacle & conflict analysis

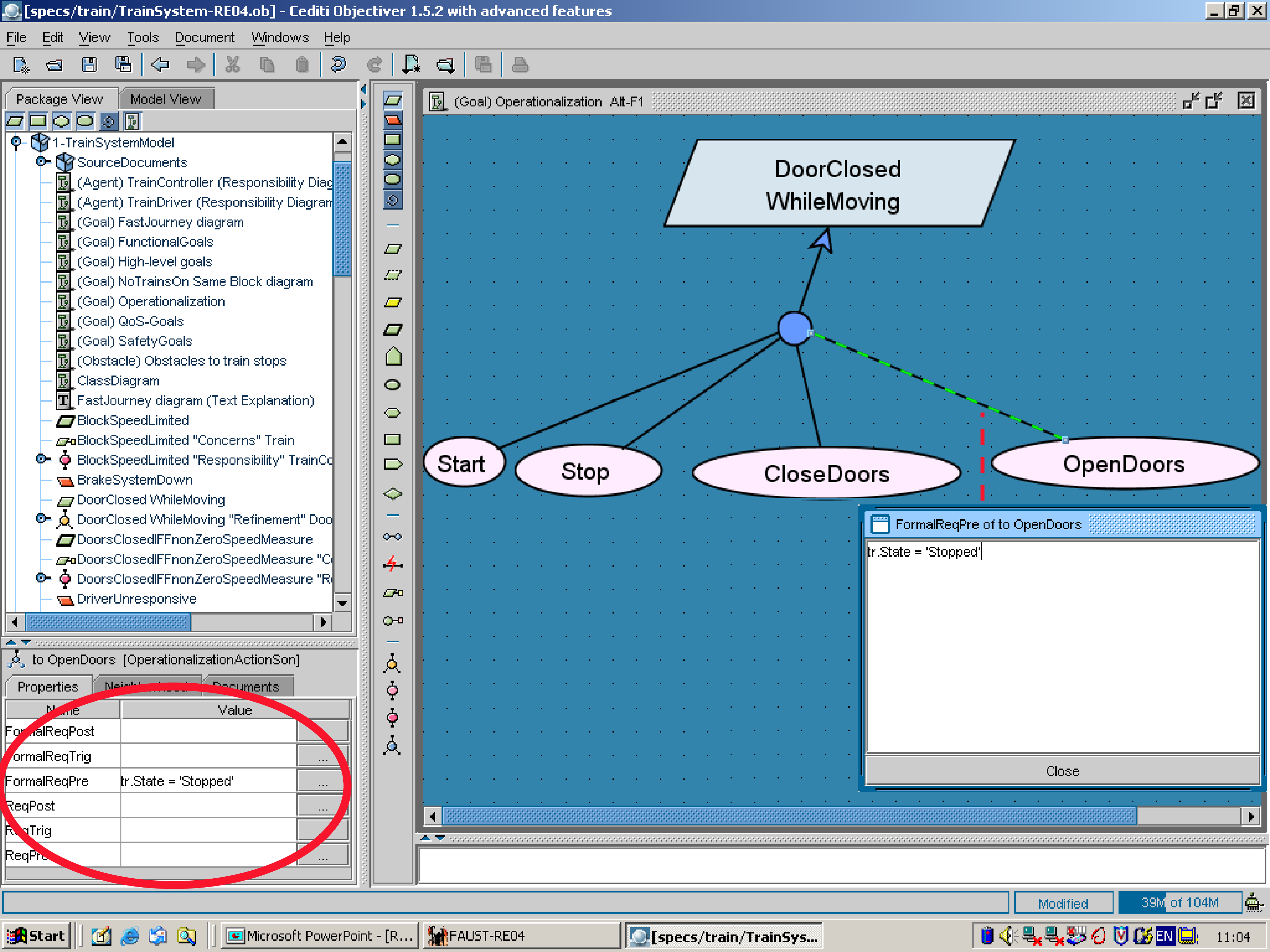


4. S2B analysis: enriched objects from new goals

5. Responsibility analysis: agent OR-assignment

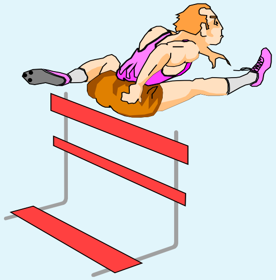


6. Operationalization & behavior analysis


[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
Package View
Model View
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
Documents
Value
FormalReqPre
tr.State = 'Stopped'
(Goal) Operationalization Alt-F1
DoorClosed WhileMoving
Start
Stop
CloseDoors
OpenDoors
FormalReqPre of to OpenDoors
tr.State = 'Stopped'
Close
Modified
39M of 104M
Start
Microsoft PowerPoint - [R...
FAUST-RE04
[specs/train/TrainSys...
EN
11:04

Course outline

- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  - A goal-oriented model building method in action
  - Obstacle analysis for high assurance
  - Formal reasoning about models
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals
  - Model checking against confidentiality requirements



Modeling what could go wrong: obstacle analysis

- Problem: goals are often too ideal, will be violated
  (unexpected or malicious agent behaviors)
- Obstacle = condition on system for goal violation

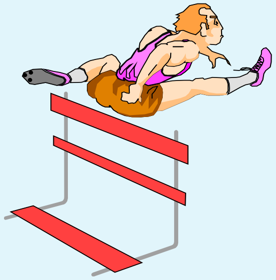
  { *O*, Dom } |= ¬ *G* — *obstruction*

  Dom |≠ ¬ *O* — *domain consistency*

  exists system behavior *S* s.t. *S* |= *O* — *feasibility*

- Particular cases

  obstruction of safety goal: obstacle = **hazard**
  obstruction of security goal: obstacle = **threat**



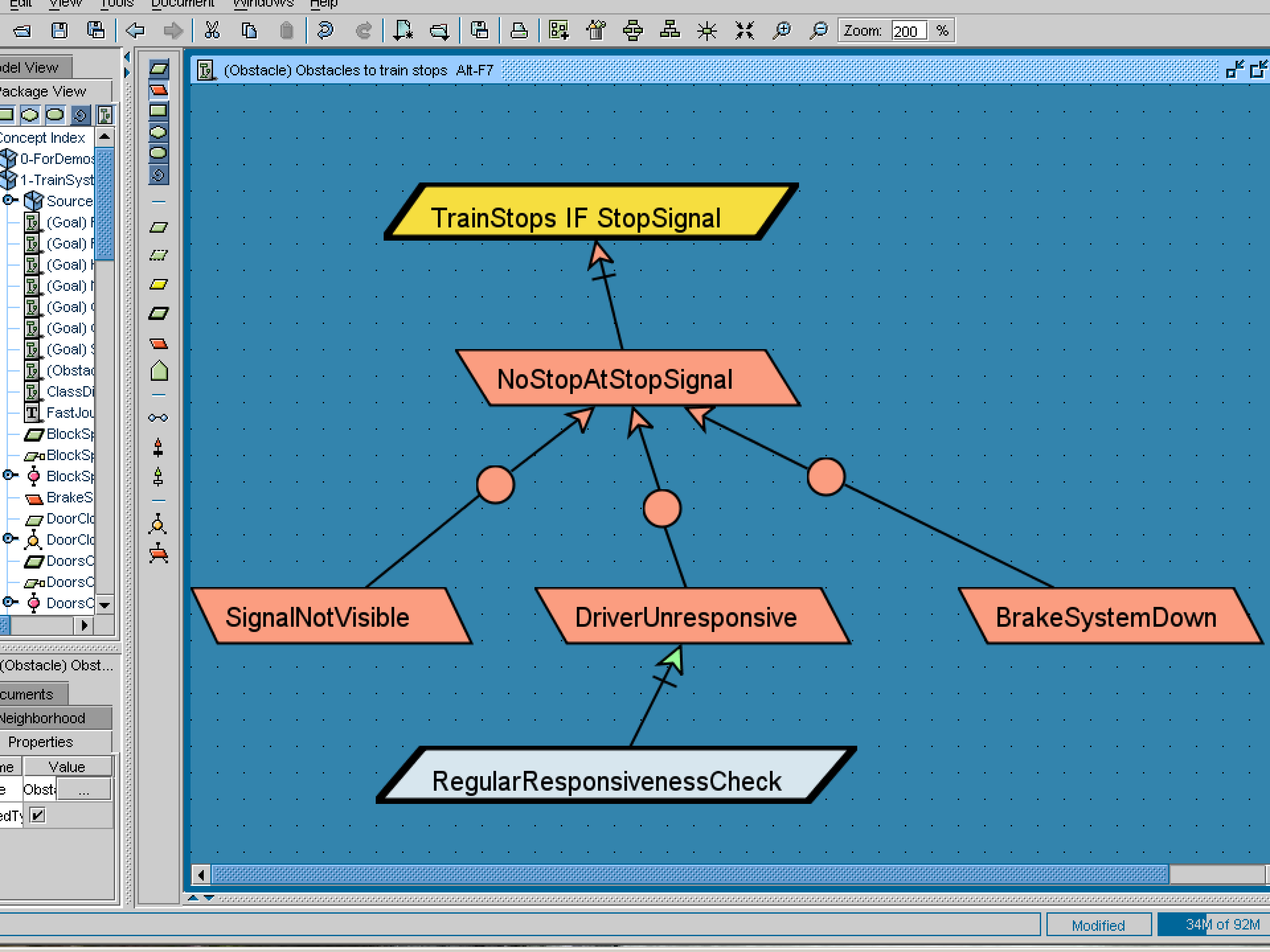
Obstacle analysis for increased reliability & security

- Anticipate obstacles ...

  ⇒ new goals (countermeasures), deidealized model

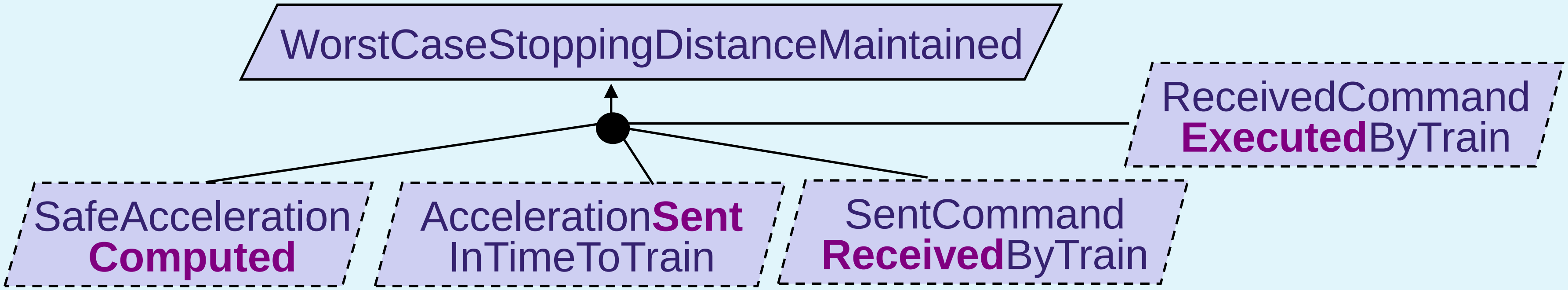
  ⇒ more complete, realistic requirements

  ⇒ more robust system

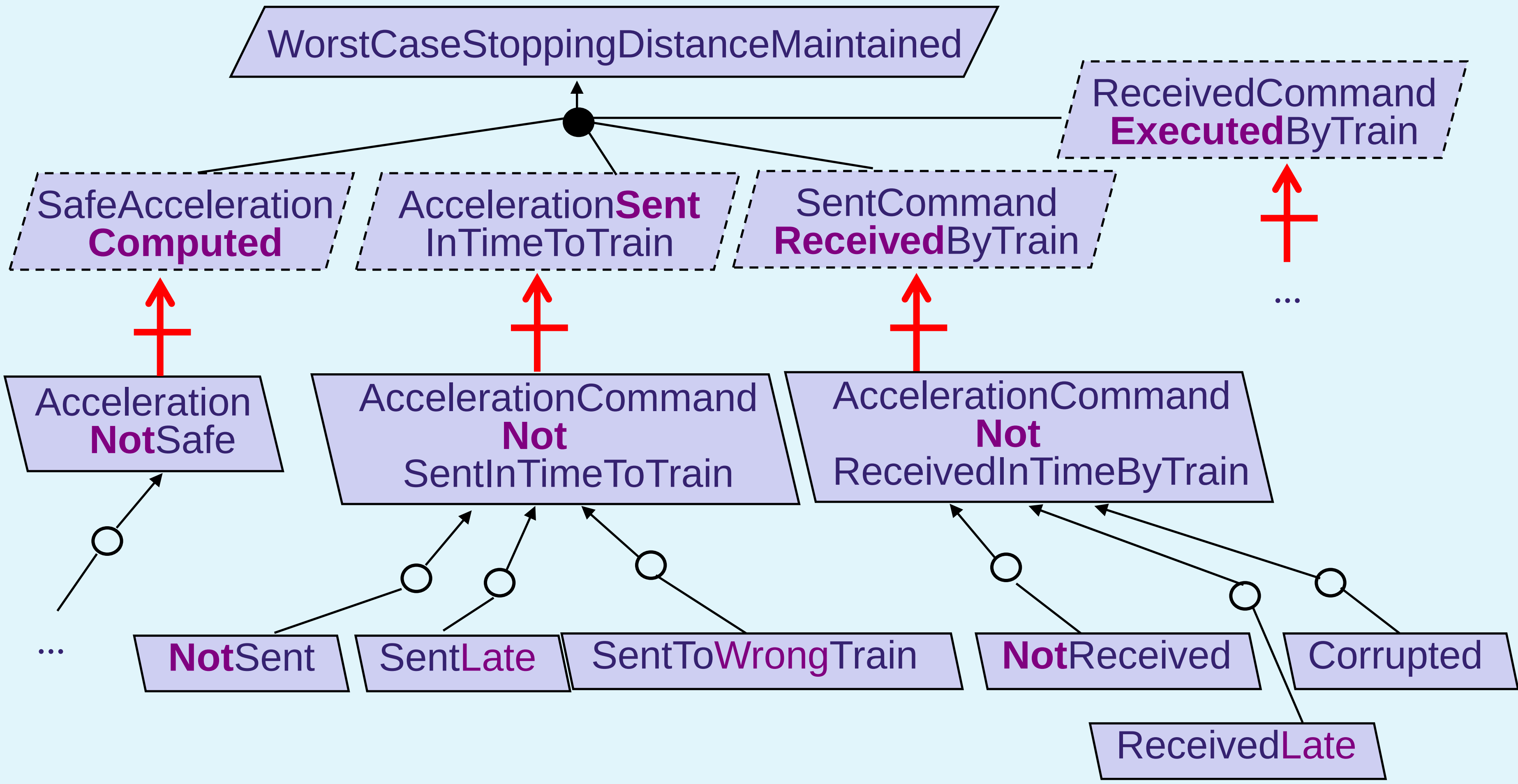


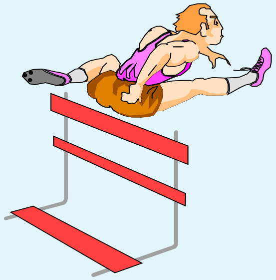
(Obstacle) Obstacles to train stops Alt-F7
TrainStops IF StopSignal
NoStopAtStopSignal
SignalNotVisible
DriverUnresponsive
BrakeSystemDown
RegularResponsivenessCheck
Zoom: 200 %
Package View
Concept Index
Properties
Value
Modified
34M of 92M

Obstacle models as goal-anchored fault trees



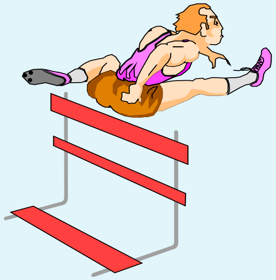
Obstacle models as goal-anchored fault trees





Obstacle analysis

- For every leaf goal in refinement graph (requirement or expectation):
  - **identify** as many obstacles to it as possible
  - **assess** their likelihood & severity
  - **resolve** them according to likelihood/severity

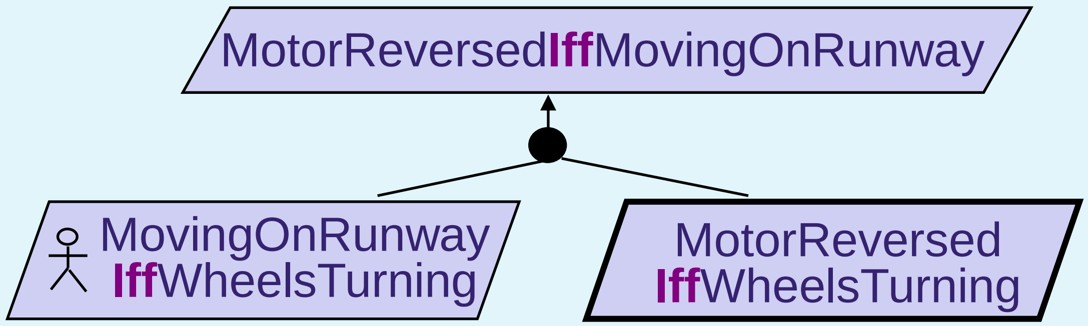


Obstacle identification

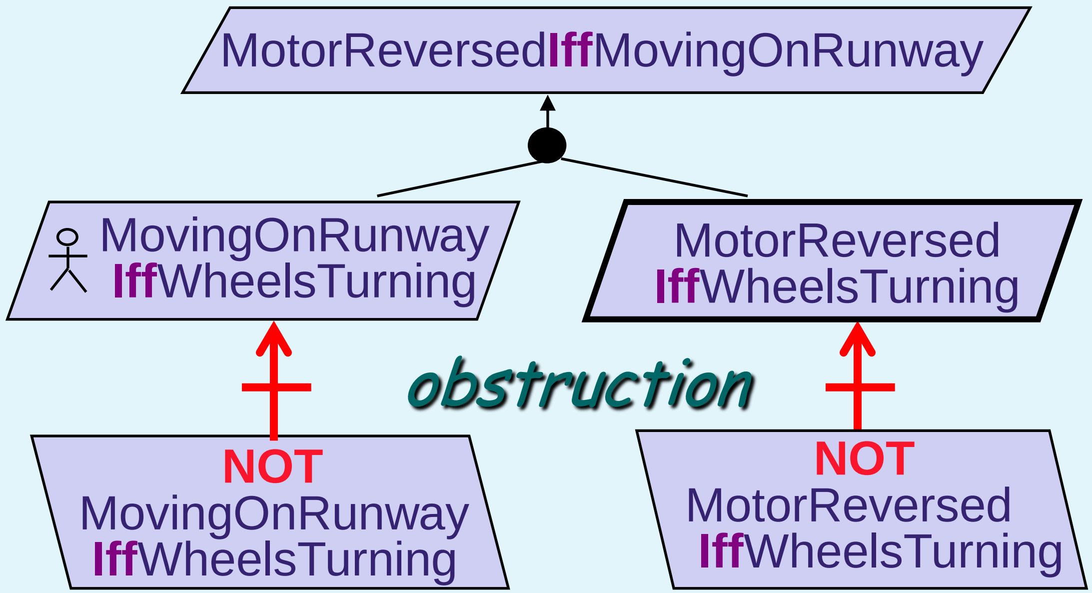
- For obstacle to goal *G*:
  - negate *G*;
  - find as many AND/OR refinements of ¬ *G* as possible in view of domain properties ...
  - ... until reaching obstruction preconditions that are *feasible* (through a system scenario)

= *goal-anchored* fault-tree construction

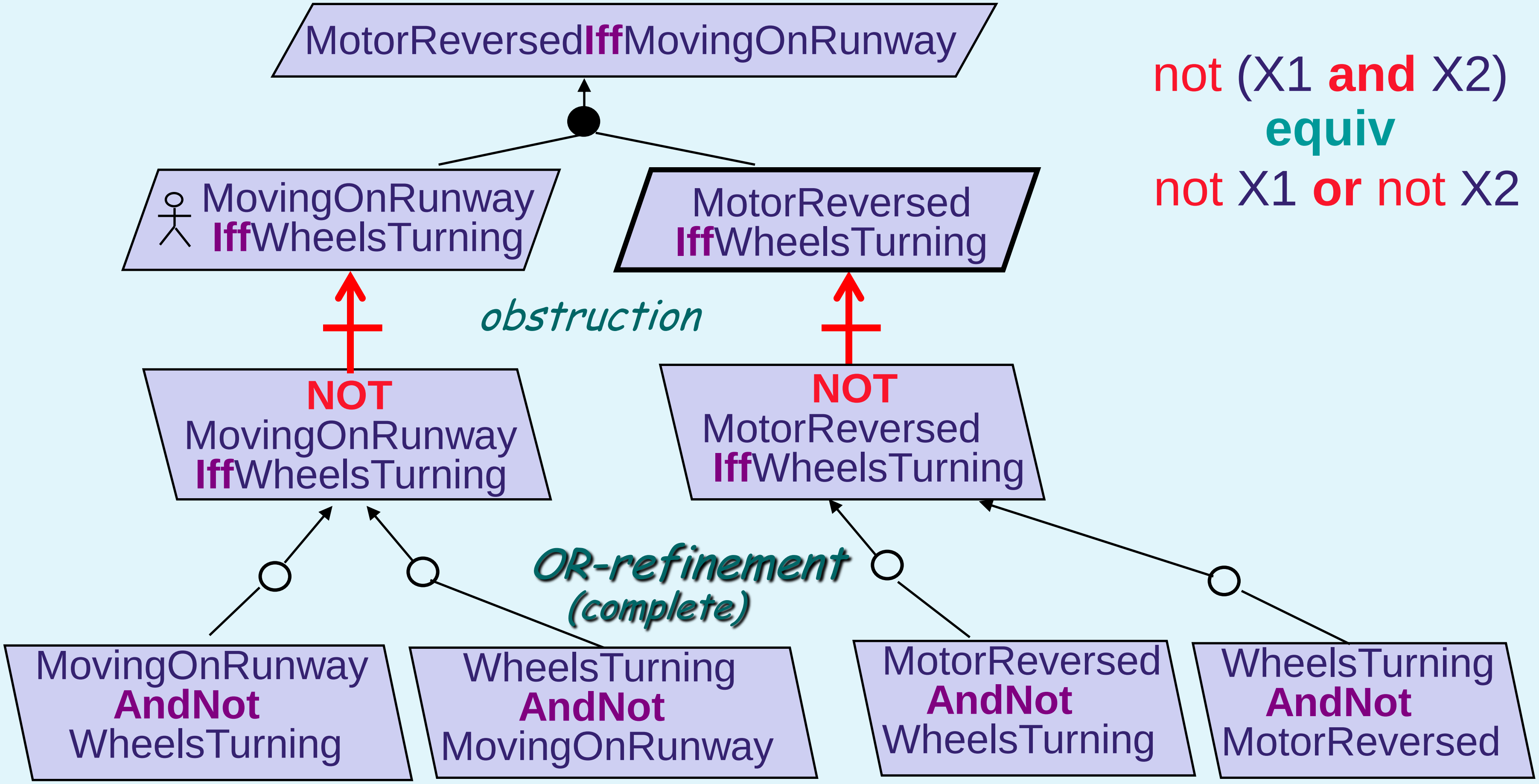
Obstacle identification: example



Obstacle identification: example

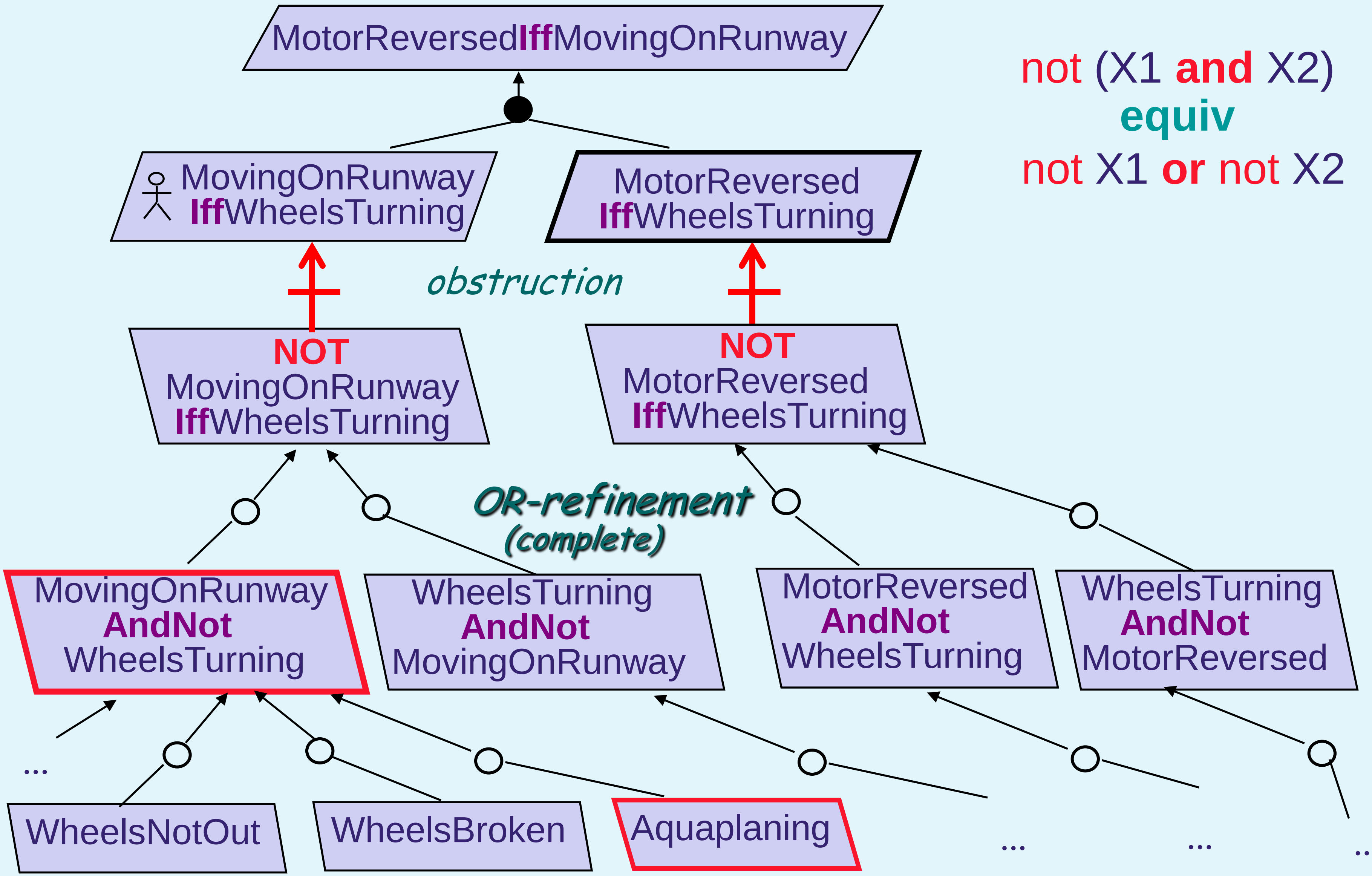


Obstacle identification: example



not (X1 **and** X2)
equiv
not X1 **or** not X2

Obstacle identification: example



Obstacle assessment & resolution

- To *assess* likelihood & severity of identified obstacle: cfr. risk management techniques
- To *resolve* identified obstacle:
  - *at RE time:* model transformation
    - generate *alternative* resolutions
    - select "best" resolution based on ...
      - likelihood/severity of obstacle
      - other non-functional/quality goals
  - *at run-time* (for non-severe, occasional obstacles): obstacle monitoring, run-time resolution

    (cf. specification-based intrusion detection)

Generating obstacle resolutions

- Use of *model transformation operators* encoding resolution tactics
  - Goal substitution: consider alternative refinement of parent goal to avoid obstruction of child goal

    MotorReversed **Iff** WheelsTurning

    → MotorReversed **Iff** *PlaneWeightSensed*
  - Agent substitution: consider altern. responsibilities

    OnBoardTrainController → *VitalStationComputer*
  - Goal weakening

    TrafficControllerOnDutyOnSector →

    TrafficControllerOnDutyOnSector ***or*** *WarningToNextSector*

Generating obstacle resolutions (2)

- Model transformation operators (cont'd):
  - Goal restoration: **enforce target condition at obstacle occurrence**

    ResourceNotReturnedInTime → *ReminderSent*

    WheelsNotOut → *WheelsAlarmGenerated*
  - Obstacle prevention: **new *Avoid* goal**

    AccelerationCommandCorrupted

    → *Avoid [*AccelerationCommandCorrupted*]*
  - Obstacle mitigation: **tolerate obstacle but mitigate its effects**

    OutdatedSpeed/PositionEstimates

    → Avoid [TrainCollision*WhenOutDatedTrainInfo*]

Course outline

- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  - A goal-oriented model building method in action
  - Obstacle analysis for high assurance
  -  Formal reasoning about models
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals
  - Model checking against confidentiality requirements

Formal reasoning about system models ...

- To support more accurate analysis & derivations
  - Checking refinements & operationalizations
  - Generating obstacles to goals
  - Generating attack graphs
  - Analyzing conflicts
  - Synthesizing behavior models from scenarios & goals
  - Goal-oriented model animation
- Optional "button": only *when* & *where* needed
- Requires formal specifications to annotate models

Some bits of real-time linear temporal logic

∘ P:	P shall hold in the next state
□ P:	P shall hold in every future state
P $\mathcal{W}$ N:	P shall hold in every future state *unless* N holds
◊ P:	P shall hold in some future state
$\Box_{\leq T}$ P:	P shall hold in every future state up to *T* time units
$\Diamond_{\leq T}$ P:	P shall hold within T time units

+ *past operators:* ● P, ■ P, ♦ P, ...

$P \Rightarrow Q$: $\Box (P \rightarrow Q)$

@ P : $\bullet (\neg P) \wedge P$

Specifying goals in RT-LTL

Goal *Maintain* [DoorsClosedUntilNextStation]

FormalSpec $\forall$ tr: Train, s: Station

$\text{At (tr, st)} \wedge \circ \neg \text{At (tr, st)} \Rightarrow$

$\text{tr.Doors = "closed"} \; \mathcal{W} \; \text{At (tr, next(st))}$

Goal *Achieve* [FastJourneyBetweenStations]

FormalSpec $\forall$ tr: Train, s: Station

$\text{At (tr, st)} \Rightarrow \Diamond_{\leq T} \text{At (tr, next(st))}$

Achieve P, Cease P

Maintain P, Avoid P : goal specification patterns

Goal-oriented spec of operations

Operation SendCommand

Input tr, tr ’: Train

Output cm: CommandMsg

DomPre ¬ Sent (cm, tr)

DomPost Sent (cm, tr)

ReqPost for *SafeCommandMsg*

Following (tr, tr’) →

cm.Accel ≤ F (tr, tr’) ∧ cm.Speed > G (tr)

ReqTrig for *CommandMsgSentInTime*

$\blacksquare_{\leq 0.5\ sec}\ \neg \exists$ cm': Sent (cm', tr)

Formal reasoning: refinement checking

- A set of assertions $\{A_1, \ldots, A_n\}$ correctly refines assertion *A* in domain theory *Dom* iff

  $\{A_1, \ldots, A_n, Dom\} \models A$ *completeness*

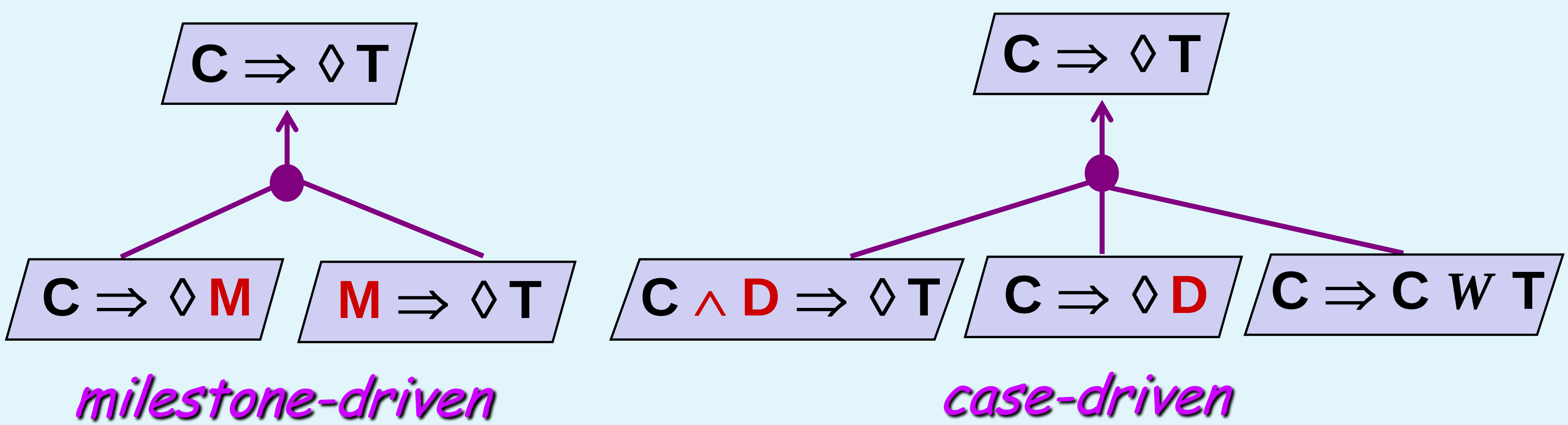
  $\{A_1, \ldots, A_n, Dom\} \not\models false$ *consistency*

  $\{\wedge_{j \neq i} A_j, Dom\} \not\models A$ for each $i \in [1..n]$ *minimality*

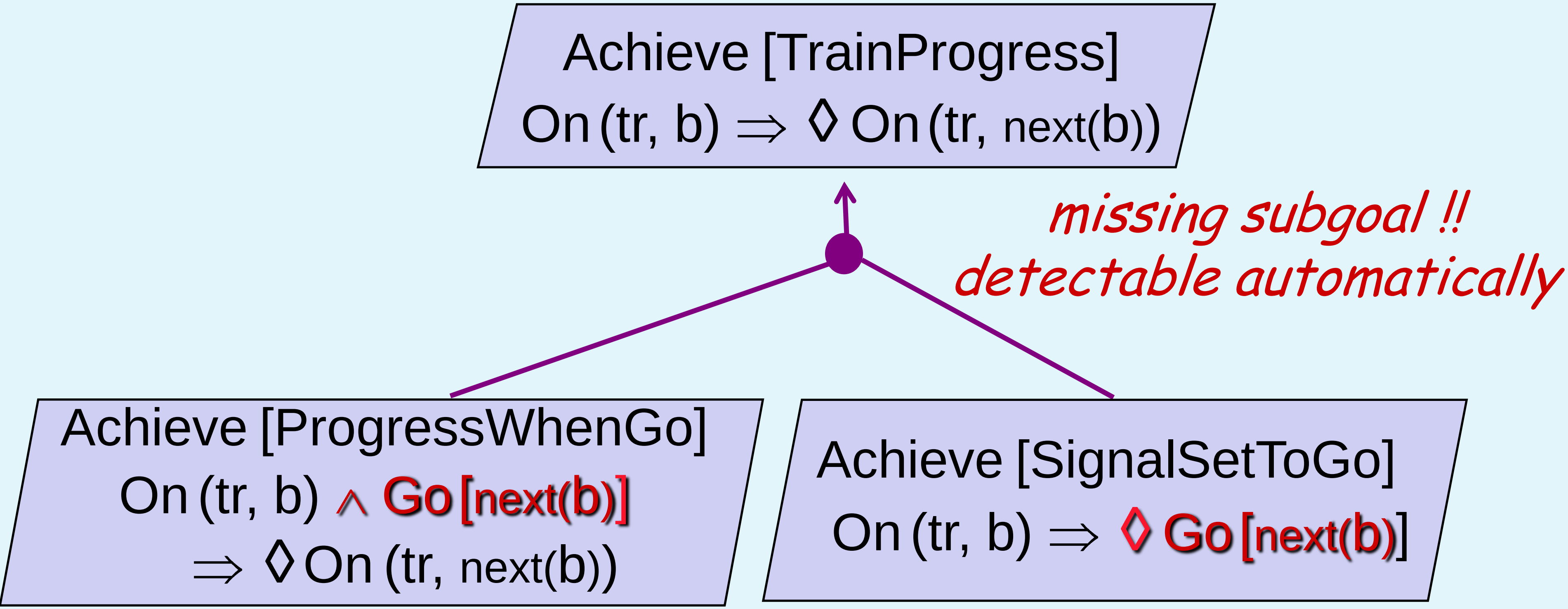
- Refinement checking =
  - Check that a refinement is correct
  - If not, suggest missing sub-assertions A_i
- Can be used for checking goal models, obstacle models, anti-goal models; and reveal missing subgoals, subobstacles, vulnerabilities (completeness is essential!)

Refinement checking: using refinement patterns

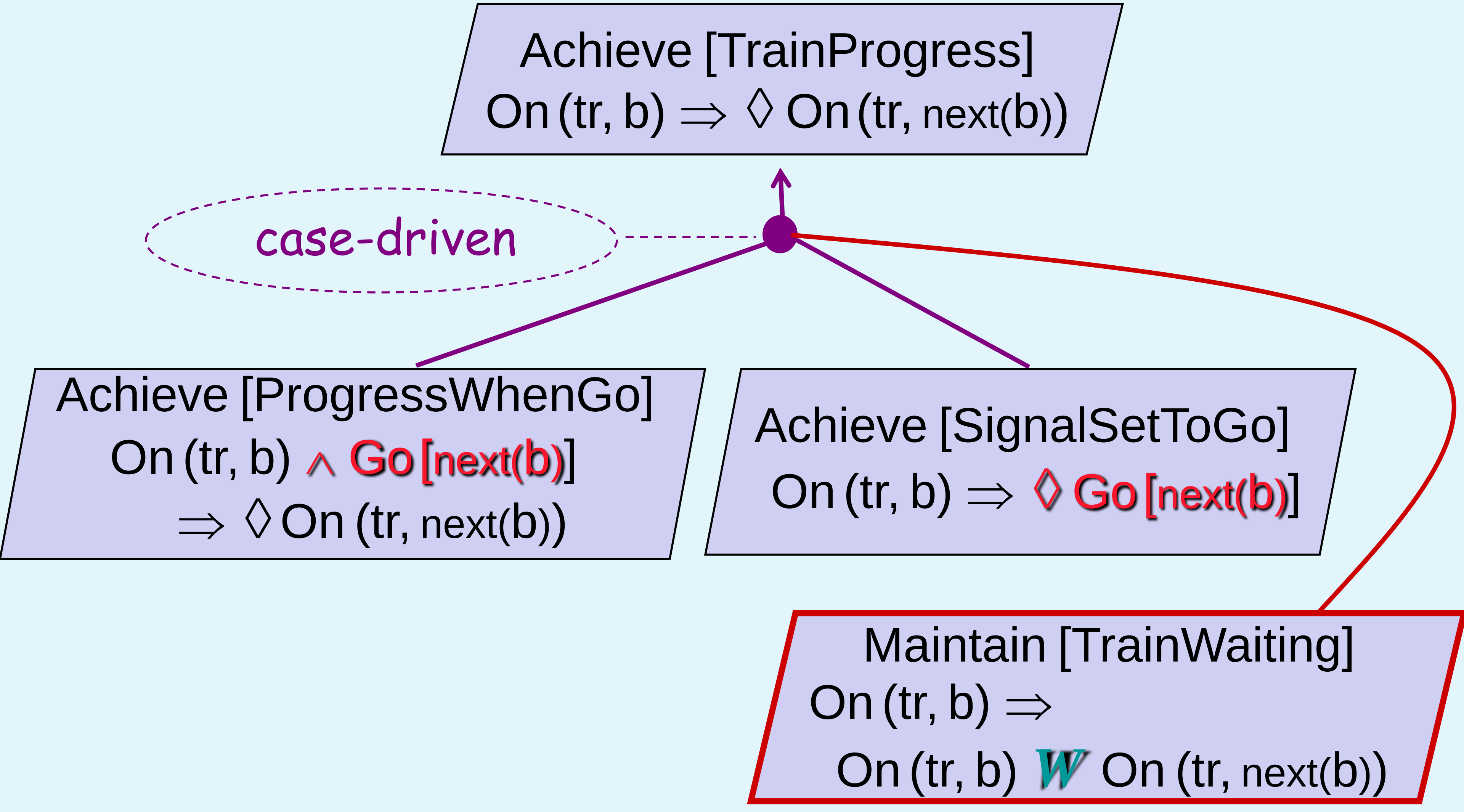
- Build catalogue of refinement patterns that encode *refinement tactics*
- Prove patterns formally, once for all
- Reuse through instantiation, in matching situation
- Some frequent patterns:



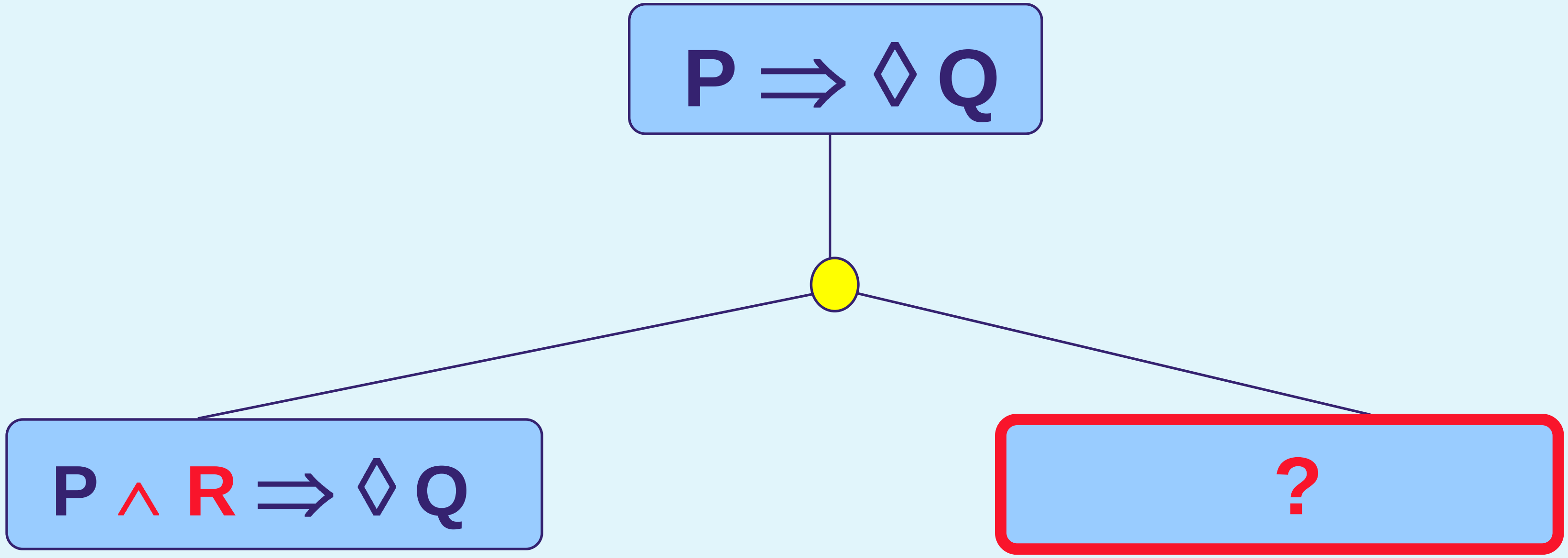
Checking a goal refinement with patterns



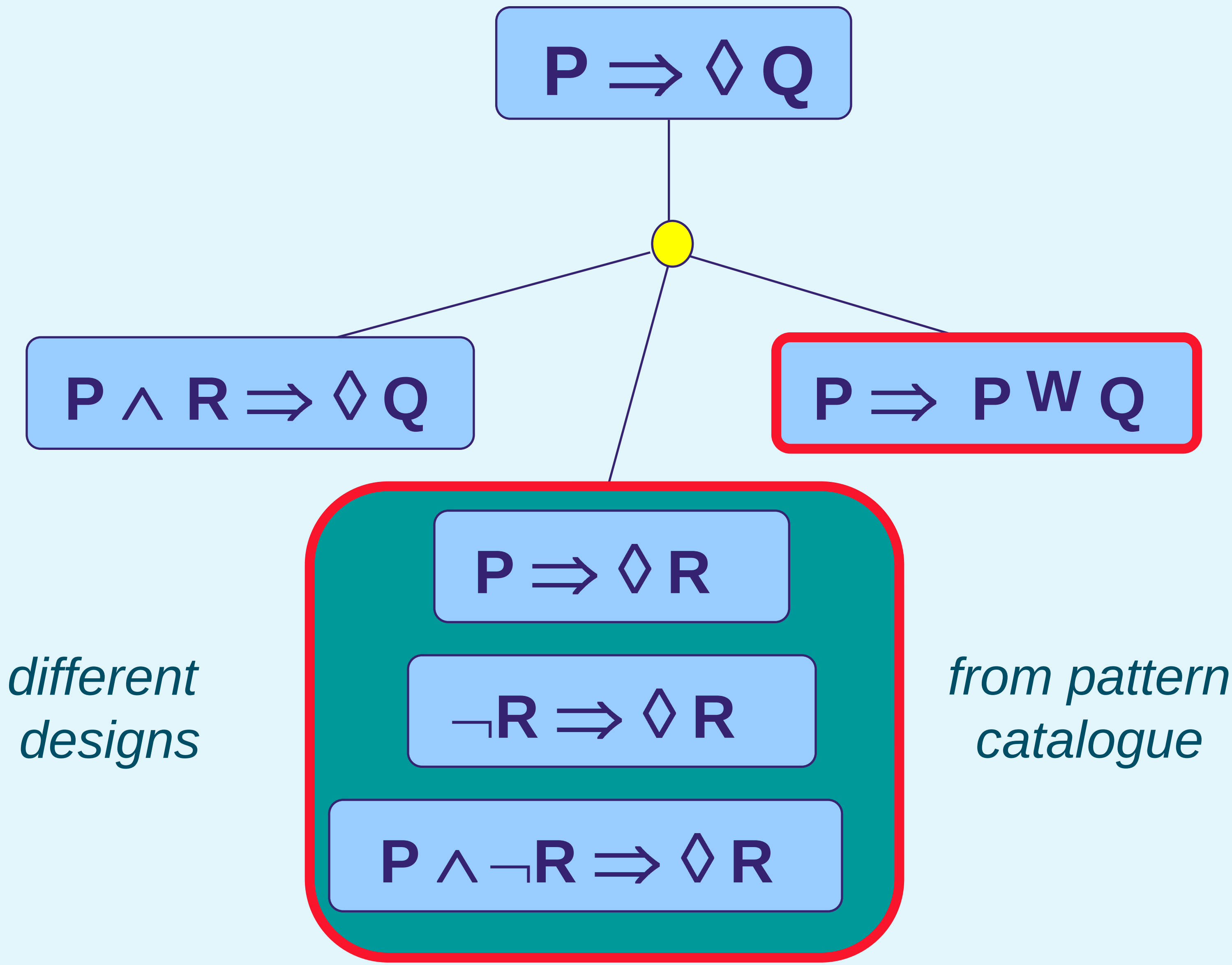
Checking goal refinements with patterns



Patterns provide guidance in formal refinement



Patterns provide guidance in formal refinement (2)



Use formal pattern => reuse formal proof

hyp

1. $P \Rightarrow \Diamond R$
2. $P \wedge R \Rightarrow \Diamond Q$

proof

3. $P \Rightarrow P \, \mathcal{W} \, Q$
4. $P \Rightarrow (P \, \mathcal{U} \, Q) \vee \Box P$
5. $P \Rightarrow \Diamond Q \vee \Box P$
6. $P \Rightarrow \Diamond R \wedge (\Diamond Q \vee \Box P)$
7. $P \Rightarrow (\Diamond R \wedge \Diamond Q) \vee (\Diamond R \wedge \Box P)$
8. $P \Rightarrow (\Diamond R \wedge \Diamond Q) \vee \Diamond (R \wedge P)$
9. $P \Rightarrow (\Diamond R \wedge \Diamond Q) \vee \Diamond \Diamond Q$
10. $P \Rightarrow (\Diamond R \wedge \Diamond Q) \vee \Diamond Q$

thesis

11. $P \Rightarrow \Diamond Q$

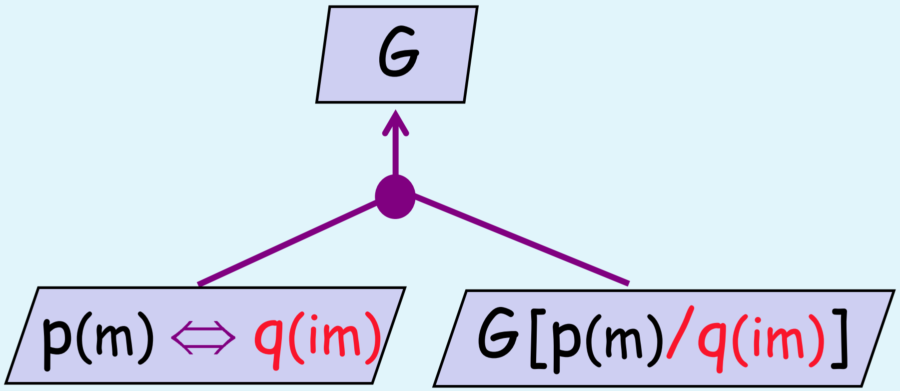
PROVED ONCE FOREVER

Resolving goal unrealizability: the *Introduce Accuracy goal* pattern

- WHEN:

  agent ag cannot monitor variable m to realize G [m]

- WHAT:
  - introduce *monitorable image* im of m
  - generate refinement :



Generating refinements & assignments

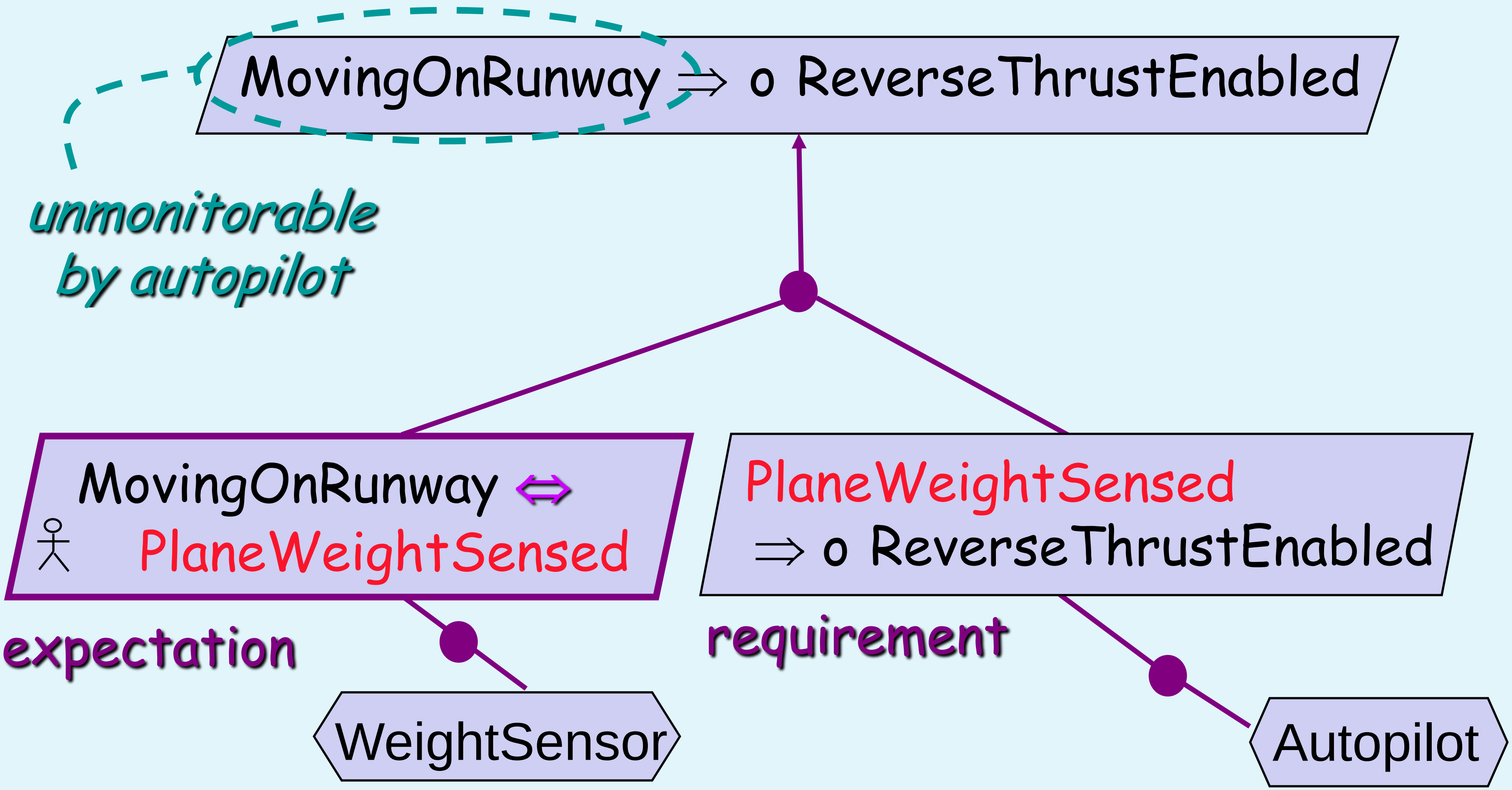
Introduce *Accuracy* goal: example

MovingOnRunway ⇒ o ReverseThrustEnabled

unmonitorable by autopilot

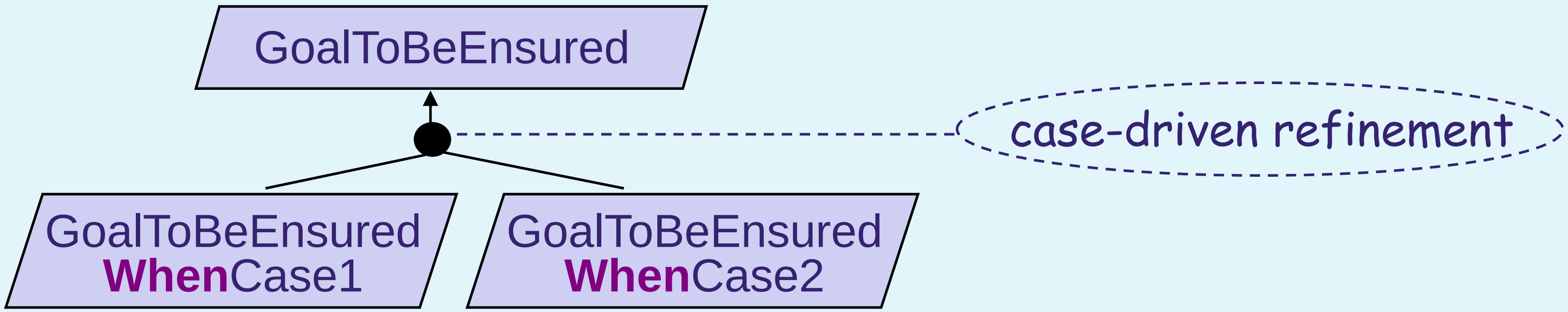
Generating refinements & assignments

Introduce *Accuracy* goal: example

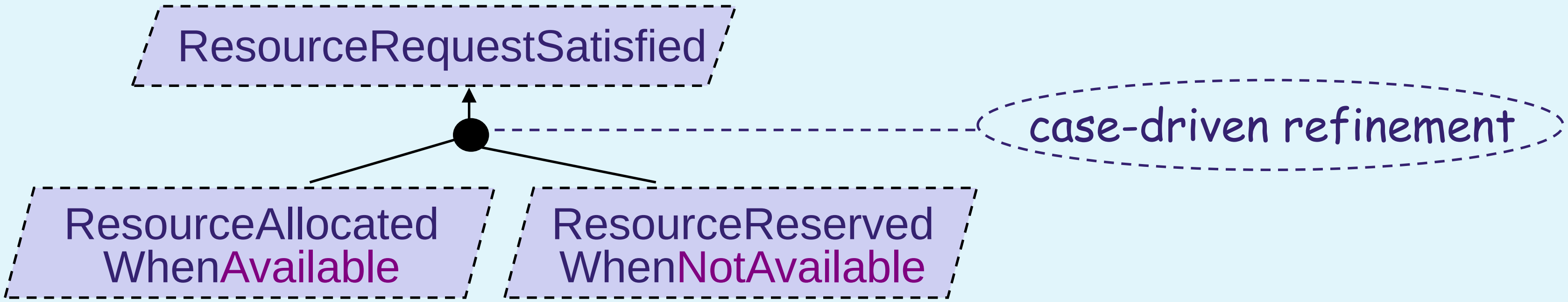


Formal refinement patterns can be used informally

- **Refinement by case**
  - Applicable when goal achievement space can be *partitioned into cases*

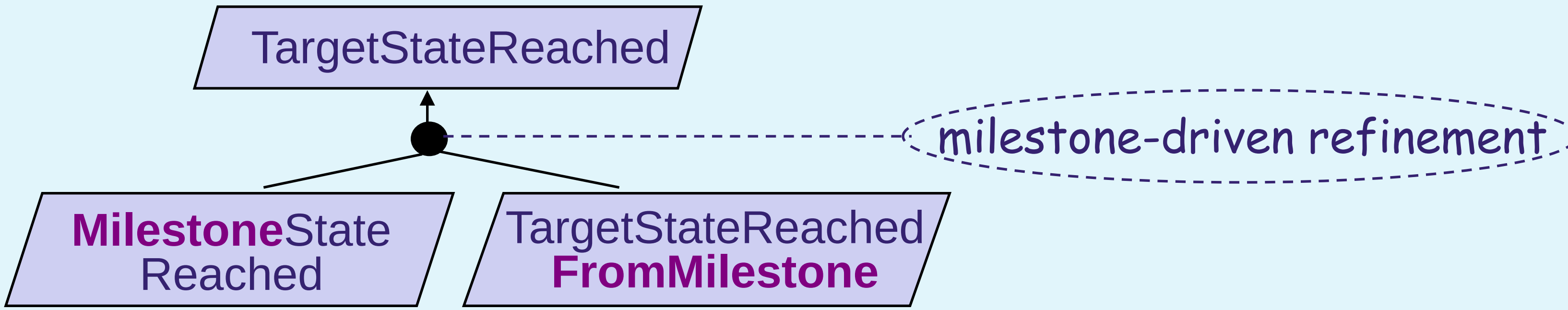


  - Example of use:

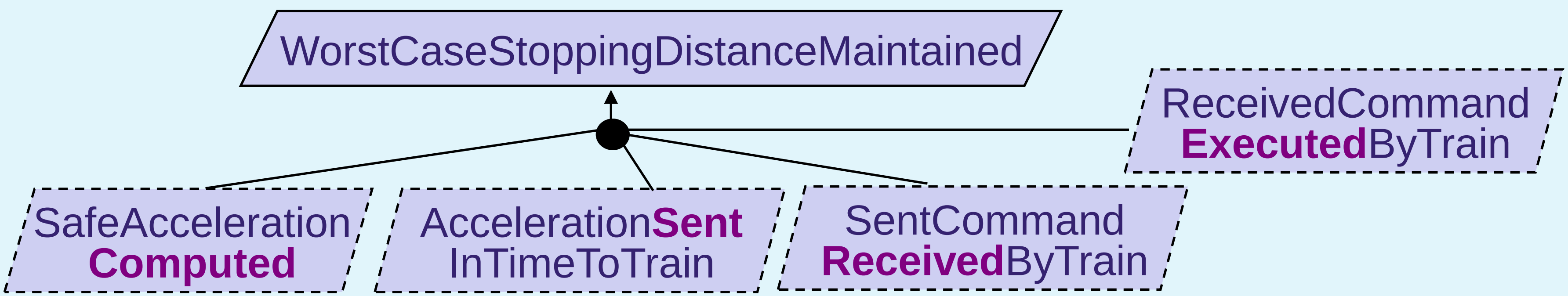


Formal patterns can be used informally (2)

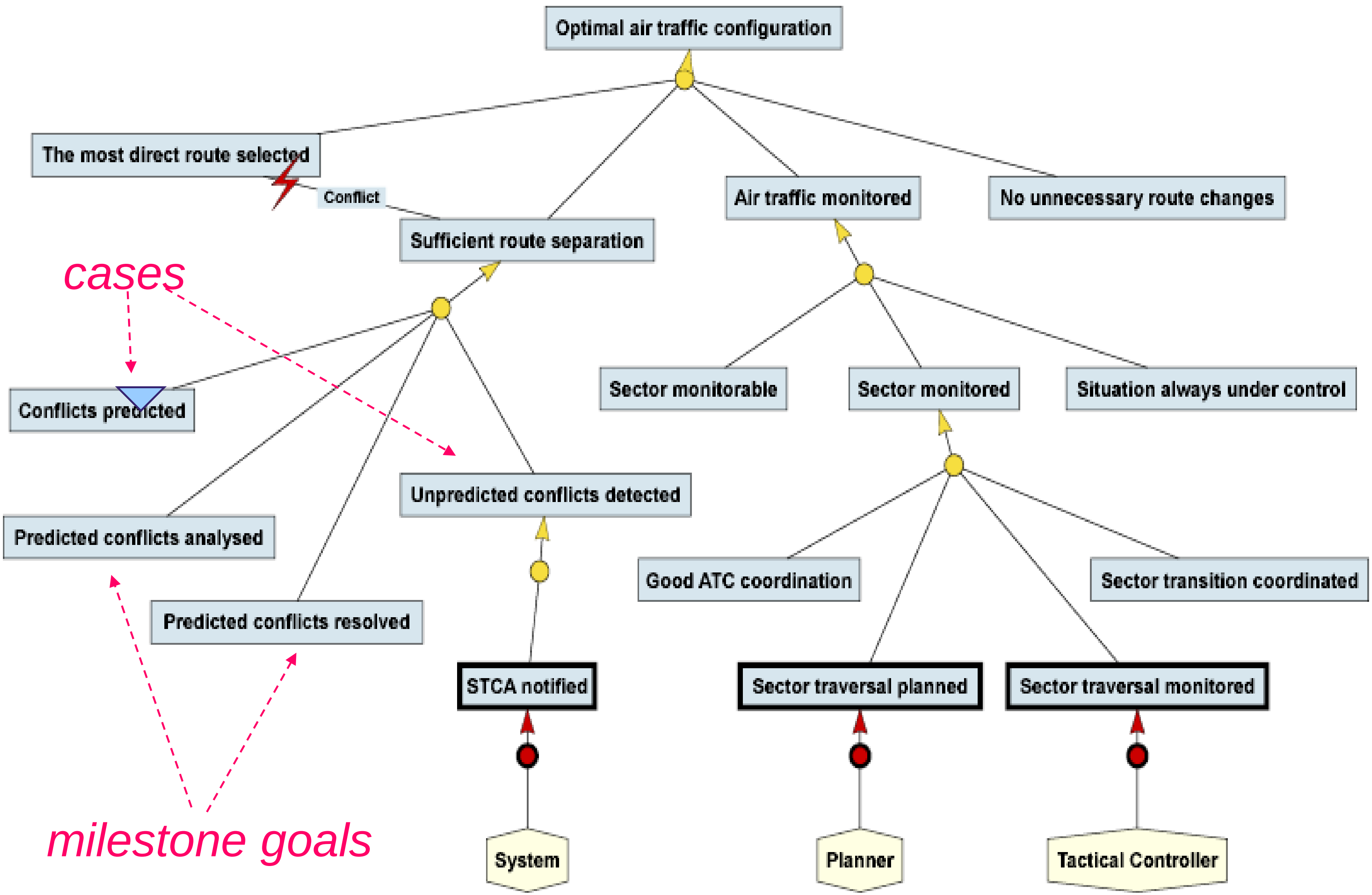
- **Refinement by milestone**
  - Applicable when *milestone states* can be identified on the way to the goal's target condition



  - Example of use:

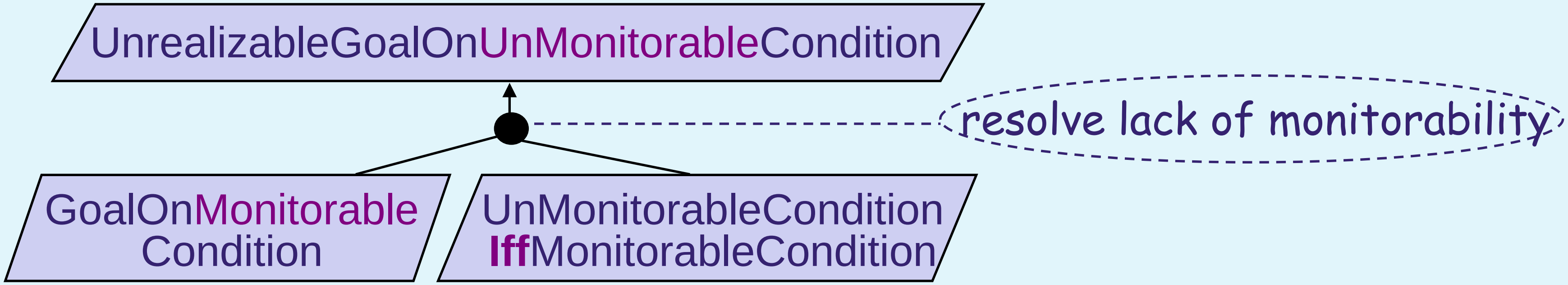


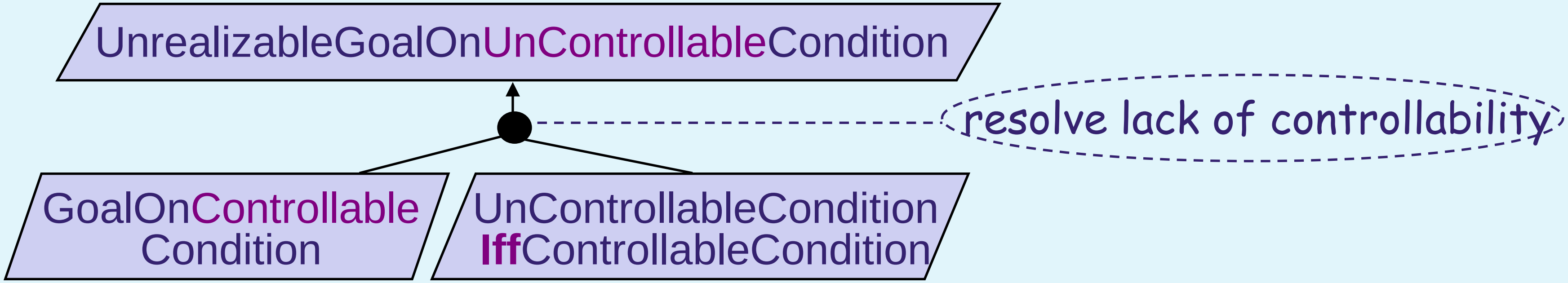
Informal use of patterns can reveal errors



Formal patterns can be used informally (3)

- Refinement towards goal realizability

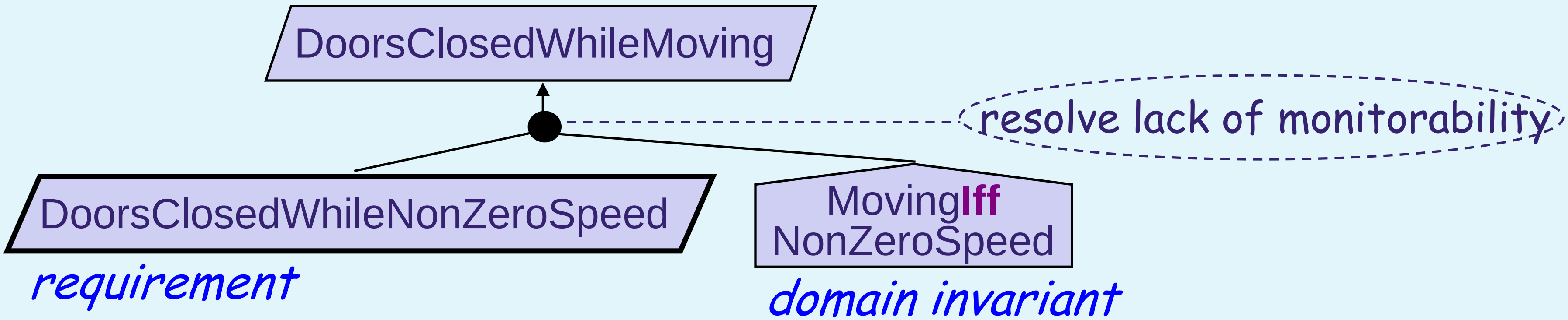


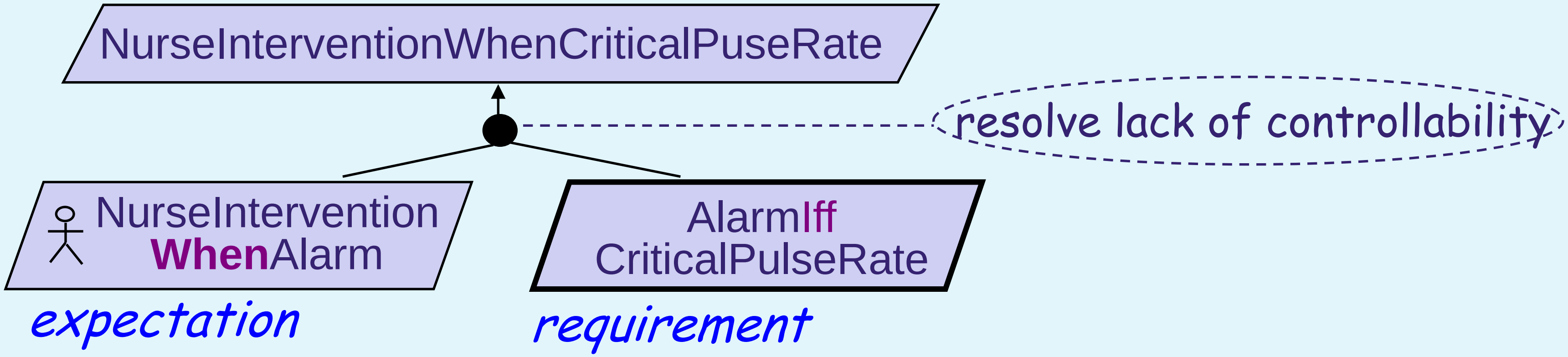


child node may be goal (incl. requirement, expectation)
or domain property (invariant/hypothesis)

Formal patterns can be used informally (4)

Refinement towards goal realizability: example of use





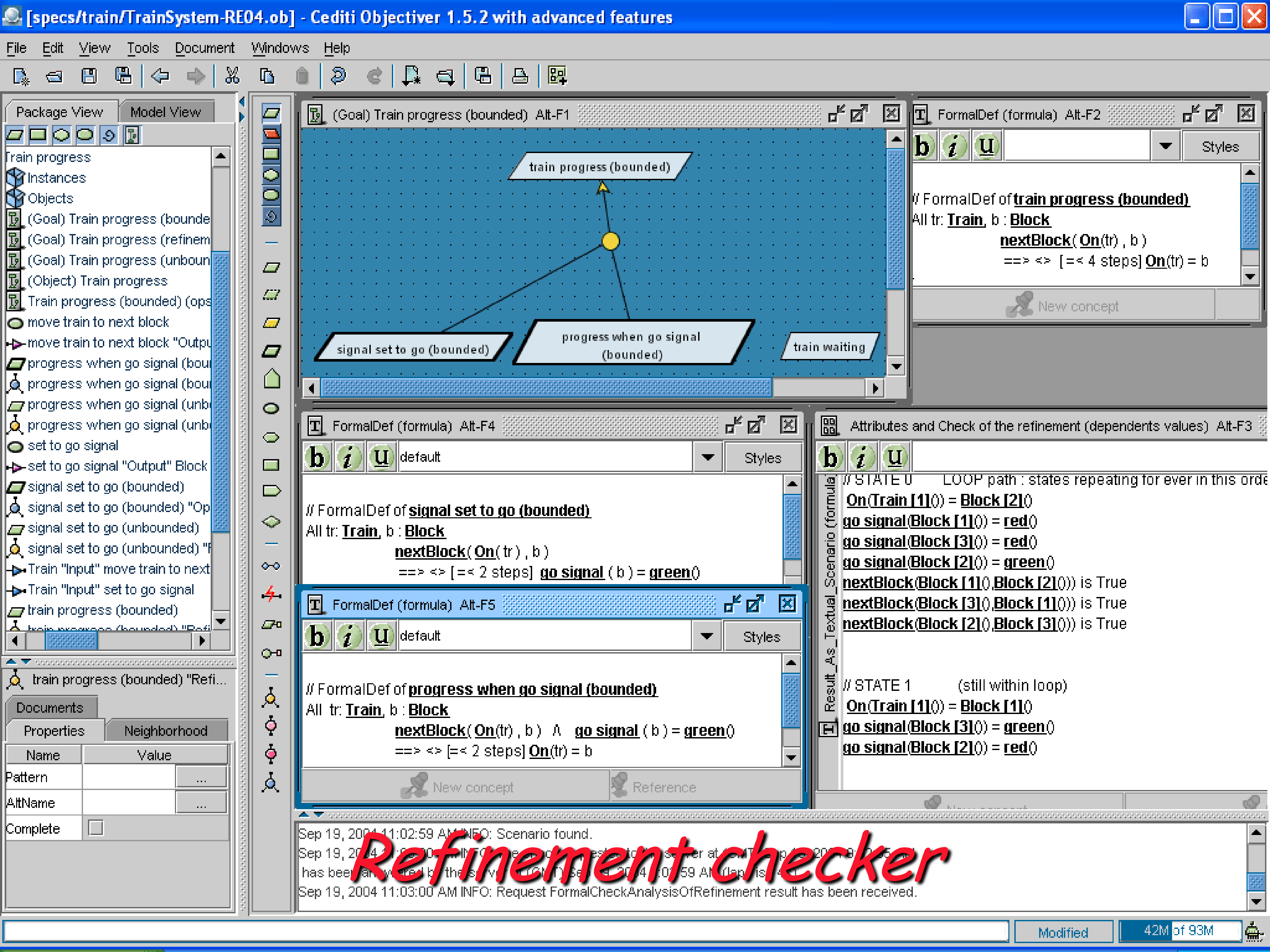
Refinement checking: roundtrip use of bounded SAT solver

- Incremental check/debug of model fragments
- On selected object instances *(propositionalization)*
- With bounded traces (to be given)
- Ouput:

OK (no counterexample found within trace bound)

KO + counter-example scenario satisfying

$$G_1 \wedge \ldots \wedge G_n \wedge \mathrm{Dom} \wedge \neg G$$


[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
Package View
Model View
Train progress
Instances
Objects
(Goal) Train progress (bounde
(Goal) Train progress (refinem
(Goal) Train progress (unboun
(Object) Train progress
Train progress (bounded) (ops
move train to next block
move train to next block "Outpu
progress when go signal (bou
progress when go signal (bou
progress when go signal (unb
progress when go signal (unb
set to go signal
set to go signal "Output" Block
signal set to go (bounded)
signal set to go (bounded) "Op
signal set to go (unbounded)
signal set to go (unbounded) "I
Train "Input" move train to next
Train "Input" set to go signal
train progress (bounded)
train progress (bounded) "Refi...
Documents
Properties
Neighborhood
Name
Value
Pattern
AltName
Complete
(Goal) Train progress (bounded) Alt-F1
train progress (bounded)
signal set to go (bounded)
progress when go signal (bounded)
train waiting
FormalDef (formula) Alt-F2
Styles
// FormalDef of train progress (bounded)
All tr: Train, b : Block
nextBlock(On(tr) , b)
==> <> [=< 4 steps] On(tr) = b
New concept
FormalDef (formula) Alt-F4
default
Styles
// FormalDef of signal set to go (bounded)
All tr: Train, b : Block
nextBlock(On(tr) , b)
==> <> [=< 2 steps] go signal (b) = green()
FormalDef (formula) Alt-F5
default
Styles
// FormalDef of progress when go signal (bounded)
All tr: Train, b : Block
nextBlock(On(tr) , b) ∧ go signal (b) = green()
==> <> [=< 2 steps] On(tr) = b
New concept
Reference
Attributes and Check of the refinement (dependents values) Alt-F3
// STATE 0 LOOP path : states repeating for ever in this orde
On(Train [1]()) = Block [2]()
go signal(Block [1]()) = red()
go signal(Block [3]()) = red()
go signal(Block [2]()) = green()
nextBlock(Block [1](),Block [2]()) is True
nextBlock(Block [3](),Block [1]()) is True
nextBlock(Block [2](),Block [3]()) is True
// STATE 1 (still within loop)
On(Train [1]()) = Block [1]()
go signal(Block [3]()) = green()
go signal(Block [2]()) = red()
Result_As_Textual_Scenario (formula
Sep 19, 2004 11:02:59 AM INFO: Scenario found.
Sep 19, 2004 11:03:00 AM INFO: Request FormalCheckAnalysisOfRefinement result has been received.
Refinement checker
Modified
42M of 93M

Formal reasoning: abductive generation of obstacles

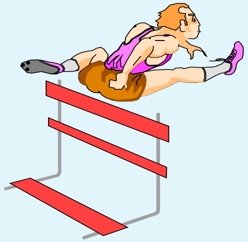
- Aim: Find *O* such that

  $O, Dom \mid\!-\!- \neg G$, $Dom \not\models \neg O$

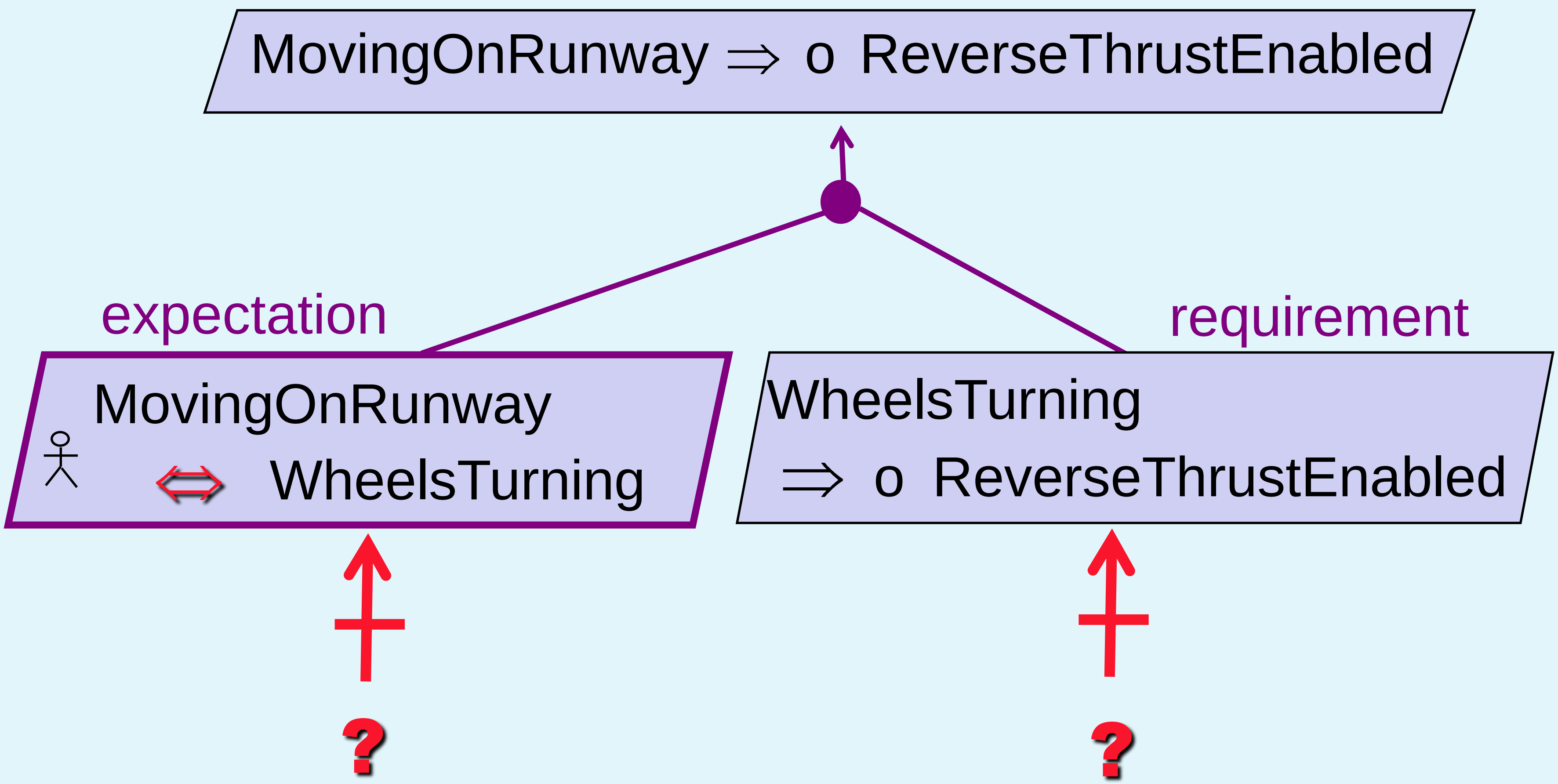
- Approach 1: Use precondition calculus to get ¬ *G* from *Dom*

  = regression of goal negation through domain theory

- Approach 2: Use formal obstruction patterns



Generating obstacles by regression



Generating obstacles by regression

Find precondition for obstruction of ...

MovingOnRunway $\Rightarrow$ WheelsTurning

Generating obstacles by regression

Find precondition for obstruction of ...

$$\text{MovingOnRunway} \Rightarrow \text{WheelsTurning}$$

→ goal negation:

$$\Diamond\ \text{MovingOnRunway} \land \neg\ \text{WheelsTurning}$$

Generating obstacles by regression

Find precondition for obstruction of ...

MovingOnRunway ⇒ WheelsTurning

→ goal negation:

◊ MovingOnRunway ∧ ¬ WheelsTurning

→ regress through Dom:

? *necessary* conditions for *wheels turning* ?

WheelsTurning ⇒ ¬ Aquaplaning

i.e. Aquaplaning ⇒ ¬ WheelsTurning

Generating obstacles by regression

Find precondition for obstruction of ...

$$MovingOnRunway \Rightarrow WheelsTurning$$

→ goal negation:

$$\Diamond\ MovingOnRunway \wedge \neg\ WheelsTurning$$

→ regress through Dom:

? *necessary* conditions for *wheels turning* ?

$$WheelsTurning \Rightarrow \neg\ Aquaplaning$$

i.e. $$Aquaplaning \Rightarrow \neg\ WheelsTurning$$

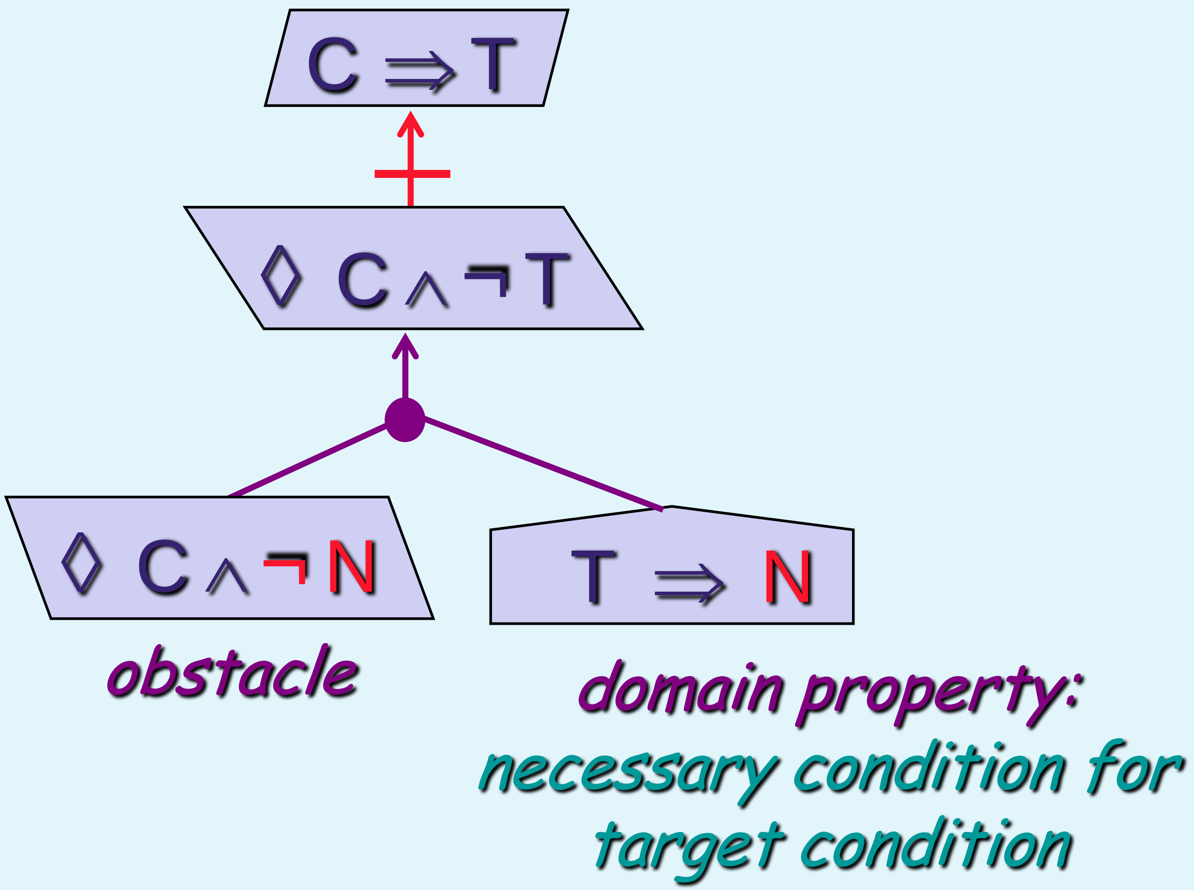
→ RHS unifiable:

$$\Diamond\ MovingOnRunway \wedge Aquaplaning$$

Warsaw obstacle

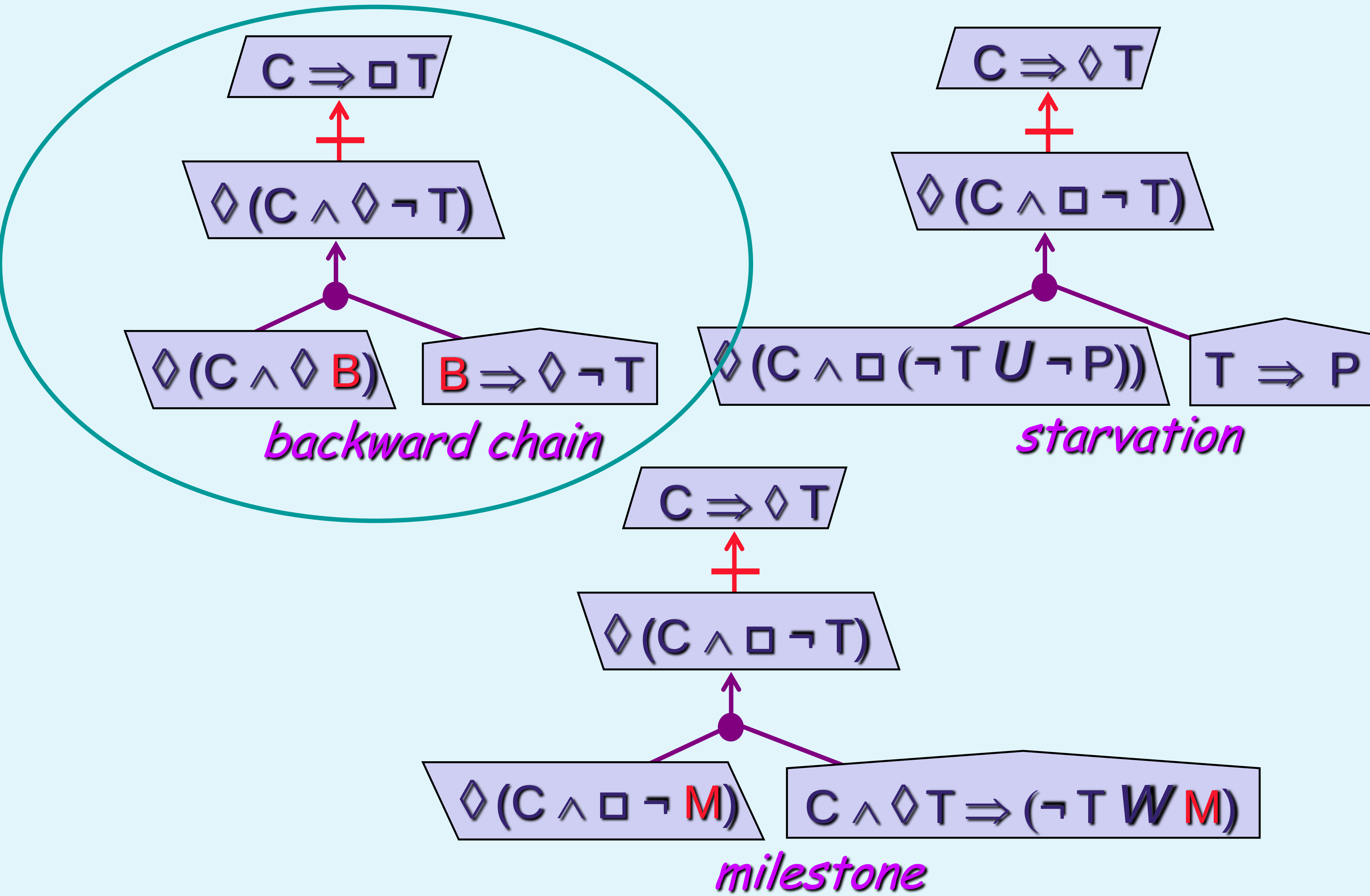
... or use formal obstruction patterns

- Very frequent pattern, used in this example:

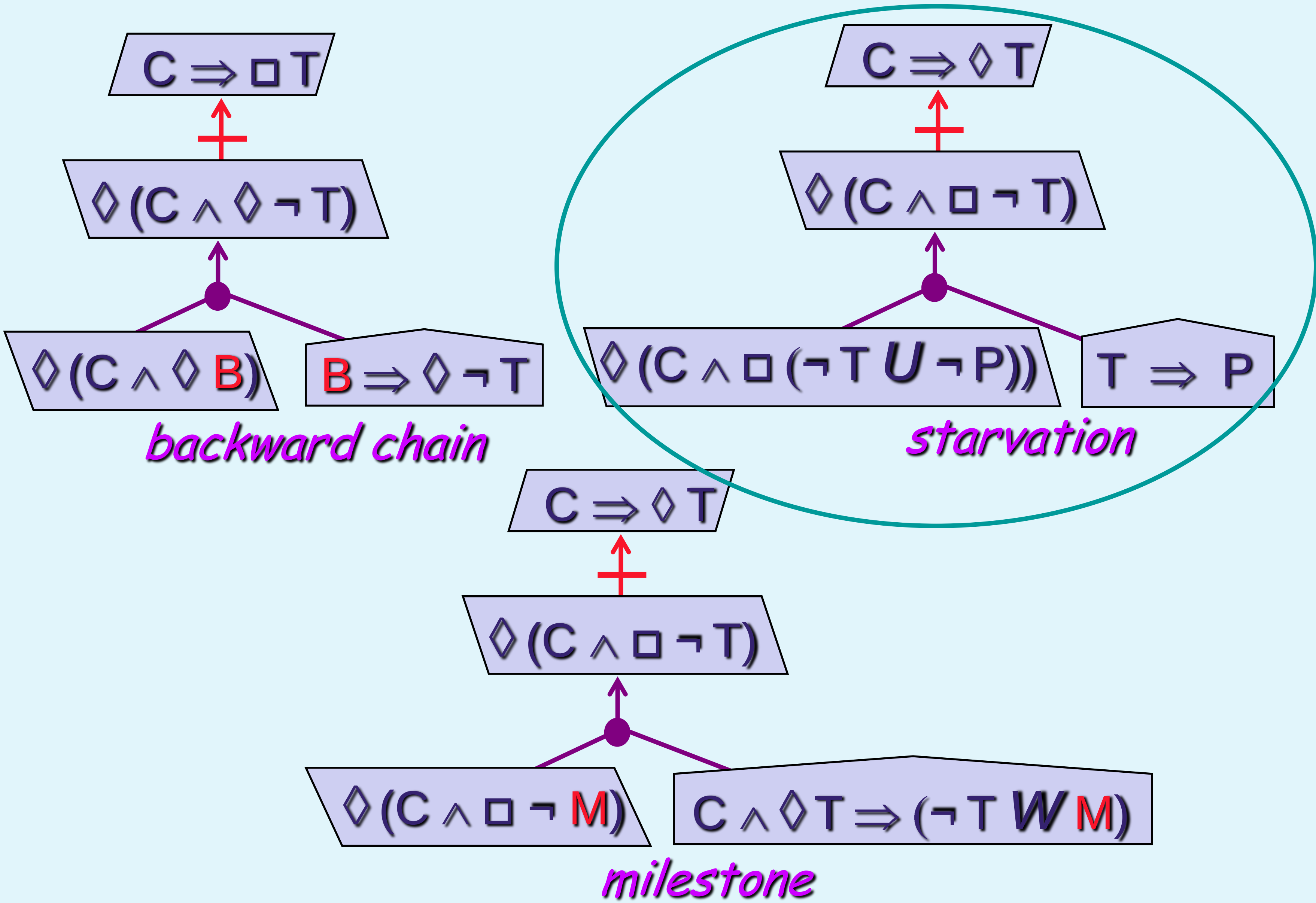


- Can be used to *elicit domain properties* as well

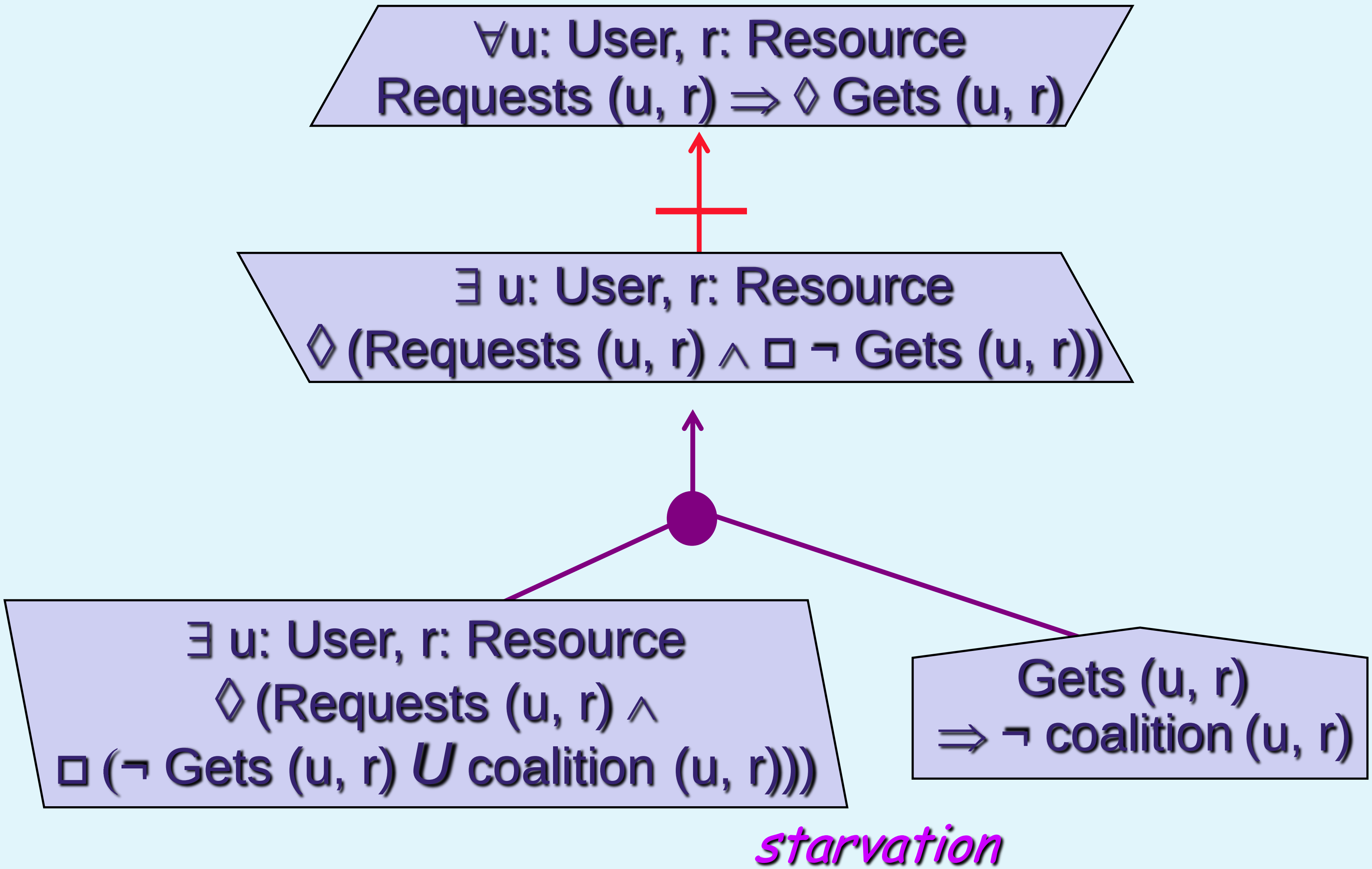
Some frequent obstruction patterns



Some frequent obstruction patterns



Example of pattern instantiation



Course outline

- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  - A goal-oriented model building method in action
  - Checking goal refinements
  - Obstacle analysis for high-assurance applications
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals
  - Model checking against confidentiality requirements

Application-level security

- Application is secure iff meets security goals
- Security goal refers to environment assets to be protected against undesired behaviors

  Confidentiality, integrity, availability, privacy, accountability, non-repudiation, ...
- Threat = obstacle to security goal
- Security countermeasure = obstacle resolution

Specification patterns for security goals

- Confidentiality goals

  Avoid [SensitiveInfoKnownByUnauthorizedAgent]

  $\forall ag$: Agent, ob: Object

  $\neg$ Authorized (ag, ob.Info) $\Rightarrow$ $\neg$ $KnowsV_{ag}$ (ob.info)

  $$KnowsV_{ag}(v) \equiv \exists x: Knows_{ag}(x = v)$$

  $$Knows_{ag}(P) \equiv Belief_{ag}(P) \wedge P$$

  ↑

  "*P* is in *ag*'s memory"

- Other patterns for privacy, availability, integrity, authentication, non-repudiation, ...

Application-specific instantiation of security goal patterns

Goal Avoid [SensitiveInfoKnownByUnauthorizedAgent]

$\forall$ag: Agent, ob: Object

$\neg$ *Authorized (ag, ob.Info)* $\Rightarrow$ $\neg$ $KnowsV_{ag}$ (ob.info)

$\downarrow$ Web banking services

Object / Account [#, PIN]

Authorized (ag, acc) $\equiv$

Owner (ag, acc) $\vee$ Proxy (ag, acc) $\vee$ Manager (ag, acc)

$\downarrow$

Goal Avoid [PaymentMediumKnownBy3rdParty]

$\forall$p: Person, acc: Account

$\neg$ [Owner (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manager (p, acc)]

$\Rightarrow$ $\neg$ [$KnowsV_p$ (acc.Acc#) $\wedge$ $KnowsV_p$ (acc.PIN)]

Application-specific instantiation of security goal patterns

Goal Avoid [SensitiveInfoKnownByUnauthorizedAgent]

$\forall$ ag: Agent, ob: Object

$\neg$ *Authorized (ag, ob.Info)* $\Rightarrow$ $\neg$ $KnowsV_{ag}$ (ob.info)

$\downarrow$ Web banking services

Object / Account [#, PIN] *sensitive info in object model*

Authorized (ag, acc) $\equiv$

Owner (ag, acc) $\vee$ Proxy (ag, acc) $\vee$ Manager (ag, acc)

$\downarrow$

Goal Avoid [PaymentMediumKnownBy3rdParty]

$\forall$ p: Person, acc: Account

$\neg$ [Owner (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manager (p, acc)]

$\Rightarrow$ $\neg$ [$KnowsV_p$ (acc.Acc#) $\wedge$ $KnowsV_p$ (acc.PIN)]

Application-specific instantiation of security goal patterns

Goal Avoid [SensitiveInfoKnownByUnauthorizedAgent]

$\forall$ ag: Agent, ob: Object

$\neg$ *Authorized (ag, ob.Info)* $\Rightarrow$ $\neg$ $KnowsV_{ag}$ (ob.info)

$\downarrow$ Web banking services

Object / Account [#, PIN] *sensitive info in object model*

Authorized (ag, acc) $\equiv$

Owner (ag, acc) $\vee$ Proxy (ag, acc) $\vee$ Manager (ag, acc)

$\downarrow$

Goal Avoid [PaymentMediumKnownBy3rdParty]

$\forall$ p: Person, acc: Account

$\neg$ [Owner (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manager (p, acc)]

$\Rightarrow$ $\neg$ [$KnowsV_p$ (acc.Acc#) $\wedge$ $KnowsV_p$ (acc.PIN)]

Further patterns for confidentiality goals

- Two dimensions of confidentiality:
  - Degree of approximate knowledge to be kept confidential
  - Timing along which knowledge should be kept confidential
- Pattern catalogue
  - Provides standard specification patterns
  - Hides complicate formulas

Specification patterns for confidentiality goals

Timing Of Knowledge	degree of knowledge					
	val	lb	ub	betw	full	Val
Now						
upTo						
Unless						
Until						
forever					full-confidential-forever	

Zooming on some patterns…

Specification patterns: a sample

Fully confidential value

$$\textit{Full-Confidential}_{ag}(x) \equiv$$
$$\forall\, v \in ran(x){:}\ \neg\, \textit{Knows}_{ag}(x \neq v)$$

Confidential forever

$$\textit{Y-Confidential-forever}_{ag}(x) \equiv$$
$$\forall\, w{:}\ x = w \rightarrow \Box\, \textit{Y-Confidential}_{ag}(x)$$
$$\text{with } Y \in \{val, lb, ub, betw, full\}$$

Specification by pattern instantiation:

$$\forall\, ep{:}\ ePurse,\ ag{:}\ Agent$$
$$\neg\, Owns(ag, ep) \wedge ag \neq ep$$
$$\Rightarrow \textit{full-Confidential-forever}_{ag}(ep.balance)$$

Course outline

- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  - A goal-oriented model building method in action
  - Checking goal refinements
  - Obstacle analysis for high-assurance applications
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals
  - Model checking against confidentiality requirements

Threat analysis: unintentional vs. intentional threats

- Unintentional threat: inadvertent violation of security goal
  - Handled by obstacle analysis on security leaf goals
  - E.g. accidental disclosure of confidential information
- Intentional threat: proactive violation of security goal by exploitation of unprotected data & system knowledge acquired through malicious behaviors, calculations, deductive inference, etc.
  - Handled by obstacle analysis augmented with malicious agents, their anti-goals, and their capabilities
  - E.g. E-shopping: Achieve[ItemReceived**AndNot**Paid]

Intentional threats require an *anti*-model

- The scope of the environment is extended to include malicious agents ("attackers")
  - human insiders or outsiders of the original system, tools, fake devices, ...
- Anti-goal = malicious obstacle to satisfy attacker's objectives (and break security goals)
- Anti-model = model linking anti-goals against a goal model

Intentional threats require an *anti*-model (2)

- An anti-model is a dual model ...
  - the software is now part of the attacker's environment
  - domain properties include software vulnerabilities
- Threat graph = refinement graph showing a plan ...
  - to achieve some anti-goal
  - in view of the attacker's capabilities

Analyzing intentional threats: attacker's capabilities

- Capabilities = two sets of conditions:
  - conditions that are monitorable by the attacker
  - conditions that are controllable by the attacker

    e.g. e-shopping: ItemPaidByCustomer,
    PaymentNotificationReceivedBySeller
- Most Knowledgeable Attacker (MKA):
  - Knows the goal model, the domain properties used in it, and the operation model

Trivially satisfied as attacker at RE time is the modeller looking for missing countermeasures

Worst-case threat analysis is desirable for complete exploration of security countermeasures

Threat analysis for intentional threats

- Build threat graphs from anti-goals:
  - Get initial anti-goals to be refined/abstracted --e.g., from negations of application-specific security goal
  - Identify attackers wishing them, their capabilities
  - Build anti-goal refinement/abstraction graphs until reaching conditions that are realizable by the attackers (monitorable or controllable)
- Derive new security goals as countermeasures to counter the leaf anti-goals in threat graphs

Step 1: Get initial anti-goals

- *Negate* security goal instantiation to application-specific "sensitive" objects ...

Goal Avoid [PaymentMediumKnownBy3rdParty]

$\forall$p: Person, acc: Account

$\neg$ Authorized (p, acc)

$\Rightarrow \neg$ [KnowsV$_p$ (acc.Acc#) $\wedge$ KnowsV$_p$ (acc.PIN)]

↓ goal negation

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]

$\Diamond \exists$ p: Person, acc: Account

$\neg$ Authorized (ag, acc)

$\wedge$ KnowsV$_p$ (acc.Acc#) $\wedge$ KnowsV$_p$ (acc.PIN)

Step 1: Get initial anti-goals

- *Negate* security goal instantiation to application-specific "sensitive" objects ...

Goal Avoid [PaymentMediumKnownBy3rdParty]

$\forall p$: Person, acc: Account

$\neg$ Authorized (p, acc)

$\Rightarrow \neg$ [$KnowsV_p$ (acc.Acc#) $\wedge$ $KnowsV_p$ (acc.PIN)]

↓ goal negation

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]

$\Diamond \exists p$: Person, acc: Account

$\neg$ Authorized (p, acc)

$\wedge$ $KnowsV_p$ (acc.Acc#) $\wedge$ $KnowsV_p$ (acc.PIN)

Step 2: Identify attackers wishing anti-goals

- For each initial anti-goal:
  - ask WHO might benefit from it
  - use of attacker taxonomies

Anti-Goal Achieve[PaymentMediumKnownBy3rdParty]

↓

Insiders: Bank QA team
Organization-specific agents

Outsiders: Thieves
Hackers
Terrorists, ...

Step 3: Build threat graph

- For each (initial anti-goal, attacker): build anti-goal refinement/abstraction graph ...
  - Informally: by use of refinement patterns or by WHY/HOW questions

WHY ⇒ parent anti-goals

HOW ⇒ child anti-goals

Formally: by regression through ...

... domain properties $P \Rightarrow AG$

⇒ anti-goal preconditions satisfiable *in domain*

... goal specs from attacked model

⇒ preconditions satisfiable *by attacked software*

Step 3: Build threat graph

- For each (initial anti-goal, attacker): build anti-goal refinement/abstraction graph ...
  - Informally: by use of refinement patterns or by WHY/HOW questions

    WHY ⇒ parent anti-goals

    HOW ⇒ child anti-goals

  - Formally: by regression through ...

    ... domain properties $P \Rightarrow AG$

    ⇒ anti-goal preconditions satisfiable *in domain*

    ... goal specs from attacked model

    ⇒ preconditions satisfiable *by attacked software*

Anti-goal refinement by regression through domain

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]

$\Diamond \exists$ p: Person, acc: Account

$\neg$ Authorized (p, acc) $\wedge$ *KnowsV$_p$ (Acc#)* $\wedge$ *KnowsV$_p$ (PIN)*

↓ *domain property as sufficient condition ?*

$\forall$p: Person, acc: Account

$\neg$ Authorized (ag, acc) $\wedge$ KnowsV$_p$ (acc.PIN)

$\wedge$ ($\exists$ x: Acc#) (Found (p, x) $\wedge$ Matching (acc.PIN, x))

$\Rightarrow$ KnowsV$_p$ (acc.Acc#) $\wedge$ KnowsV$_p$ (acc.PIN)

↓ *anti-subgoal:*

$\Diamond \exists$ p: Person, acc: Account

$\neg$ Authorized (ag, acc) $\wedge$ KnowsV$_p$ (acc.PIN)

$\wedge$ ($\exists$ x: Acc#) (Found (p, x) $\wedge$ Matching (acc.PIN, x))

Anti-goal refinement by regression through domain

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]

$\Diamond \exists$ p: Person, acc: Account

$\neg$ Authorized (p, acc) $\wedge$ *$KnowsV_p$ (Acc#)* $\wedge$ *$KnowsV_p$ (PIN)*

↓ *dom prop as sufficient condition ?*

$\forall$p: Person, acc: Account

$\neg$ Authorized (p, acc) $\wedge$ $KnowsV_p$ (acc.PIN)

$\wedge$ ($\exists$ x: Acc#) (Found (p, x) $\wedge$ Matching (acc.PIN, x))

$\Rightarrow$ *$KnowsV_p$ (acc.Acc#)* $\wedge$ *$KnowsV_p$ (acc.PIN)*

↓ *anti-subgoal:*

$\Diamond \exists$ p: Person, acc: Account

$\neg$ Authorized (ag, acc) $\wedge$ $KnowsV_p$ (acc.PIN)

$\wedge$ ($\exists$ x: Acc#) (Found (p, x) $\wedge$ Matching (acc.PIN, x))

Anti-goal refinement by regression through domain

Anti-Goal Achieve [PaymentMediumKnownBy3rdParty]

$\Diamond \exists$ p: Person, acc: Account

$\neg$ Authorized (ag, acc) $\wedge$ *KnowsV$_p$ (Acc#)* $\wedge$ *KnowsV$_p$ (PIN)*

↓ *dom prop as sufficient condition ?*

$\forall$p: Person, acc: Account

$\neg$ Authorized (ag, acc) $\wedge$ KnowsV$_p$ (acc.PIN)

$\wedge$ ($\exists$ x: Acc#) (Found (p, x) $\wedge$ Matching (acc.PIN, x))

$\Rightarrow$ *KnowsV$_p$ (acc.Acc#)* $\wedge$ *KnowsV$_p$ (acc.PIN)*

↓ *anti-subgoal:*

$\Diamond \exists$ p: Person, acc: Account

$\neg$ Authorized (p, acc) $\wedge$ KnowsV$_p$ (acc.PIN)

$\wedge$ ($\exists$ x: Acc#) (Found (p, x) $\wedge$ Matching (acc.PIN, x))

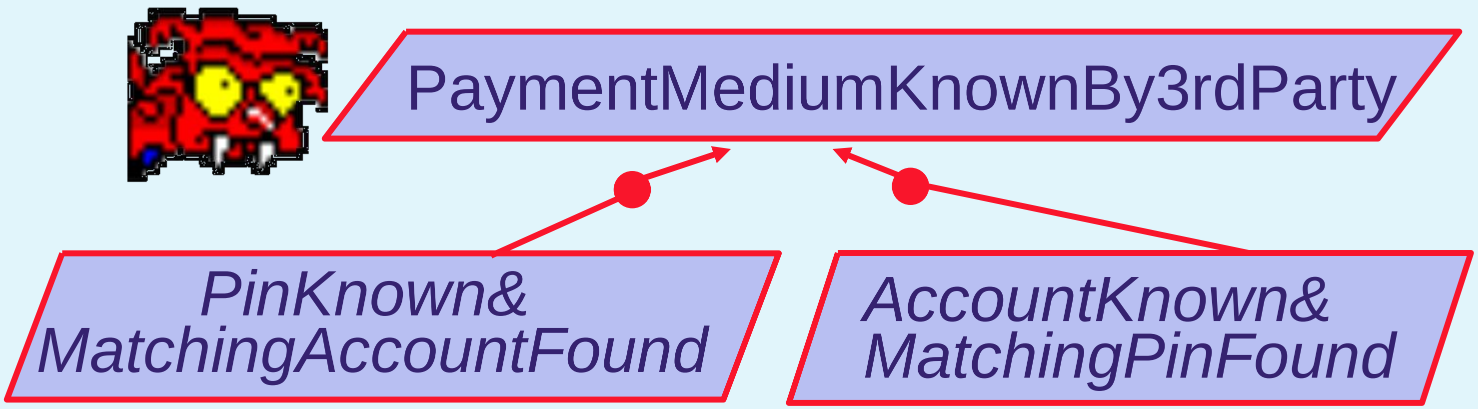
Build threat graph: refine until …

- … terminal conditions are reached …
  - anti-requirements

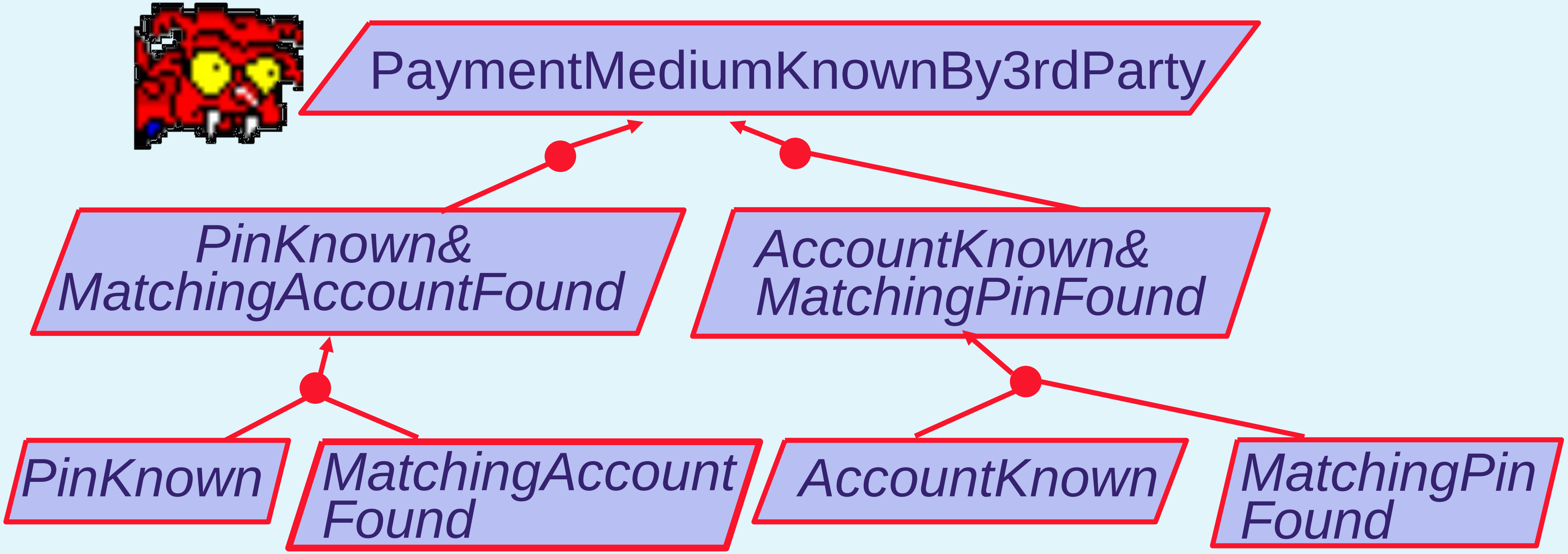
    realizable in terms of attacker's capabilities
  - vulnerabilities of attackee

    properties of anti-domain

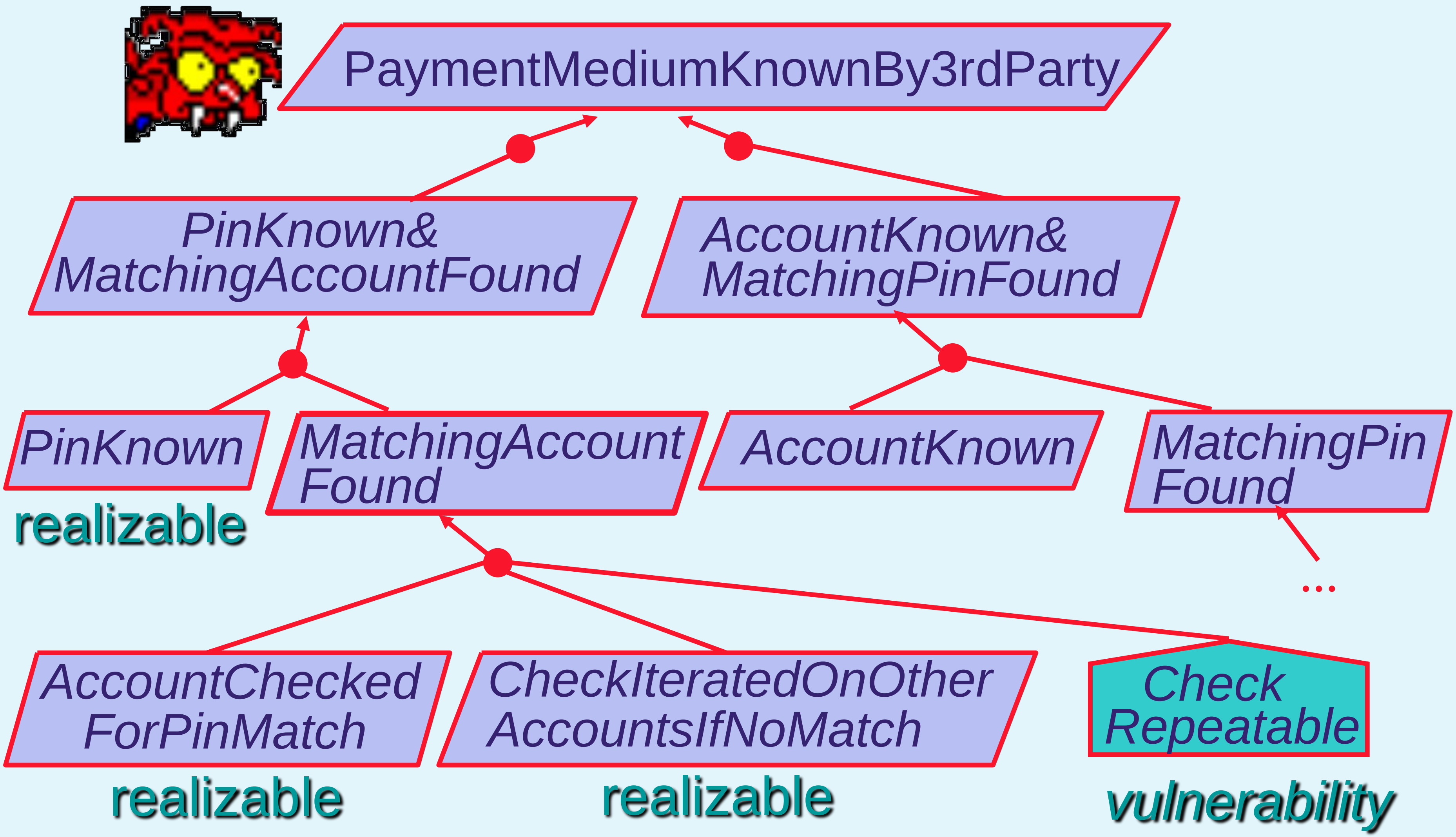
Refinement towards realizability by attacker: a known attack



Refinement towards realizability by attacker: a known attack



Refinement towards realizability by attacker: a known attack



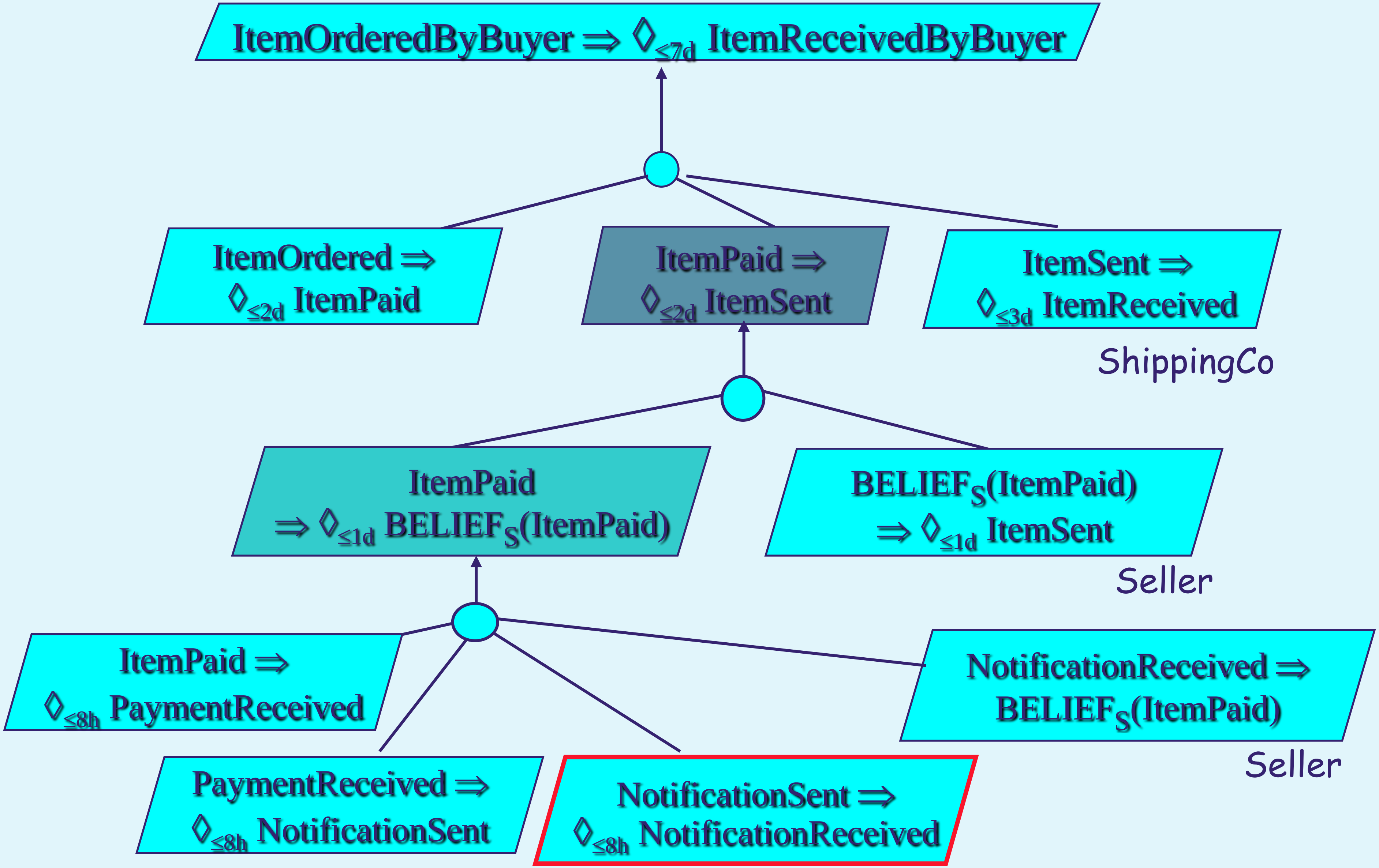
Deriving countermeasures

- New security goals obtained by application of resolution operators, e.g.
  - *Avoid* [anti-goal]:

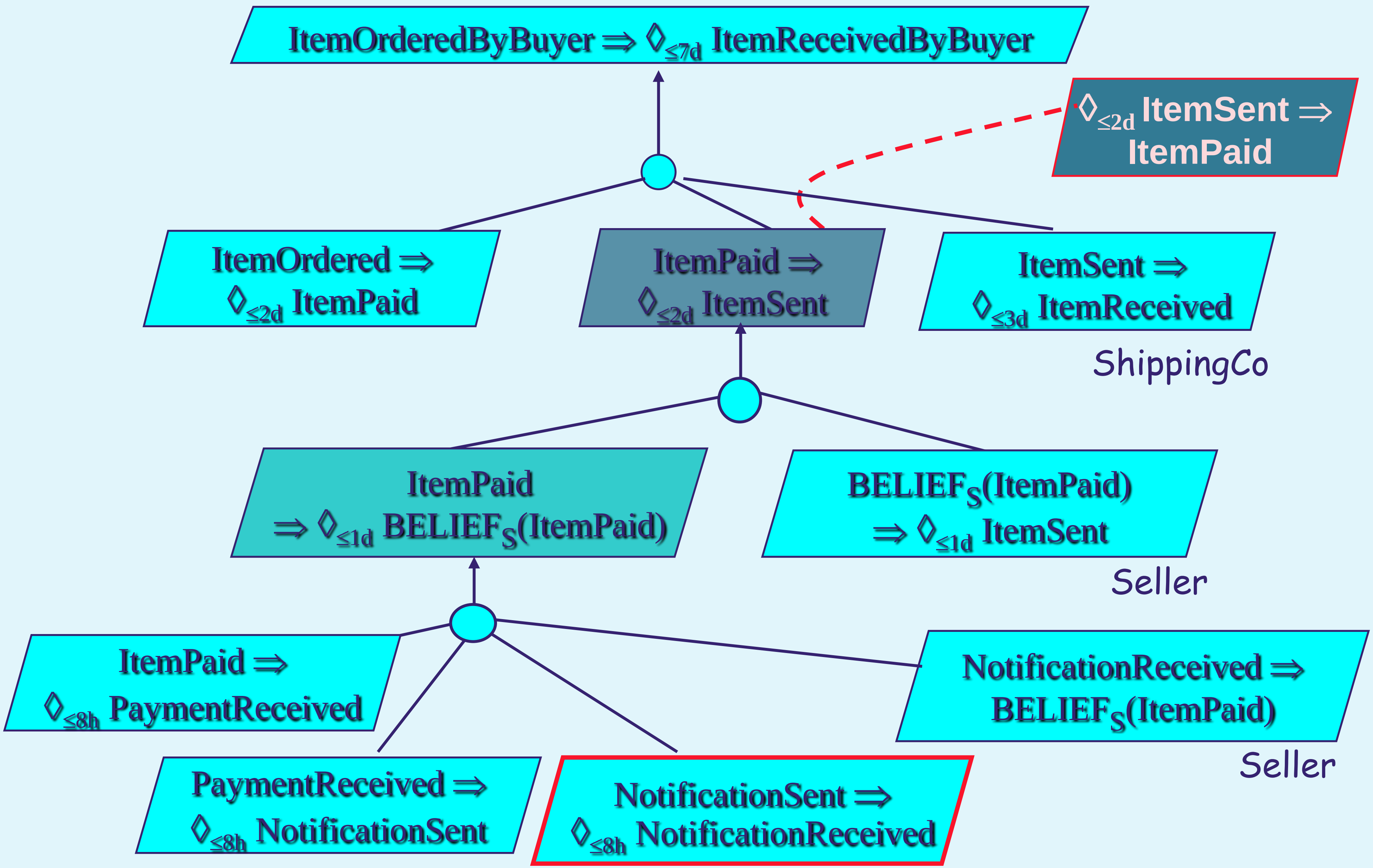
    Avoid [AccountCheckRepeatableFromPin]

    Avoid [PinCheckRepeatableFromAccount]
  - Make vulnerability condition *unmonitorable* by attacker
  - Make anti-requirement *uncontrollable* by attacker
- To be further refined along alternative OR-branches in the updated goal model

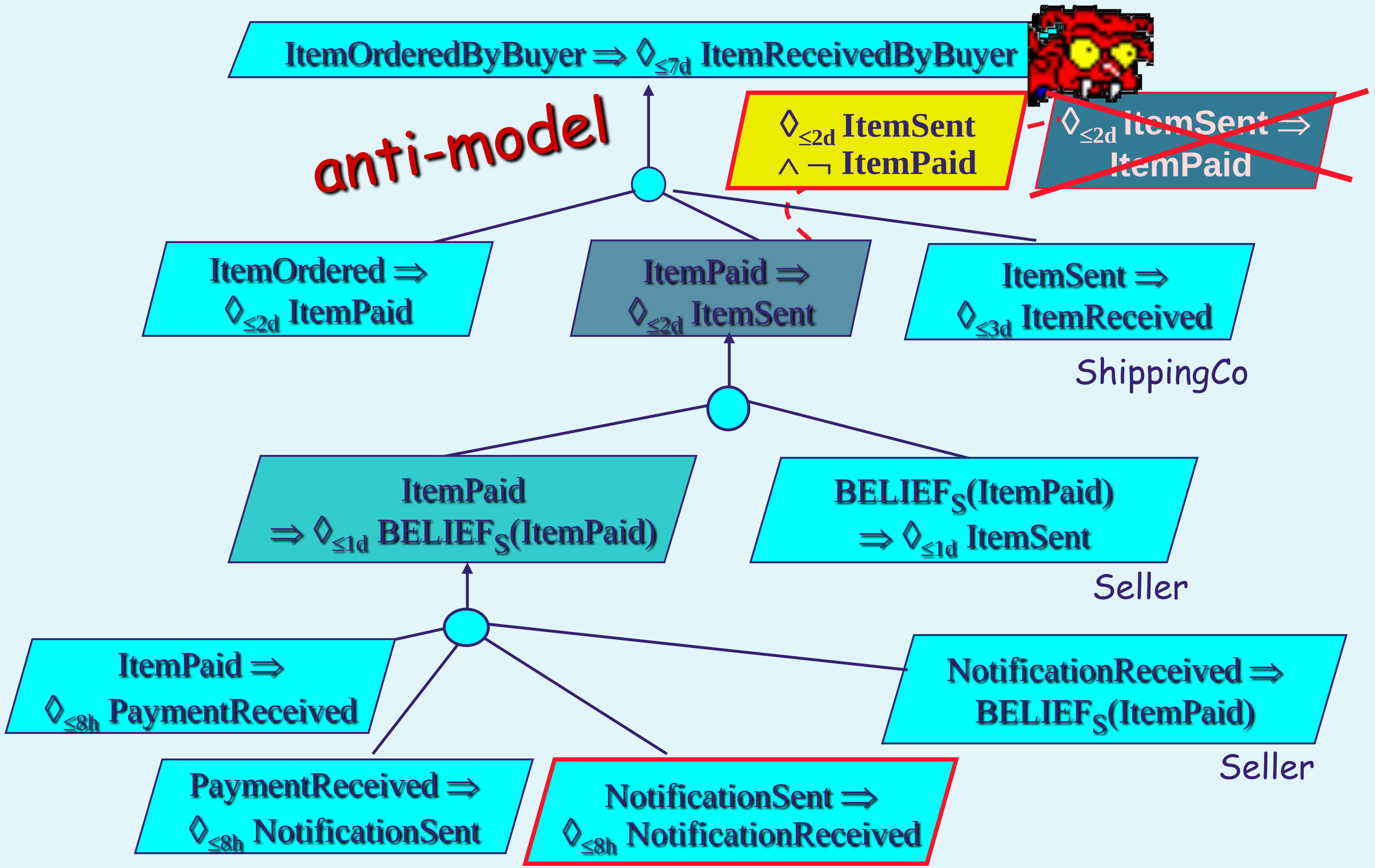
Online shopping: functional goals



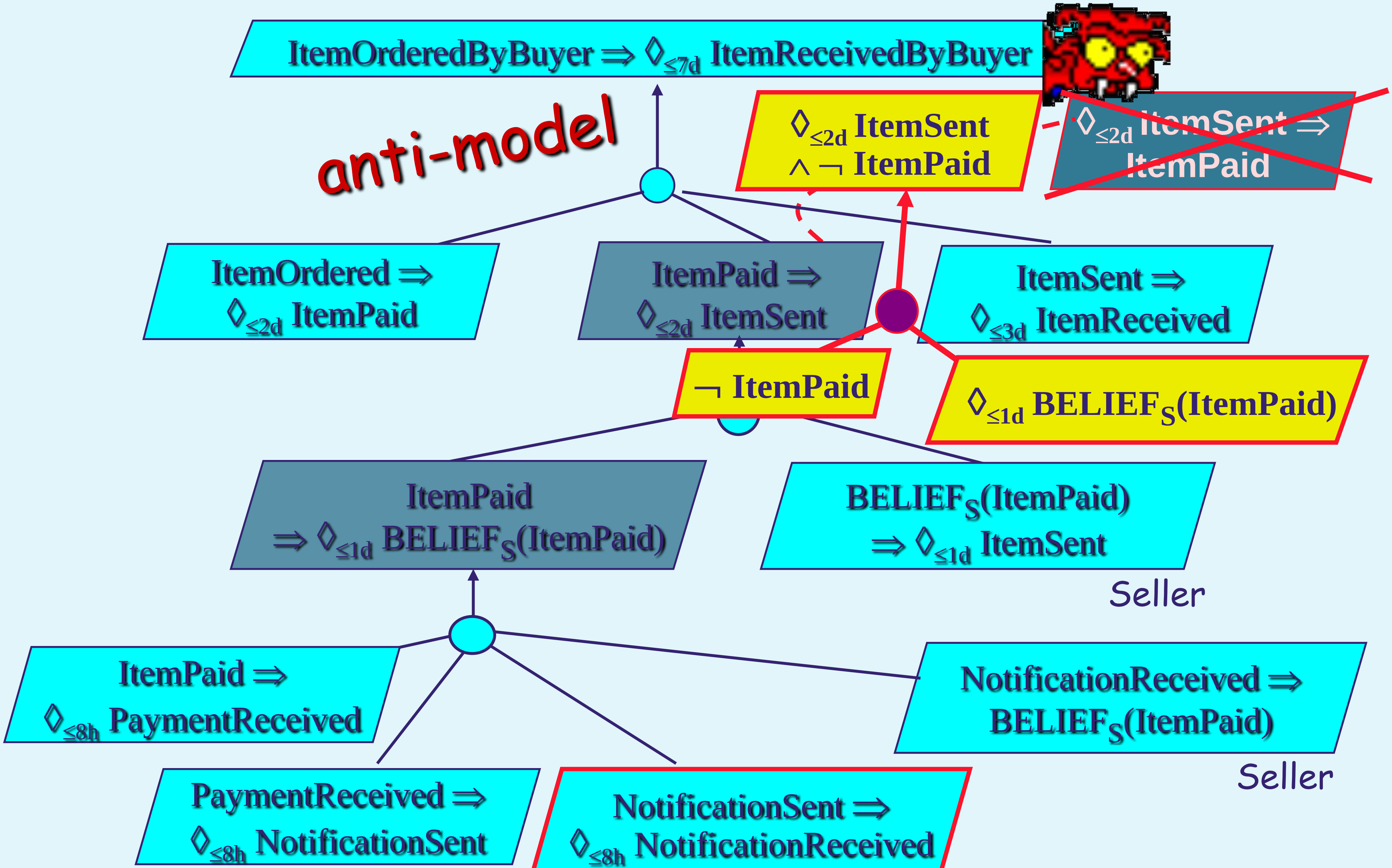
Online shopping: a security goal



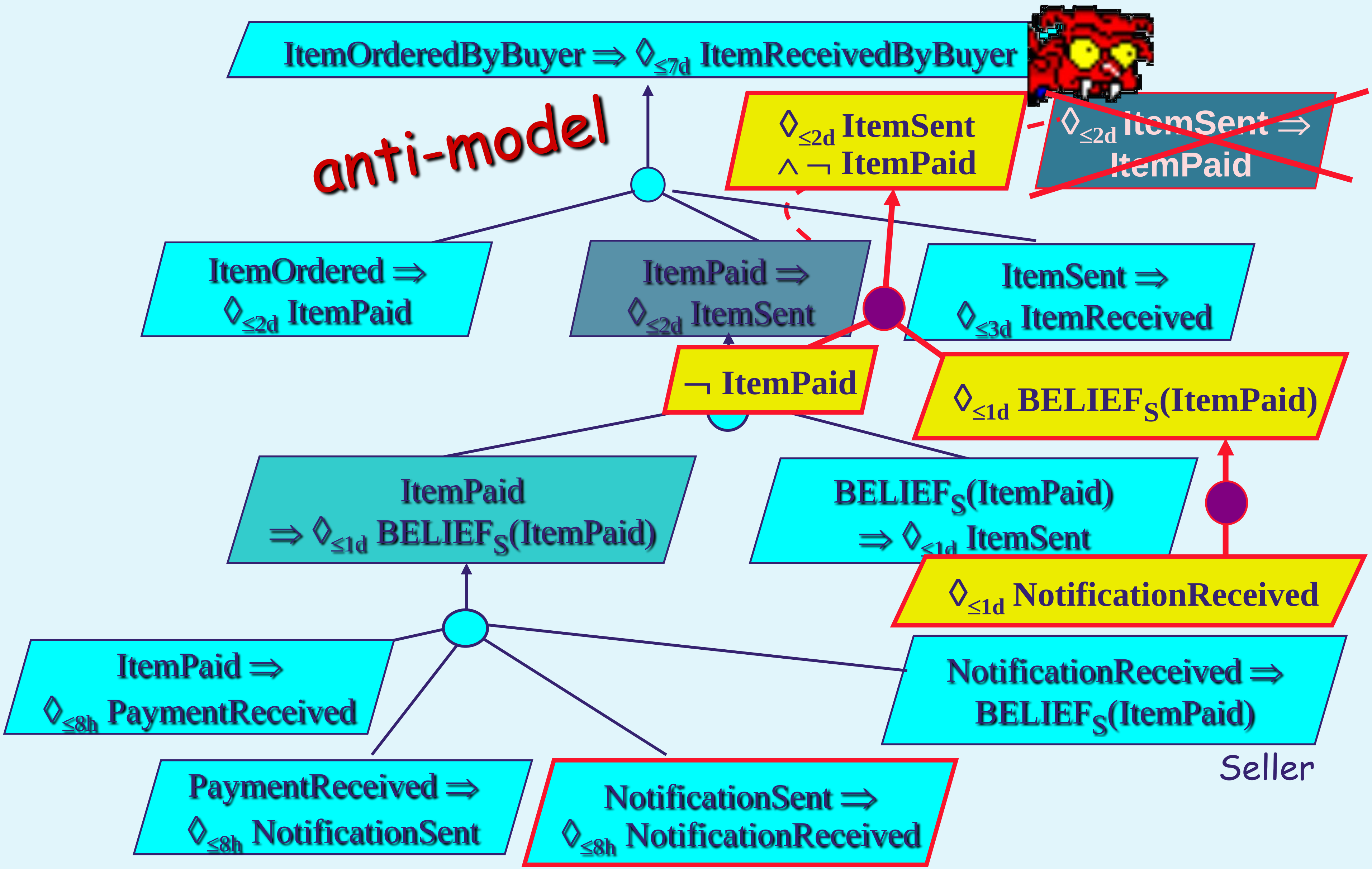
Online shopping: anti-goal



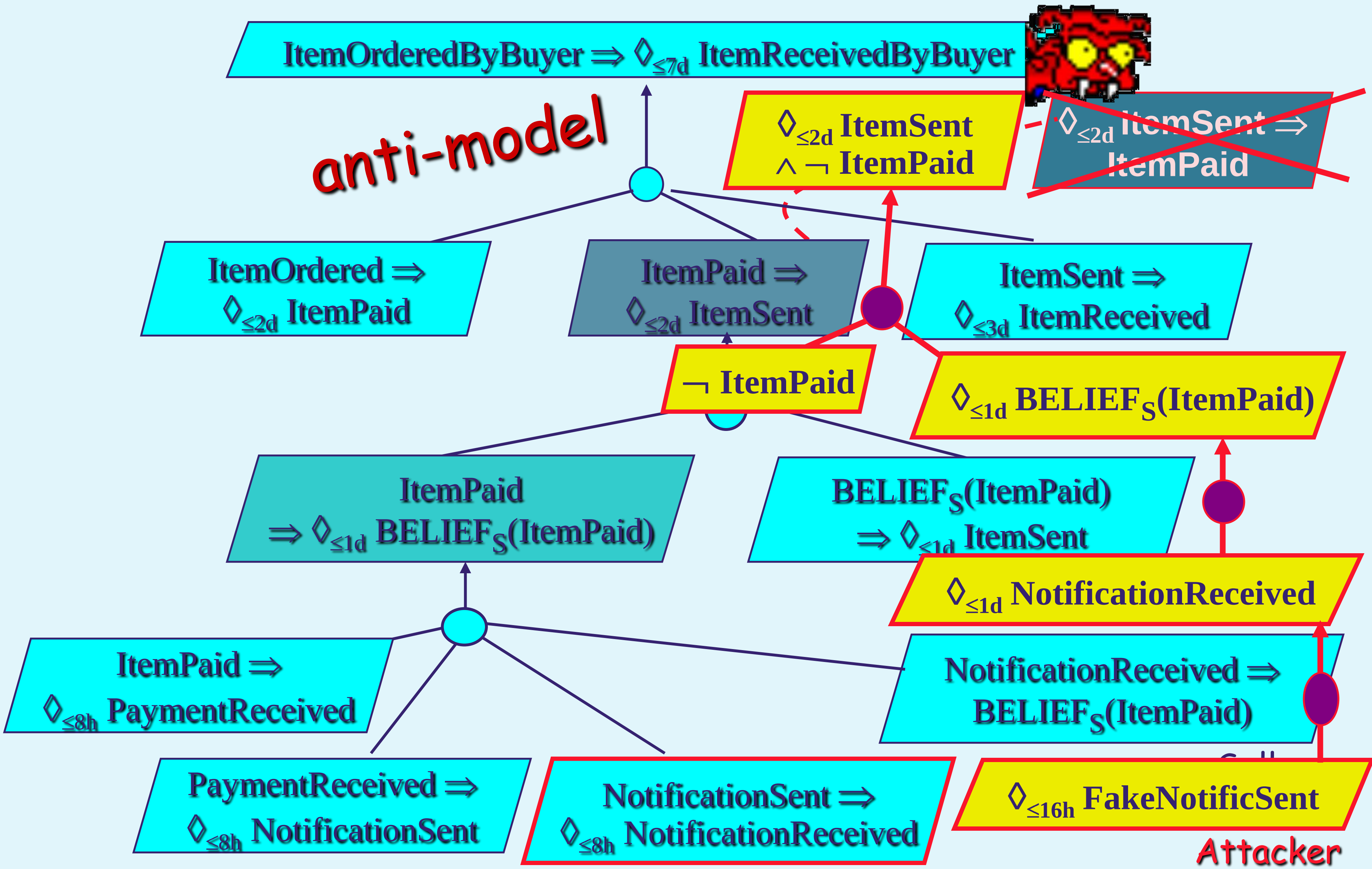
Online shopping: anti-goal model



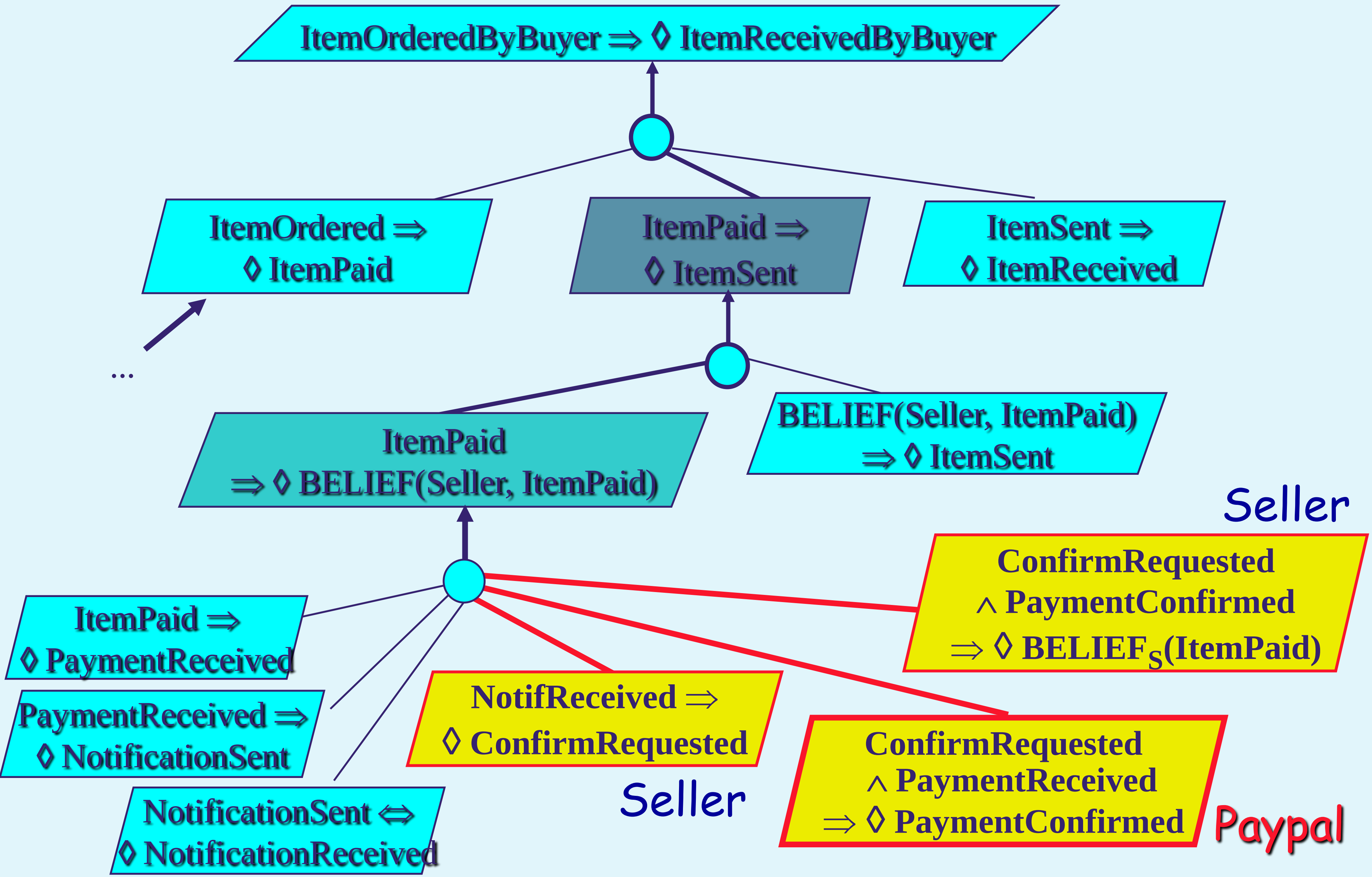
Online shopping: anti-goal model



Online shopping: anti-goal model



Online shopping: goal model with countermeasures



Application:

<u>S</u>ecurity of <u>A</u>ircraft in the <u>F</u>uture <u>E</u>uropean <u>E</u>nvironment

Threats against crew & passengers

(External threats)

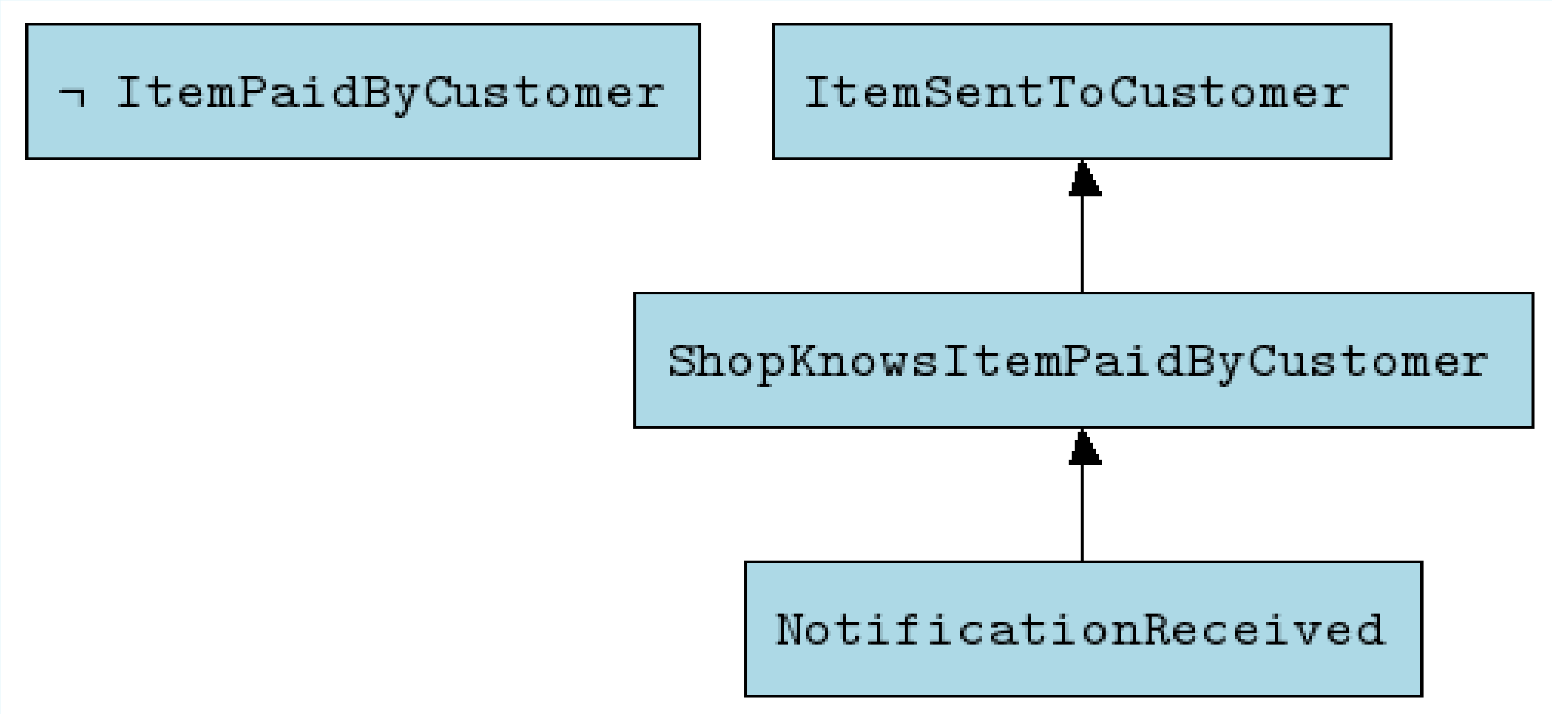
Threats from baggage area

- Modeling terrorist threats (anti-goal model)
- RE for on-board threat *detection* & *reaction* system

Automated synthesis of threat graphs

- Builds a proof showing realizability of anti-goal in view of attacker's capabilities & knowledge of environment
- Capabilities = Boolean state variables (atomic conditions that are monitorable/controllable)
- Based on BDD representation of anti-goal
- Weakens powerful macro-agent by removal of capabilities, following BDD state-variable ordering

Synthesizing attack graphs (plan generation)



Attacker **anti-goal**:

$\neg$ **ItemPaidByCustomer** $\wedge$ **ItemSentToCustomer**

Attacker **capabilities**:

Controls **ItemPaidByCustomer, NotificationReceived**

Monitors --

Course outline

- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  - A goal-oriented model building method in action
  - Checking goal refinements
  - Obstacle analysis for high-assurance applications
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals
  - Model checking against confidentiality requirements

Conflict analysis

- Divergence is most frequent case of conflicting goals, requirements or assumptions:

  potential logical inconsistency

- Goals $G_1, \ldots, G_n$ are divergent *iff*

  there exists a *boundary condition* B :

  $\{ B, \wedge_i G_i, Dom\} \models false$ *inconsistency*

  $\{ B, \wedge_{j \neq i} G_j, Dom\} \not\models false$ *minimality*

  exists system behavior S s.t. $S \models B$ *feasibility*

Divergence frequently involves security goals

Maintain[ReviewerAnonymity]:

$$\text{Reviews}(r, pap, rep) \land \text{AuthorOf}(a, pap)$$
$$\Rightarrow \Box \neg \text{Knows}(a, \text{Reviews}(r, pap, rep))$$

Achieve[ReviewIntegrity]:

$$\text{Reviews}(r, pap, rep) \land \text{AuthorOf}(a, pap)$$
$$\Rightarrow \Diamond \text{Gets}(a, rep', pap, r) \land rep' = rep$$

Boundary condition: $\Diamond \exists\, r, pap, a, rep, rep'$

$$\text{Reviews}(r, pap, rep) \land \text{AuthorOf}(a, pap)$$
$$\land \Diamond \text{Gets}(a, rep', pap, r) \land rep' = rep$$
$$\land \text{French}(r) \land \neg \exists\, r' \neq r: \text{Expert}(r') \land \text{French}(r')$$

Divergence frequently involves security goals

Maintain[ReviewerAnonymity]:

$\text{Reviews}(r, pap, rep) \land \text{AuthorOf}(a, pap)$

$\Rightarrow \Box \neg \text{Knows}(a, \text{Reviews}[r, pap, rep])$

Achieve[ReviewIntegrity]:

$\text{Reviews}(r, pap, rep) \land \text{AuthorOf}(a, pap)$

$\Rightarrow \Diamond \text{Gets}(a, rep', pap, r) \land rep' = rep$

Boundary condition: $\Diamond \exists r, pap, a, rep, rep'$

$\text{Reviews}(r, pap, rep) \land \text{AuthorOf}(a, pap)$

$\land \Diamond \text{Gets}(a, rep', pap, r) \land rep' = rep$

$\land \text{French}(r) \land \neg \exists r' \neq r: \text{Expert}(r') \land \text{French}(r')$

Conflict analysis (2)

- Detecting divergence:
  - by regression: derive B as precondition for $\neg G_i$ from $\{\wedge_{j \neq i} G_j, Dom\}$
  - by use of formal conflict patterns
- Resolving divergence: resolution operators
  - avoid boundary condition: $\Box \neg B$
  - restore divergent goals: $B \Rightarrow \Diamond \wedge_i G_i$
  - anticipate conflict: $P \Rightarrow \Diamond_{\leq T} \neg P$
  - weaken goals, specialize objects, etc.

Deriving boundary condition for conflict

By regression:

$AtStation \wedge \circ \neg AtStation \Rightarrow DoorsClosed \; \mathcal{W} \; AtNext$

$\bullet \, (Stopped \wedge Alarm) \Rightarrow DoorsOpen$

Deriving boundary condition for conflict

By regression:

$AtStation \wedge \circ \neg AtStation \Rightarrow DoorsClosed \; \mathcal{W} \; AtNext$

$\bullet\, (Stopped \wedge Alarm) \Rightarrow DoorsOpen$

→ **negate G1:**

$AtStation \wedge \circ \neg AtStation \wedge$

$\neg AtNext \; \mathcal{U} \; (DoorsOpen \wedge \neg AtNext)$

Deriving boundary condition for conflict

By regression:

$AtStation \wedge \circ \neg AtStation \Rightarrow DoorsClosed\ \mathcal{W}\ AtNext$

$\bullet\,(Stopped \wedge Alarm) \Rightarrow DoorsOpen$

→ **negate G1:**

$AtStation \wedge \circ \neg AtStation \wedge$

$\neg AtNext\ \mathcal{U}\ (DoorsOpen \wedge \neg AtNext)$

→ **regress ¬ G1 through G2:**

$AtStation \wedge \circ \neg AtStation$

$\wedge \neg AtNext\ \mathcal{U}\ (\bullet\, Stopped \wedge \bullet\, Alarm \wedge \neg AtNext)$

boundary condition for conflict

Course outline

- Goal-oriented RE for high-assurance applications
  - Modeling goals, objects, agents, operations, behaviors
  - A goal-oriented model building method in action
  - Checking goal refinements
  - Obstacle analysis for high-assurance applications
- Engineering security requirements
  - Security goals and their specification
  - Threat analysis for model consolidation
  - Analyzing conflicts among security goals

  - Model checking against confidentiality requirements

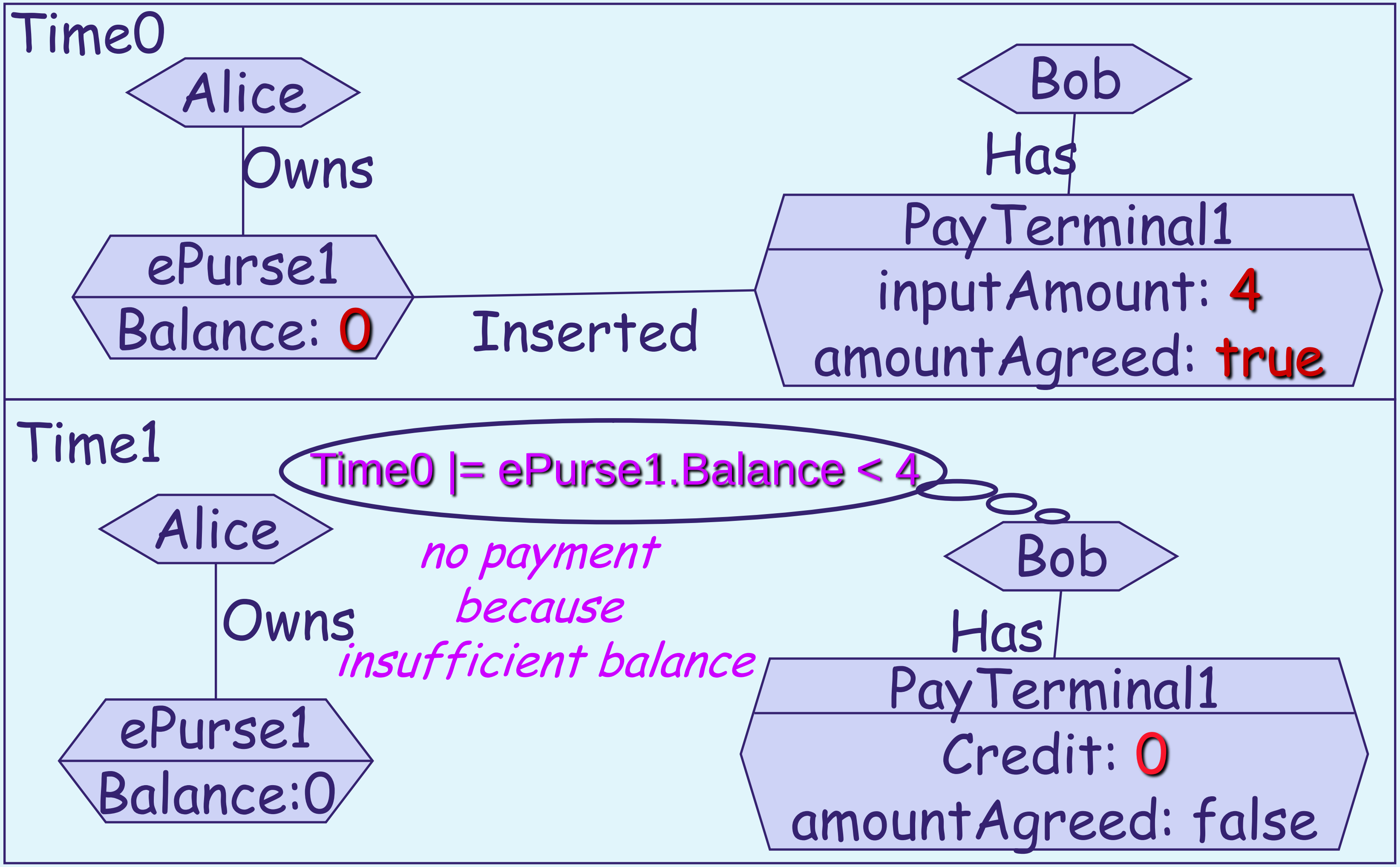
CONCHITA: checking requirements models against confidentiality claims

- Given ...
  - an object model (entities, associations, agents)
  - a list of requirements
  - assumed confidentiality requirements
  - claimed confidentiality requirements

  } specified with patterns
- Find a finite trace ...
  - satisfying the requirements
  - where an agent instance can acquire knowledge that violates one of the confidentiality claims
- + Explain how the agent acquired this knowledge

Implementation:
Bounded Model Checking, Finite instantiation,
CSP solver (efficient arithmetic and search space pruning)

Running CONCHITA on e-Purse system: trace leading to information disclosure



+ explanation = knowledge fragments used in the deduction

Example of axioms about unauthorized agent (UA)

Maximal Input *at any time, UA knows the value of every non confidential variable*

Ex: seller knows the amount that is entered in the terminal

Example of axioms about unauthorized agent (UA)

Maximal Input *at any time, UA knows the value of every non confidential variable*

Perfect System knowledge *UA knows all the requirements the software implements and all the properties of the domain.*

Ex: the seller knows that payment is denied in case of insufficient balance.

Example of axioms about unauthorized agent (UA)

Maximal Input *at any time, UA know the value of every non confidential variable*

Perfect System knowledge *UAs know all the requirements the software implements and all the properties of the domain.*

Perfect Recall *UAs always remember facts and properties they used to know in the past.*

Ex: at time1, the seller remembers the entered amount, the insertion of the e-Purse, …

Conclusion

- Rich models are essential for HA applications
  - multiple dimensions: intentional, structural, responsibility, operational, behavioral
  - software + environment (e.g., humans, devices, other software, mother Nature, attacker, attackee)

    start thinking about high assurance at RE time

  - alternative refinements, assignments, resolutions
  - seamless transition from high-level concerns to operational requirements

Conclusion (2)

- The building of such models is hard & critical; should therefore be guided by methods...
  - systematic
  - top-down + bottom-up
  - incremental
  - supporting the analysis of partial models

Conclusion (3)

- Goal-based reasoning is central for…
  - model building & elaboration of requirements
  - exploration & evaluation of alternatives
  - conflict management
  - anticipation of hazards and threats
    (requirements-level exception handling)

Conclusion (4)

- Goal completeness can be achieved through multiple means ...
  - refinement checking =>
    missing subgoals, subobstacles, threats/vulnerabilities
  - obstacle/threat analysis => countermeasure goals
  - animation (not discussed here)

Conclusion (5)

- Be pessimistic from beginning about software *and environment*

  hazards, threats, conflicts

- Benefits of multi-button framework
  - semi-formal ...

    for modeling, navigation, traceability
  - formal, when and where needed ...

    for precise, incremental reasoning on model pieces

Thanks ...

- To the KAOS crew *at UCL, CETIC & RESPECT-IT* as *researchers, consultants,* or *tool developers*

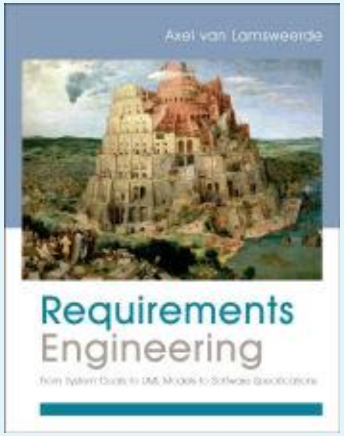
  C. Damas, A. Dardenne, R. Darimont,

  R. De Landtsheer, E. Delor, B. Lambeau, E. Letier,

  P. Massonet, C. Ponsard, A. Rifaut, H. Tran Van

- To Steve Fickas and his group at Univ. Oregon

- To the EU & Region of Wallonia for significant funding of those efforts



More information available ...

- ... on the method & associated techniques in:

  A. van Lamsweerde, *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2008.

  www.info.ucl.ac.be/~avl

- ... on tools at:

  http://www.objectiver.com

  http://faust.cetic.be

Relevant papers

A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective". *Keynote Paper, Proc. ICSE'2000 - Intl Conf on Software Engineering*, June 2000, IEEE CS Press, pp. 5-19.

R. Darimont & A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration". *Proc. FSE-4 - Fourth ACM Conf on Foundations of Software Engineering*, San Francisco, Oct. 1996, 179-190.

E. Letier & A. van Lamsweerde, "Agent-Based Tactics for Goal-Oriented Requirements Elaboration", *Proc. ICSE'2002 - 24th Intl Conf on Software Engineering*, Orlando, May 2002, IEEE CS Press, 83-93.

A. van Lamsweerde & E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", *IEEE Transactions on Software Engineering, Special Issue on Exception Handling*, Vol. 26, No. 10, October 2000.

E. Letier & A. van Lamsweerde, "Deriving Operational Software Specifications from System Goals", *Proc FSE'2002 - 10th ACM Conf on the Foundations of Software Engineering*, Charleston (South Carolina), November 2002.

E. Letier and A. van Lamsweerde, "Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering", *Proc FSE'04, 12th ACM Intl Symp. Foundations of Software Engineering*, Newport Beach (CA), Nov. 2004.

Relevant papers (2)

A. van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models", *Proc ICSE'04 - 26th Intl Conf on Software Engineering*, Edinburgh, May. 2004, ACM-IEEE, 148-157.

R. De Landtsheer & A. van Lamsweerde, "Reasoning about Confidentiality at Requirements Engineering Time", *Proc. ESEC/FSE'05, 13th ACM Intl Symp. on the Foundations of Software Engineering*, Lisbon, Sept. 2005, 41-49.

A. van Lamsweerde, R. Darimont & E. Letier, Managing Conflicts in Goal-Driven Rquirements Engineering, *IEEE Transactions on Software Engineering,* Vol. 24 No. 11, November 1998, pp. 908 - 926.

C. Damas, B. Lambeau, P. Dupont & A. van Lamsweerde, "Generating Annotated Behavior Models from End-User Scenarios", *IEEE Transactions on Software Engineering*, Vol. 31, No. 12, December 2005, 1056-1073.

H. Tran Van, A. van Lamsweerde, P. Massonet, Ch. Ponsard, "Goal-Oriented Requirements Animation", *Proc RE'04, 12th IEEE Joint Intl Requirements Engineering Conference*, Kyoto, Sept. 2004, 218-228.