KAOS in Action: The BART System

Emmanuel Letier and Axel van Lamsweerde

Dept. Ingénierie Informatique, Univ. Louvain
B-1348 Louvain-la-Neuve (Belgium)
{eletier, avl}@info.ucl.ac.be

Outline

Introduction
- Goal-Oriented RE with KAOS

The BART Case Study
- Identifying Goals from Initial Document
- Formalizing Goals and Identifying Objects
- Elaborating the Goal Structure
- Identifying Agents and Responsibilities
- Deriving Monitored and Controlled Quantities
- Exploring Alternative Responsibility Assignments
- Identifying Operations
- Operationalizing Goals through Strenghtened Operations

- Conflict Analysis: an Example

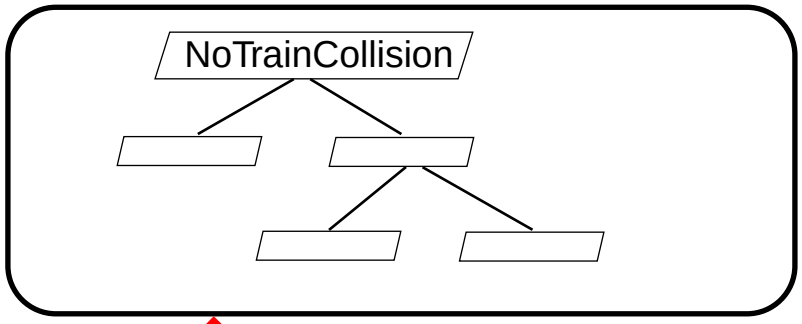
- Obstacle Analysis
  - Obstacle Generation
  - Obstacle Resolution

Conclusion

Goal-Oriented Requirement Engineering with KAOS

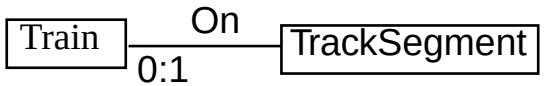
2-level language:
semantic net level
formal assertions

Goal Model

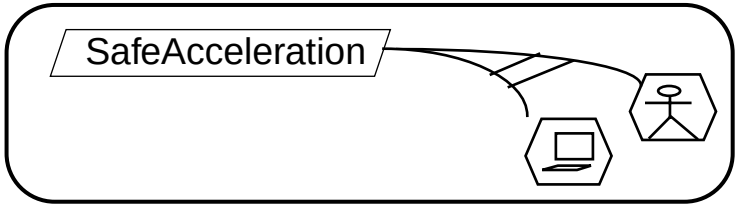


AND/OR goal refinement
Goals, Requirements, Assumptions
+ DomProps

Object Model



Responsibility Model

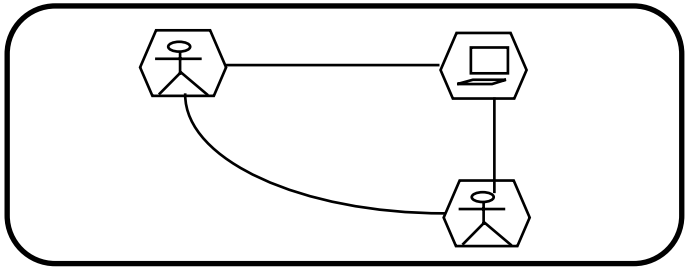


Or responsibilities

Dom Pre/Post
Req Pre/ Trigger / Post

software agents
+ environment agents

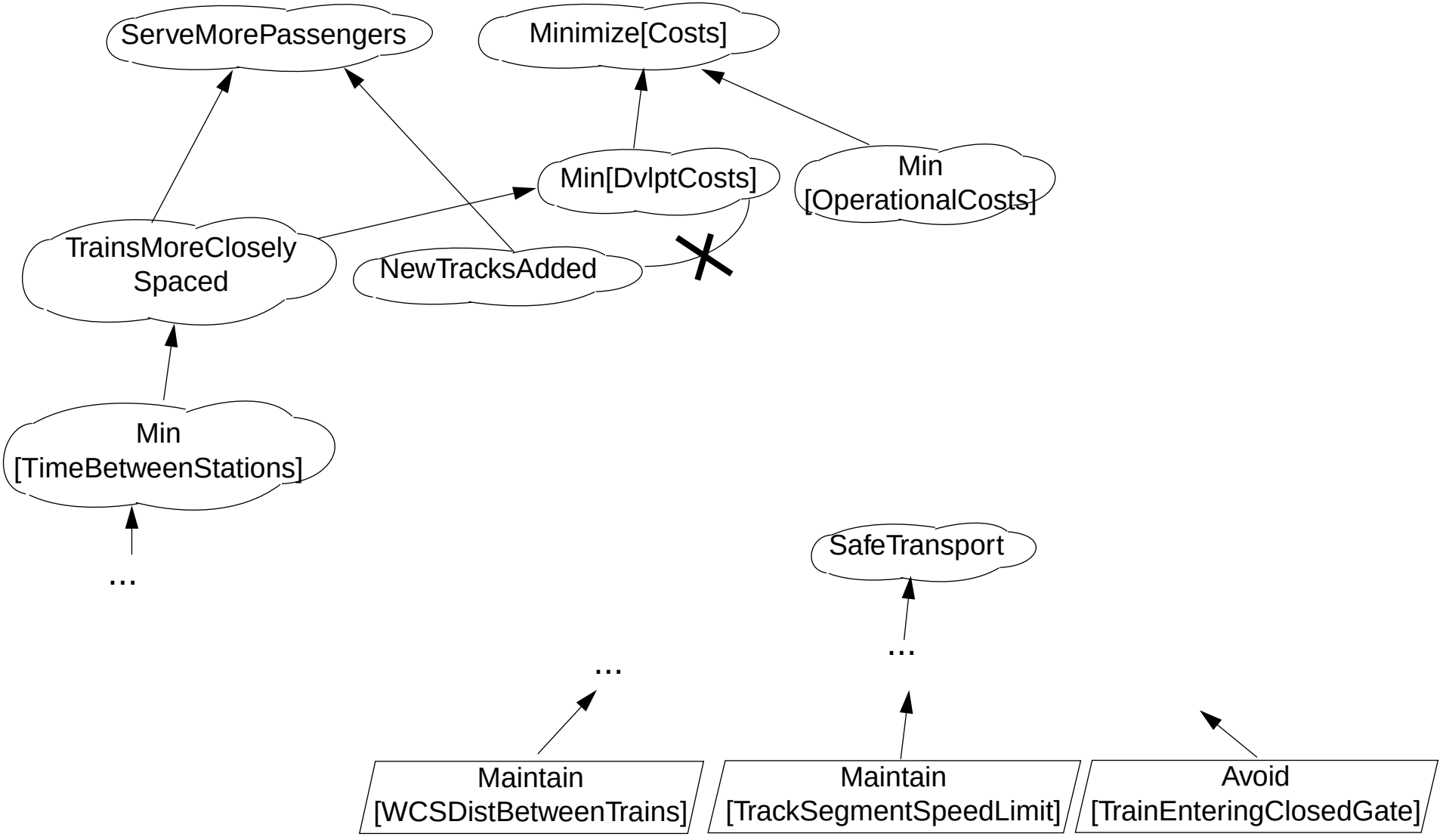
Agent Interface Model



Operation Model

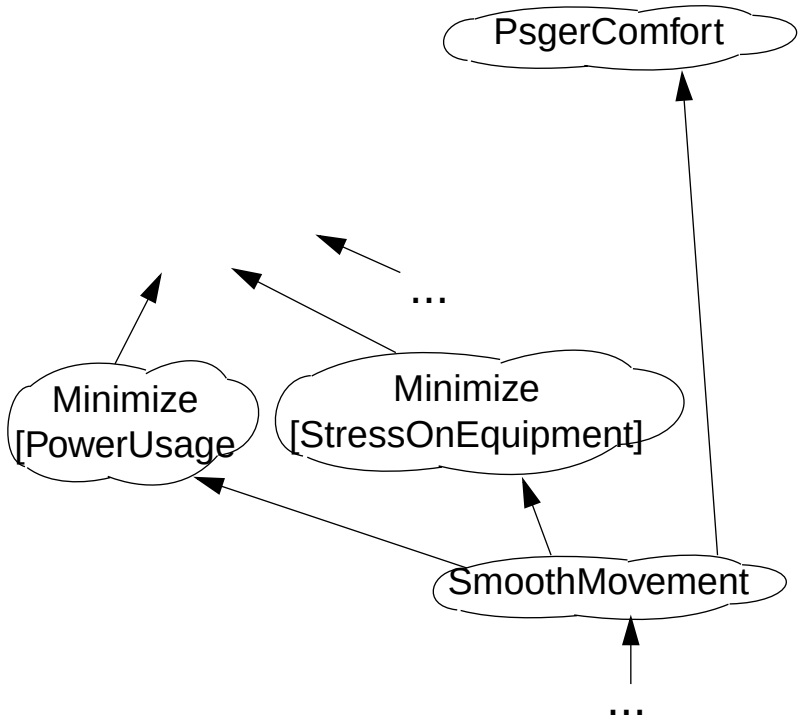
Operation SendComand
DomPre ¬ Sent(m, tr)
DomPost Sent(m, tr)
ReqPostFor SafeAceleration
m.Acceleration ≤ F(tr.Loc, tr.Speed, ...)

Goal Identification From Initial Document

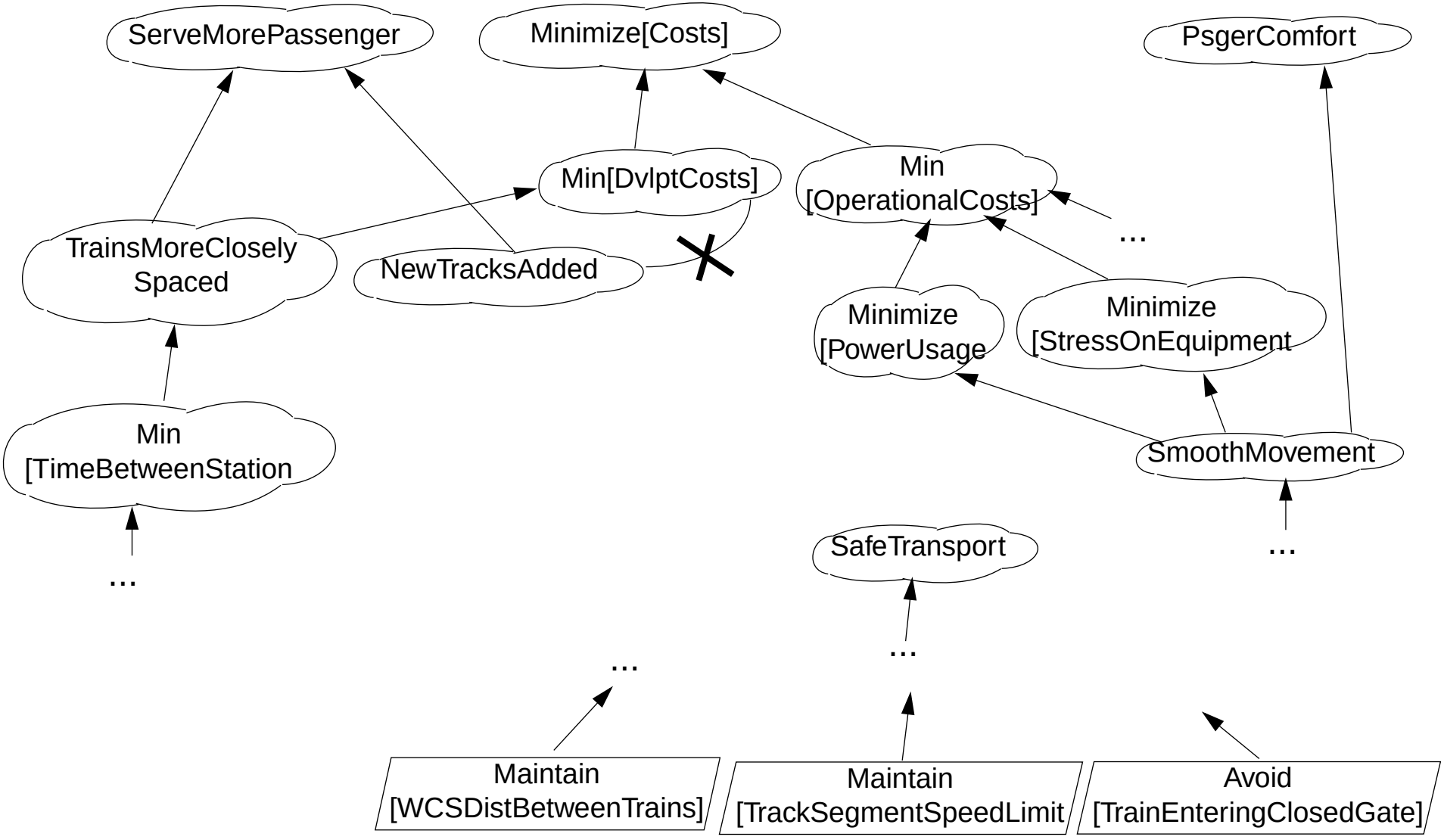


Initial document: see http://www.hcecs.sandia.gov/bart.htm

==> **Further goals identified by asking WHY and HOW questions**


PsgerComfort
...
Minimize
[PowerUsage
Minimize
[StressOnEquipment]
SmoothMovement
...

Goal Identification From Initial Statement



==> Further Goals identified by asking WHY and HOW questions

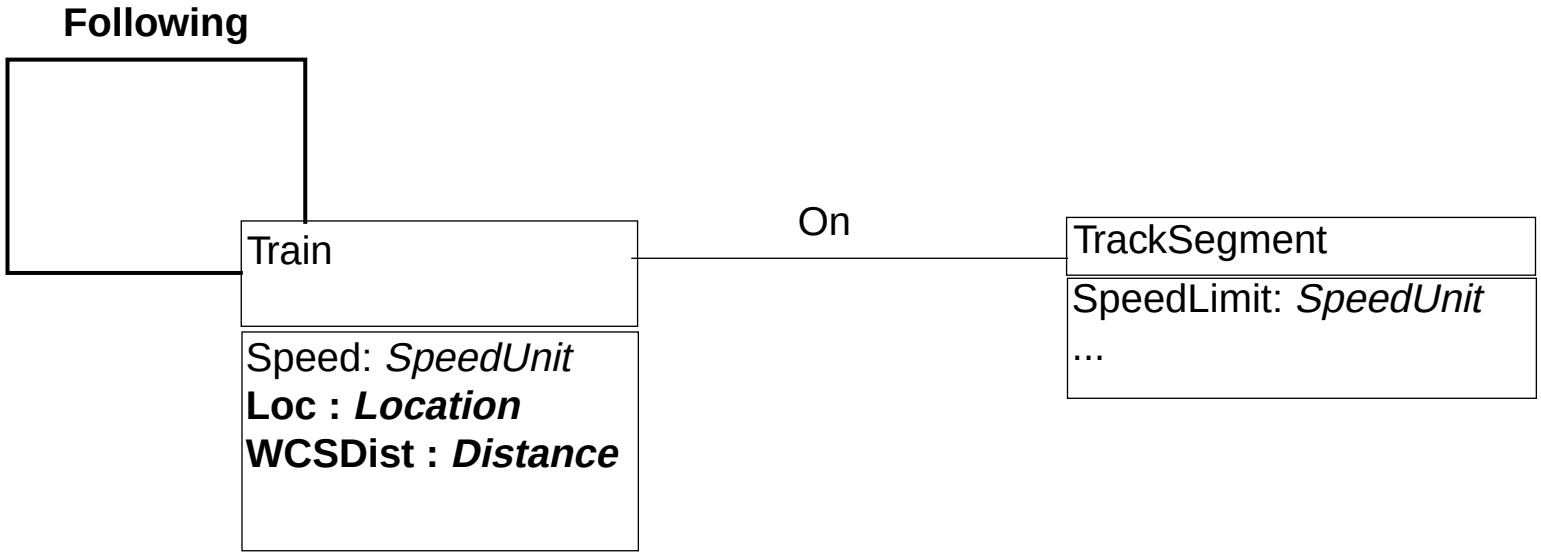
Formalizing Goals and Identifying Objects (1)

Goal Maintain[TrackSegmentSpeedLimit]
Definition A train should stay below the maximum speed the track segment can handle.
FormalDef $\forall$ tr: Train, s: TrackSegment : On(tr, s) $\Rightarrow$ tr.Speed $\leq$

Goal-oriented vs. Object-oriented RE
Goals provide precise criterion for identification of objects, relationships, attributes

Formalizing Goals and Identifying Objects (2)

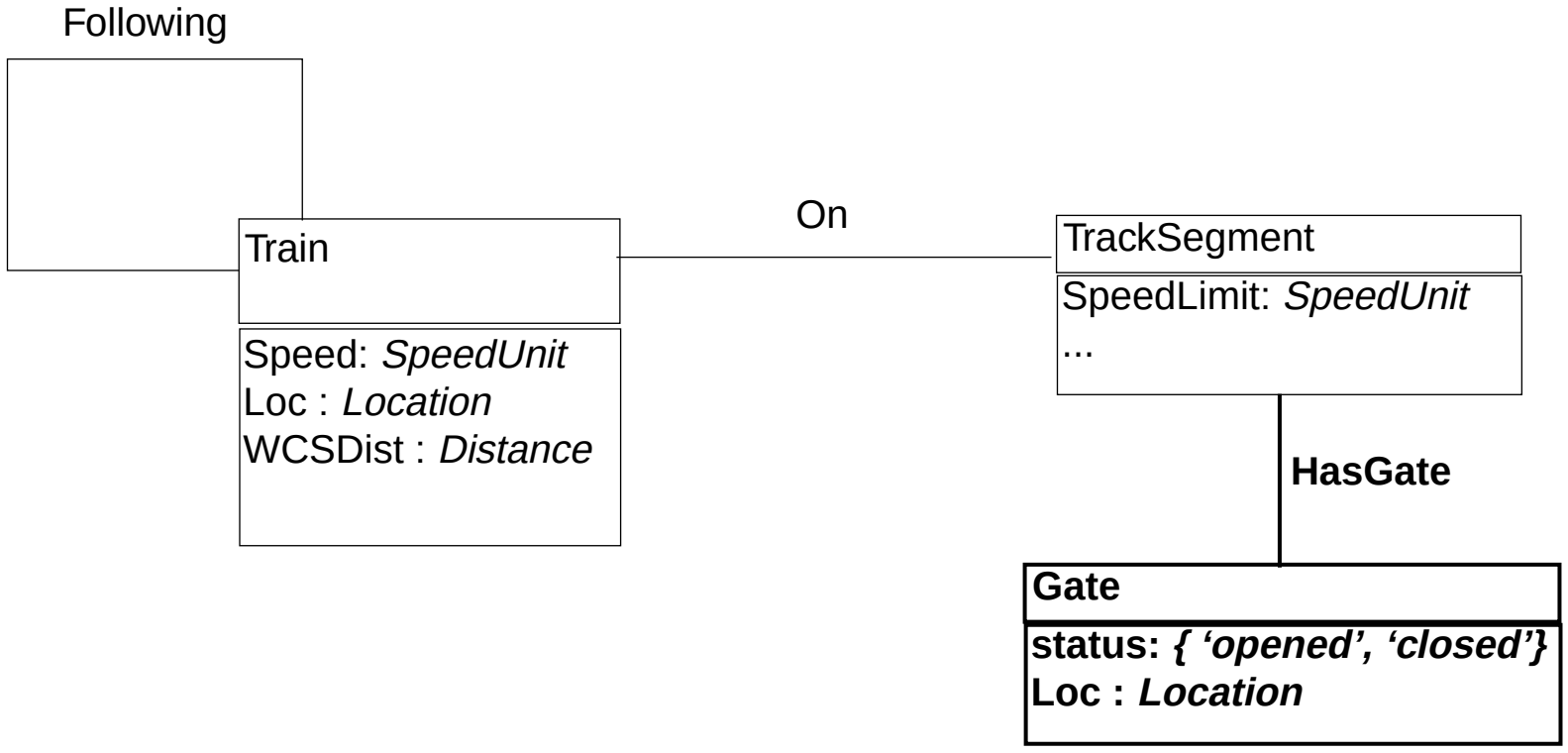
Goal Maintain[WCSDistBetweenTrains]
Definition A train should never get so close to a train in front so that if the train in front stops suddenly (e.g., derailment) the next train would hit it.
FormalDef ∀ tr1, tr2: Train :
Following(tr1, tr2)⇒ tr1.Loc - tr2.Loc > tr1.WCSDist



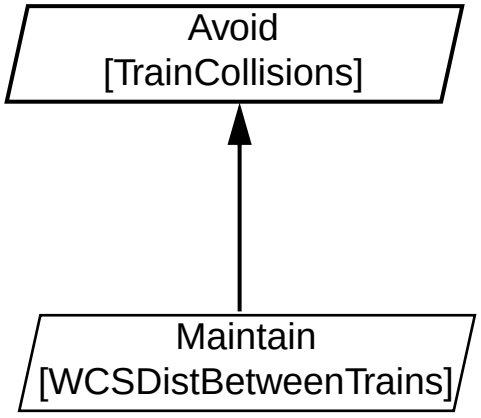
Formalizing Goals and Identifying Objects (3)

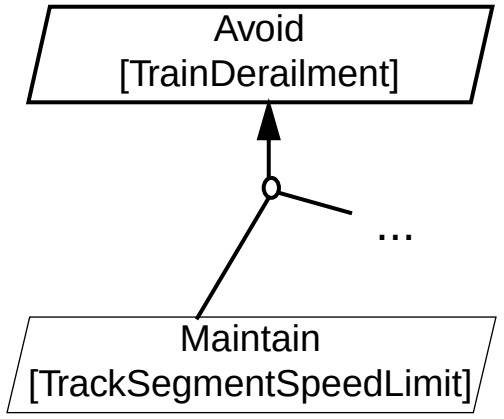
Goal Avoid[TrainEnteringClosedGate]
Definition A train should not enter a closed gate if it can,
(i.e. if it is possible for the train to stop before the gate)
FormalDef ∀ tr: Train, g: Gate, s: TrackSegment:
g.status = 'closed' ***Since*** tr.Loc - g.Loc > tr.WCSDist
∧ HasGate(s, g)
⇒
¬ @ On(tr, s)

"*Since*":
"*Until*" in the past



Eliciting New Goals : WHY Questions (1)

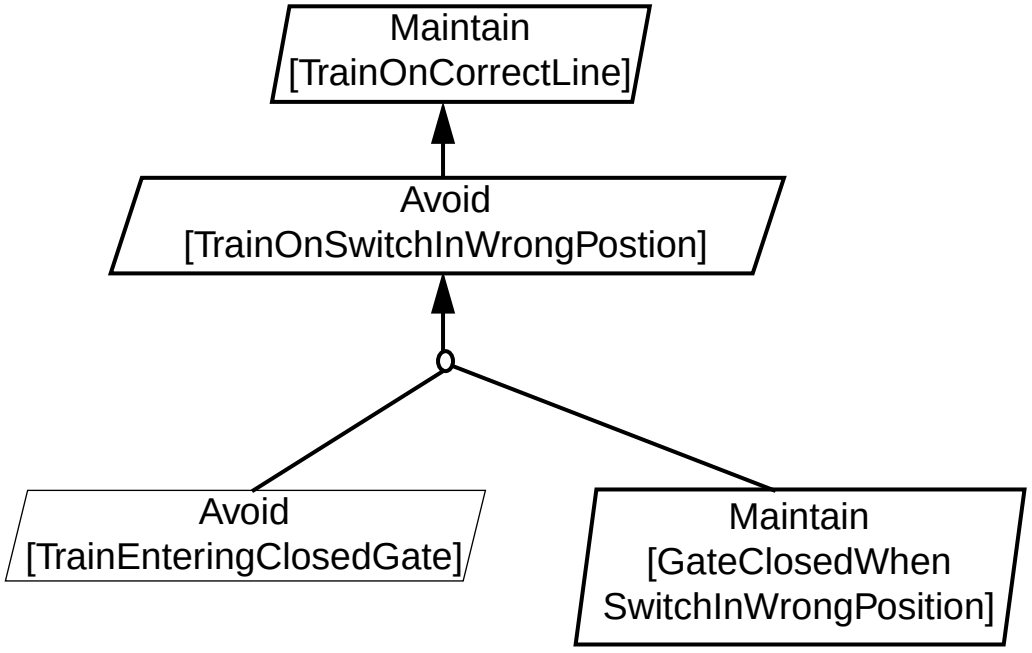




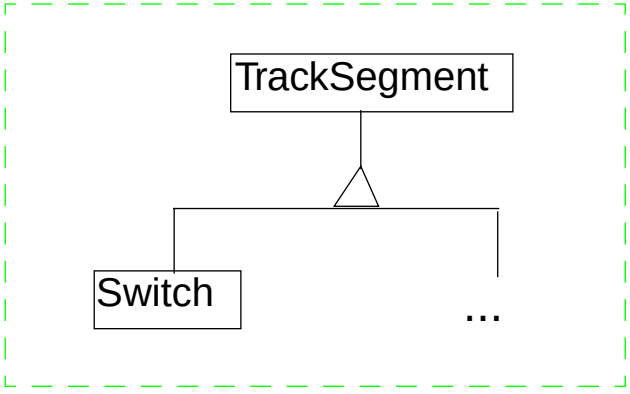
Goal Avoid[TrainCollisions]
Definition Trains should nerver collide
FormalDef ∀ tr1, tr2: Train :
❑ ¬ Collision(tr1, tr2)

Goal Avoid[TrainDerailent]
Definition Trains should never derail
FormalDef ∀ tr: Train :
❑ ¬ Derailment(tr)

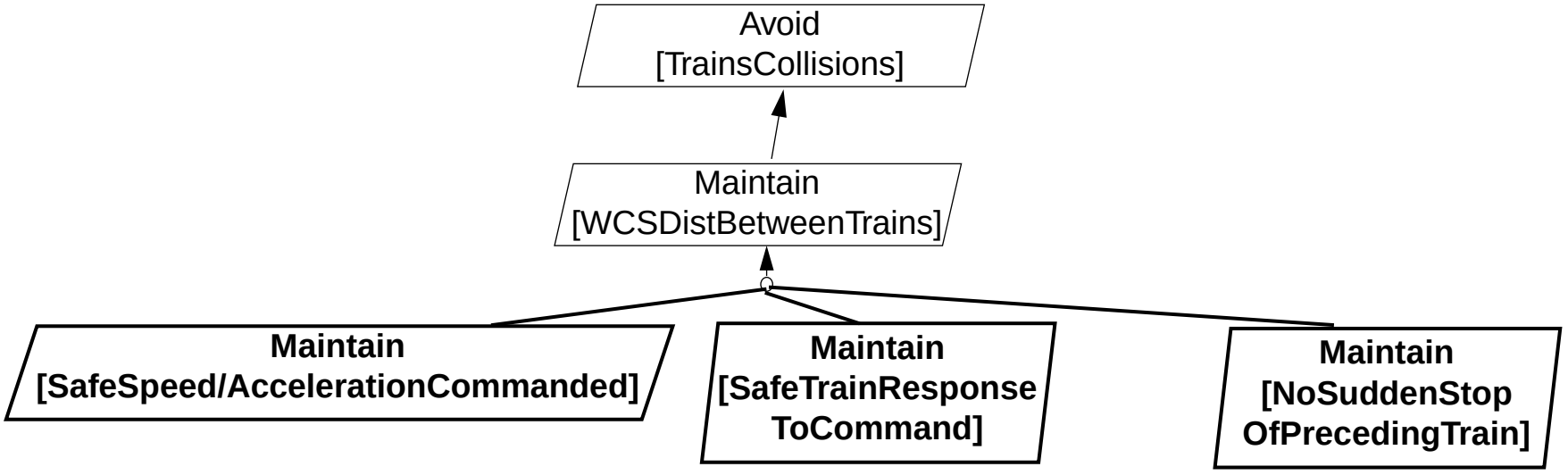
Eliciting New Goals : WHY Questions (2)



Goal Avoid[TrainOnSwitchInWrongPostion]
Definition When a train is on a switch, the switch should be in the direction of travel of the train
FormalDef ∀ tr: Train, sw: Switch:
On(tr, sw) ⇒ sw.Position = tr.Direction



Eliciting New Goals: HOW Questions (1)



WCSDist : the physical Worst-Case Stopping Distance based on the *physical* speed of the train

Following(tr1, tr2)
⇒
tr1.Loc -tr2.Loc > tr1.WCSDist

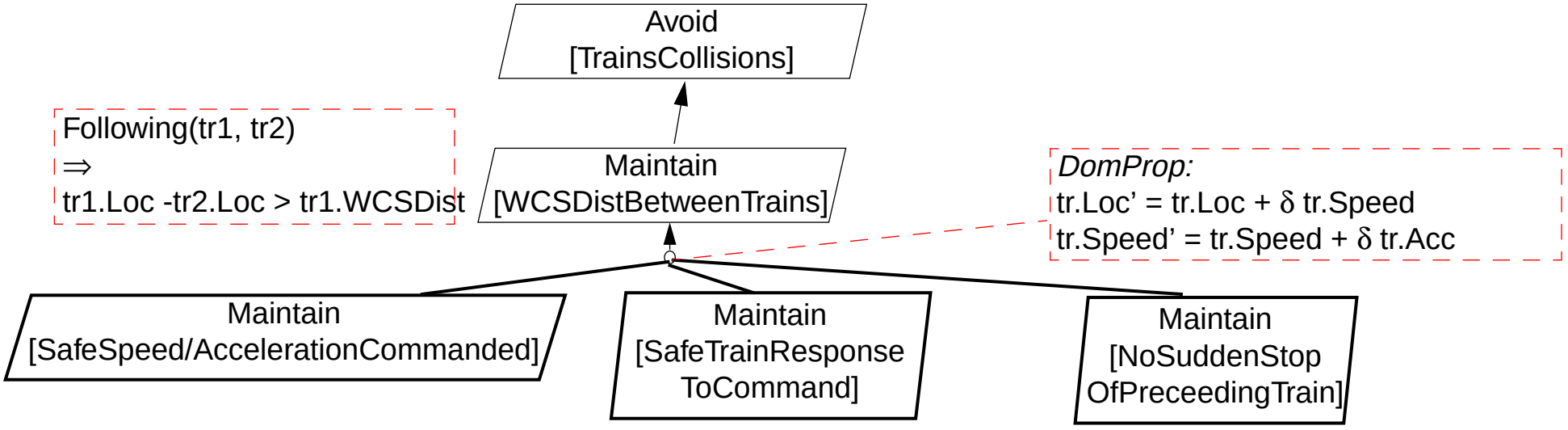
DomProp:
tr.Loc' = tr.Loc + δ tr.Speed
tr.Speed' = tr.Speed + δ tr.acc

Following(tr1, tr2)
⇒
$tr1.Acc_{CM}' \leq F(tr1.Loc, tr2.Loc, tr1.Speed, tr2.Speed)$
∧
$tr1.Speed_{CM}' > tr1.Speed$ (!)

∀ tr: Train
$tr.Acc_{CM} \geq 0 \Rightarrow tr.Acc' \leq tr.Acc_{CM}$
∧
$\blacksquare_{\leq MCdelay}\, tr.Acc_{CM} < 0 \Rightarrow tr.Acc' \leq 0$
∧
$tr.Speed \leq tr.Speed \Rightarrow tr.Speed' \leq tr.Speed_{CM}$

∀ tr2: Train
❑
tr2.Speed' ≥ tr2.Speed - δ MaxBrakeRate

Eliciting New Goals: HOW Questions (1)



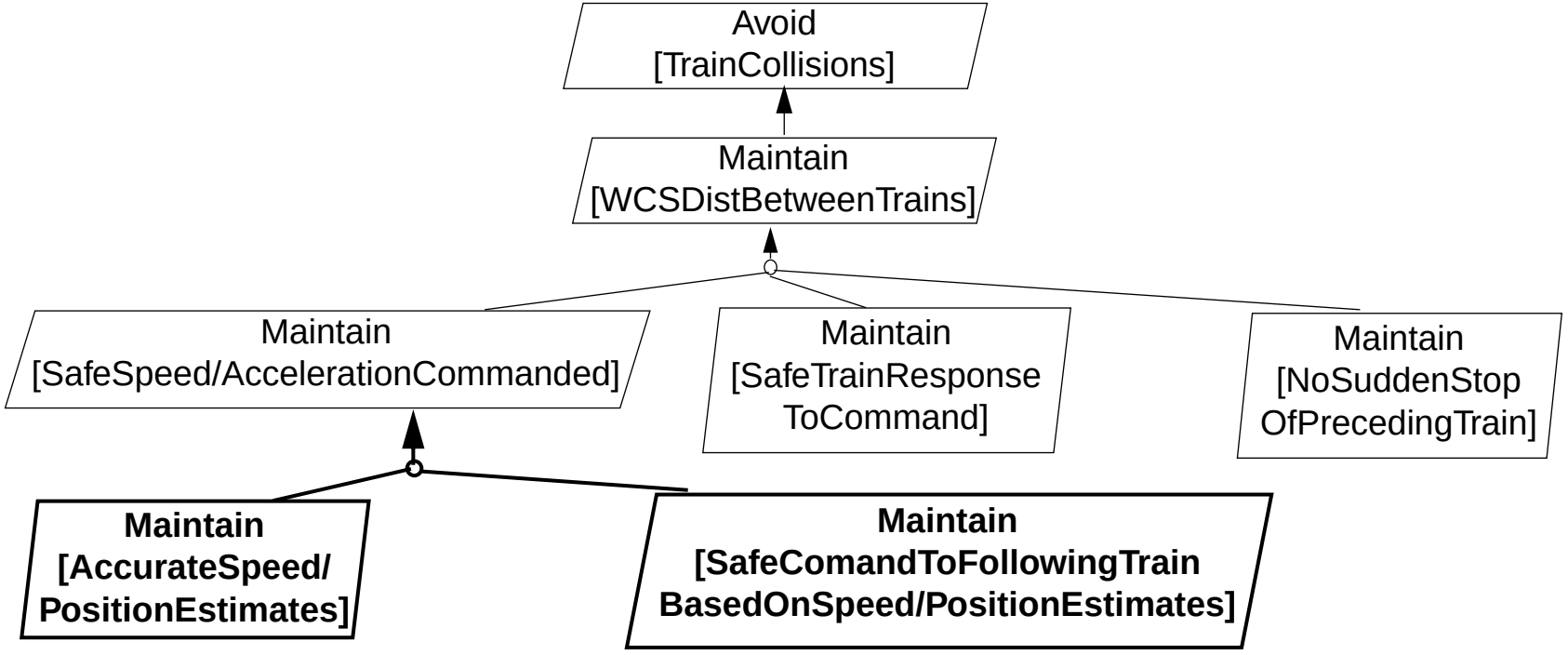
Following(tr1, tr2)
⇒
$tr1.Acc_{CM}' \leq F(tr1.Loc, tr2.Loc, tr1.Speed, tr2.Speed)$
∧
$tr1.Speed_{CM}' > tr1.Speed$ (!)

∀ tr: Train
$tr.Acc_{CM} \geq 0 \Rightarrow tr.Acc' \leq tr.Acc_{CM}$
∧
$\blacksquare_{\leq MCdelay}\ tr.Acc_{CM} < 0 \Rightarrow tr.Acc' \leq 0$
∧
$tr.Speed \leq tr.Speed_{CM} \Rightarrow tr.Speed' \leq tr.Speed_{CM}$

∀ tr2: Train
❑
tr2.Speed' ≥ tr2.Speed - δ MaxBrakeRate

WCSDist : the physical Worst-Case Stopping Distance based on the *physical* speed of the train

Eliciting New Goals: HOW Questions (2)

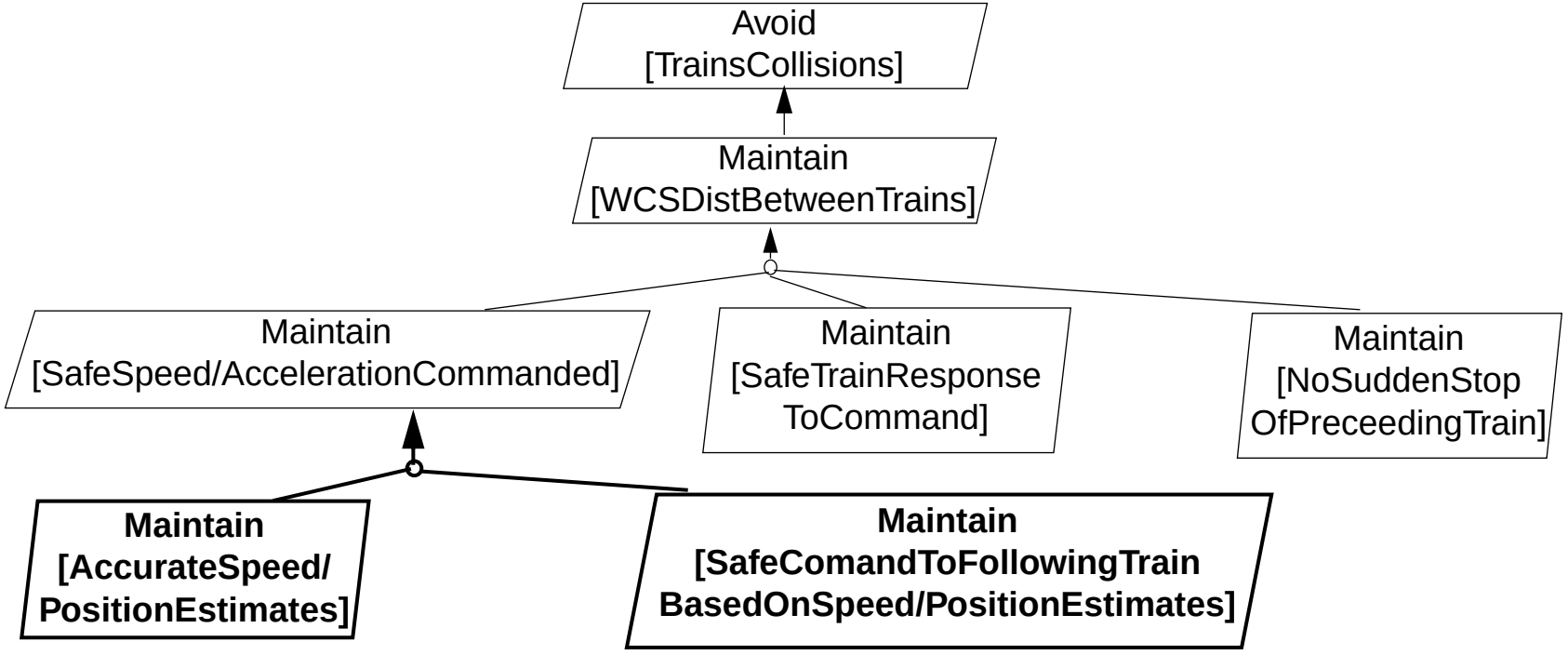


$\forall$ tr: Train, $\exists!$ ti: TrainInfo: Tracking(ti,tr)

$\forall$ tr: Train, ti: TrainInfo:
Tracking(ti, tr)
$\Rightarrow \Box$
ti.Loc - ti.Ldev $\leq$ tr.Loc $\leq$ ti.Loc +ti.Ldev
$\wedge$
ti.Speed - ti.Sdev $\leq$ tr.Speed $\leq$ ti.Speed +Sdev

FollowingInfo(ti1, ti2)
$\wedge$ Tracking(ti1, tr1) $\wedge$ Tracking(ti2, tr2)
$\Rightarrow$
$tr1.Acc_{CM}' \leq F$ (**ti1.Loc+ ti1.LDev**, **ti2.Loc - ti2.Ldev**,
ti1.Speed + ti1.Sdev, **ti2.Speed - ti2.Sdev**)
$\wedge$
$tr1.Speed_{CM}' >$ **ti1.Speed+ ti1.Sdev**

Eliciting New Goals: HOW Questions (2)

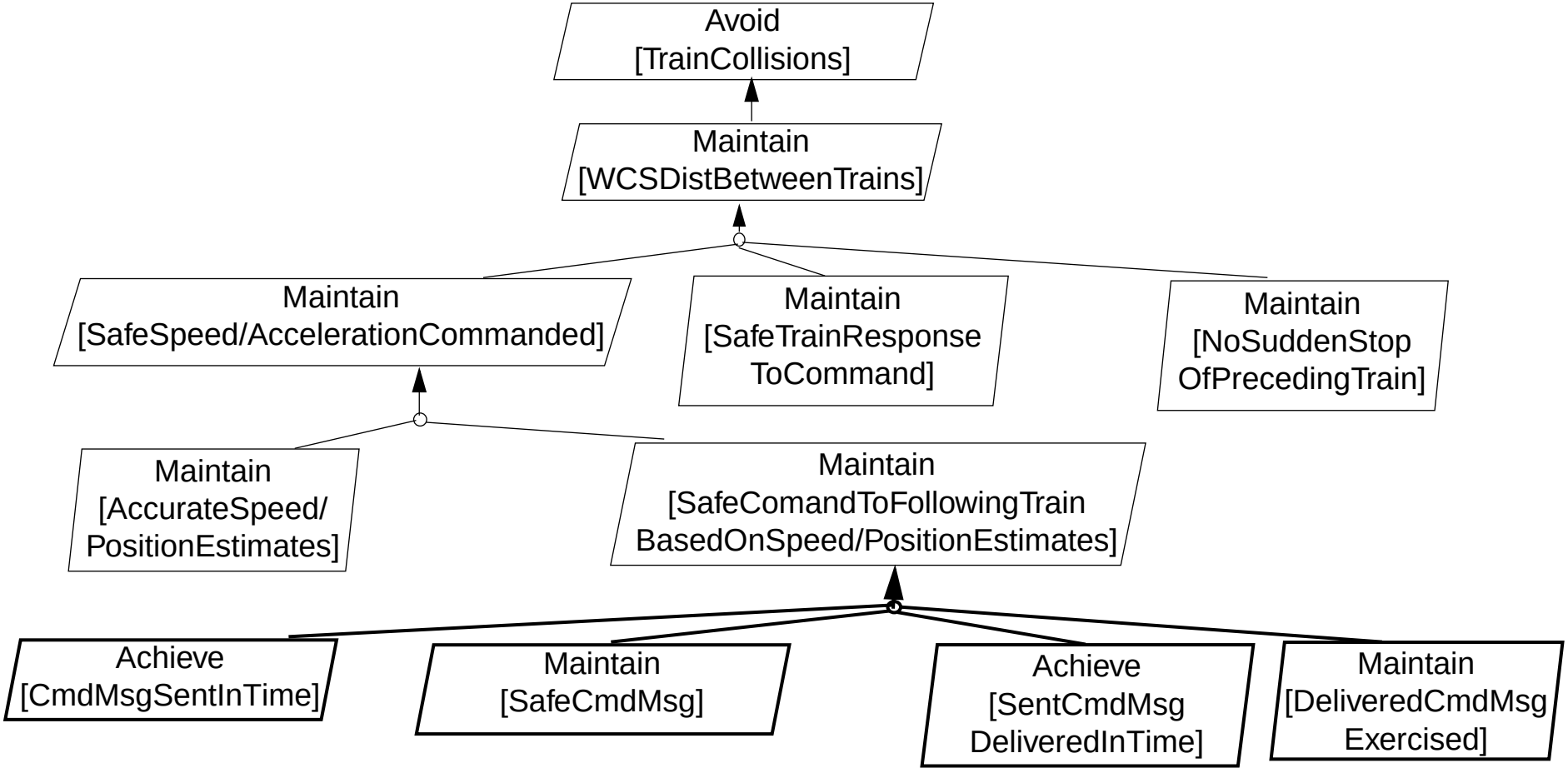


$\forall$ tr: Train, $\exists!$ ti: TrainInfo: Tracking(ti,tr)

$\forall$ tr: Train, ti: TrainInfo:
Tracking(ti, tr)
$\Rightarrow$ ❑
ti.Loc- ti.Ldev $\leq$ tr.Loc $\leq$ ti.Loc + ti.Ldev
$\wedge$
ti.Speed- ti.Sdev $\leq$ tr.Speed $\leq$ ti.Speed +Sdev

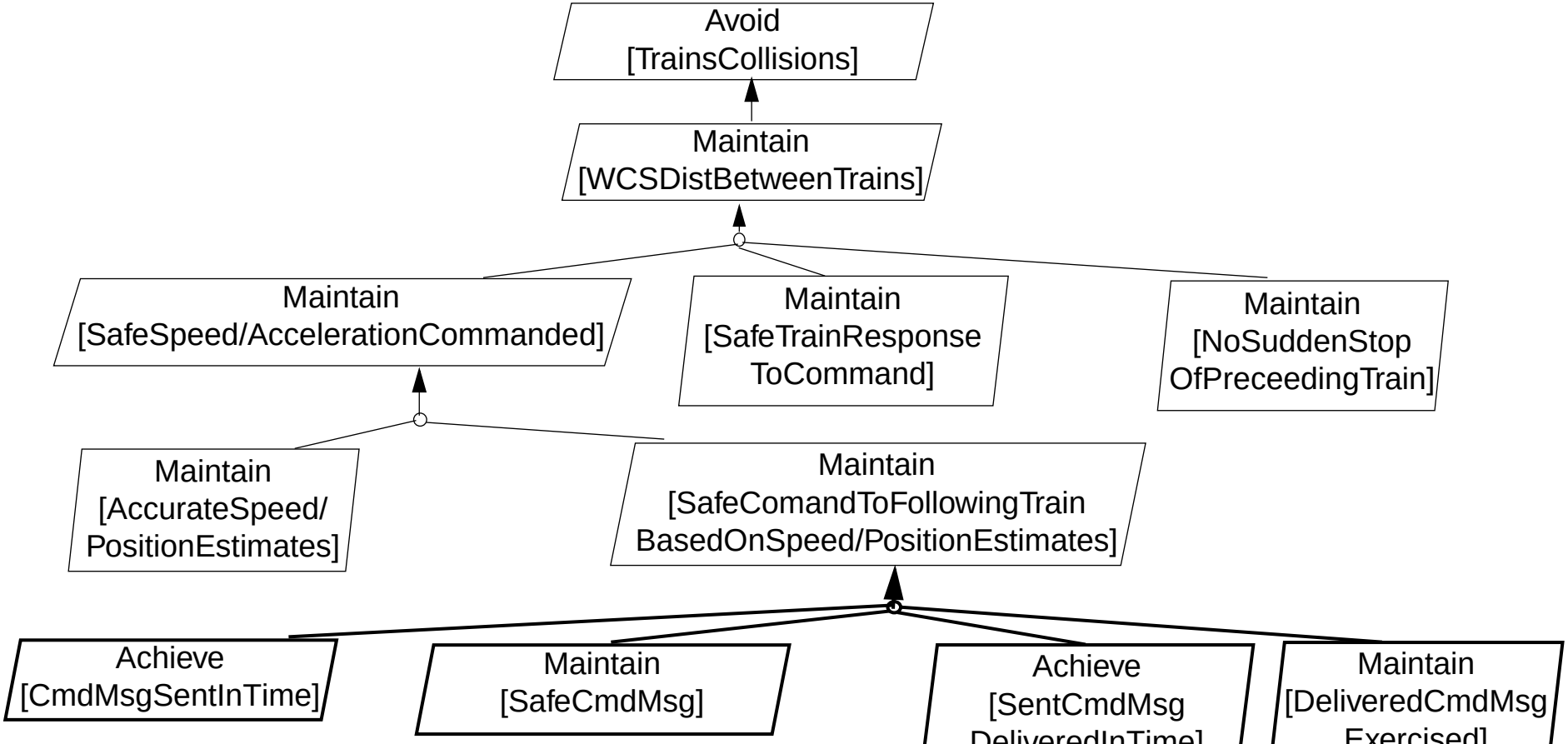
FollowingInfo(ti1, ti2)
$\wedge$ Tracking(ti1, tr1) $\wedge$ Tracking(ti2, tr2)
$\Rightarrow$
$tr1.acc_{CM}' \leq F$ (**ti1.Loc+ ti1.LDev**, **ti2.Loc - ti2.Ldev**,
ti1.speed + ti1.Sdev, **ti2.Speed - ti2.Sdev**)
$\wedge$
$tr1.Speed_{CM}' >$ ti1.Speed+ ti1.Sdev

Eliciting New Goals: HOW Questions (3)



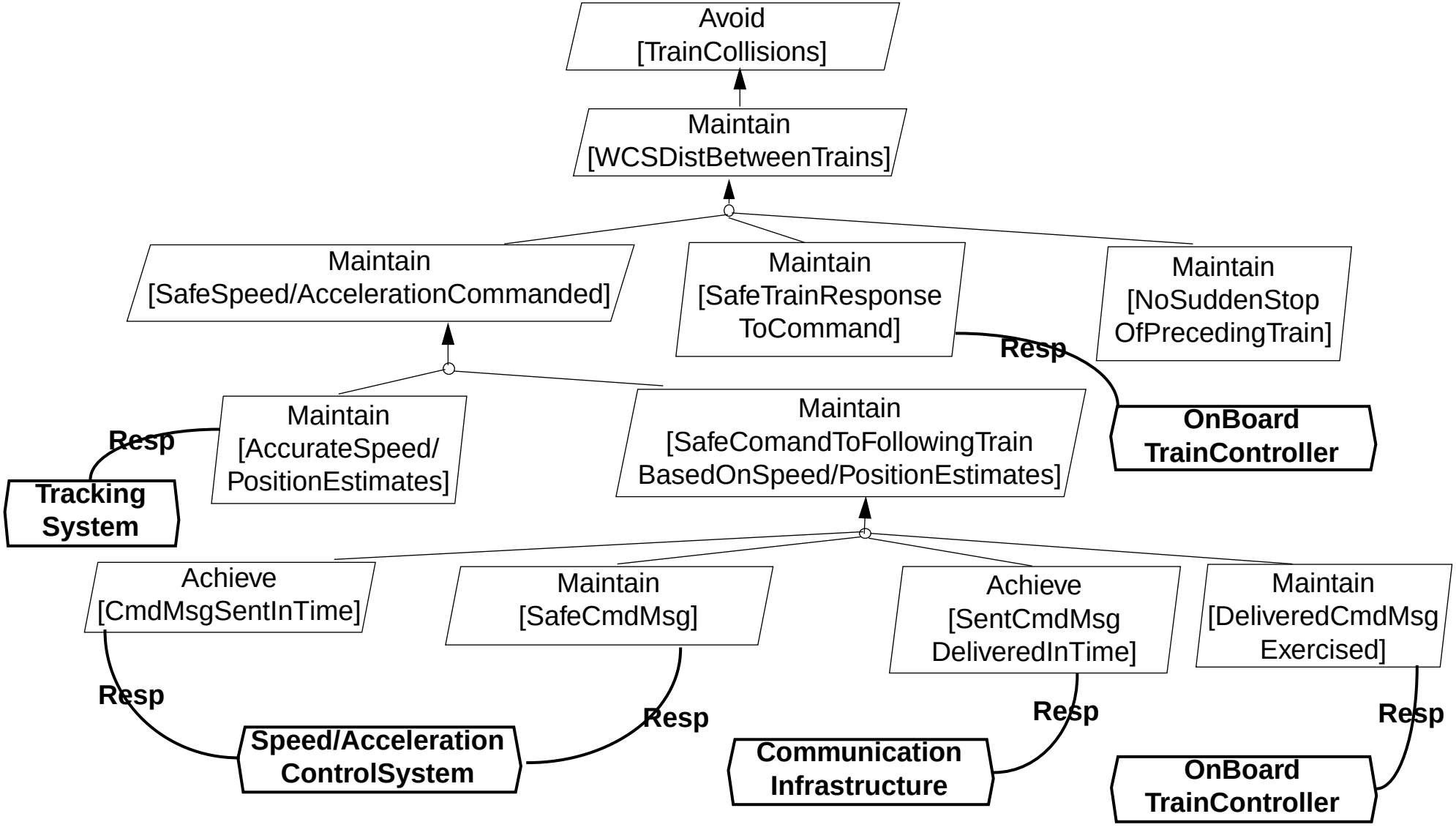
$\forall$ **cm: CommandMessage**, ti1, ti2: TrainInfo
cm.Sent $\wedge$ cm.TrainID = ti1.TrainID
$\wedge$ FollowingInfo(ti1, ti2)
$\Rightarrow$
cm.Acc $\leq$ F (ti1.Loc+ ti1.LDev, ti2.Loc - ti2.Ldev,
ti1.Speed + ti.Sdev, ti2.Speed - ti2.Sdev)
$\wedge$ **cm.Speed** $>$ ti1.Speed+ ti1.Sdev

Eliciting New Goals: HOW Questions (3)



$\forall$ **cm: CommandMessage**, ti1, ti2: TrainInfo
cm.Sent $\wedge$ cm.TrainID = ti1.TrainID
$\wedge$ FollowingInfo(ti1, ti2)
$\Rightarrow$
cm.Acc $\leq$ F (ti1.Loc+ ti1.LDev, ti2.Loc - ti2.Ldev,
ti1.Speed + ti.Sdev, ti2.Speed - ti2.Sdev)
$\wedge$ **cm.Speed** > ti1.Speed+ ti1.Sdev

Identifying Potential Responsibility Assignments



Corresponding Agent Interface Model

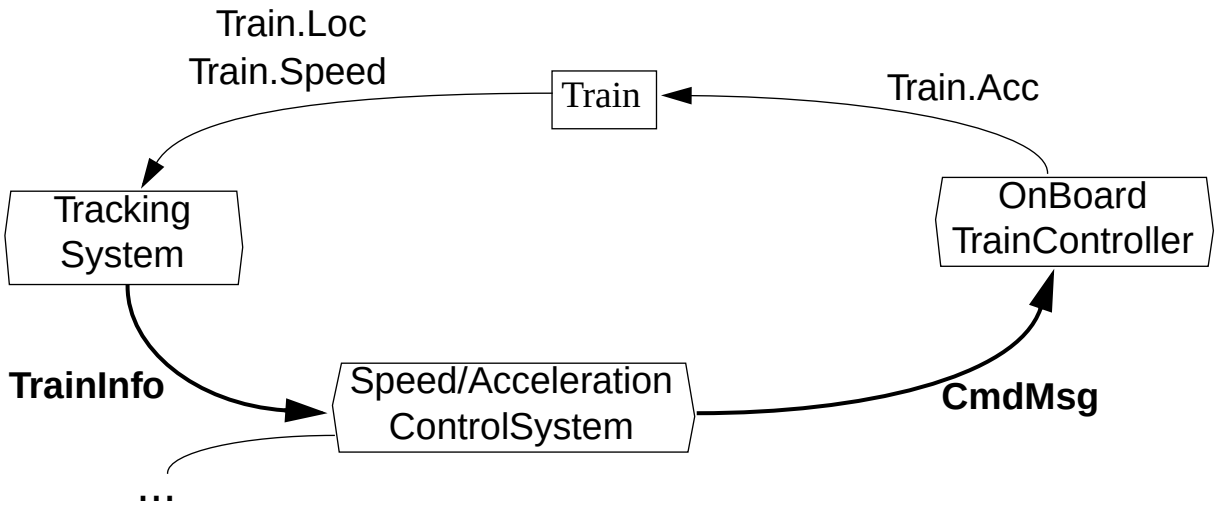
Goal Maintain[SafeCmdMsg]

FormalDef ∀ cm: *CommandMessage*, ti1, ti2: *TrainInfo*

cm.Sent ∧ cm.TrainID = ti1.TrainID
∧ FollowingInfo(ti1, ti2)
⇒
cm.Acc ≤ F (ti1.Loc+ ti1.LDev, ti2.Loc - ti2.Ldev,
ti1.Speed + ti.Sdev, ti2.Speed - ti2.Sdev)
∧ cm.Speed > ti1.Speed+ ti1.Sdev

Resp

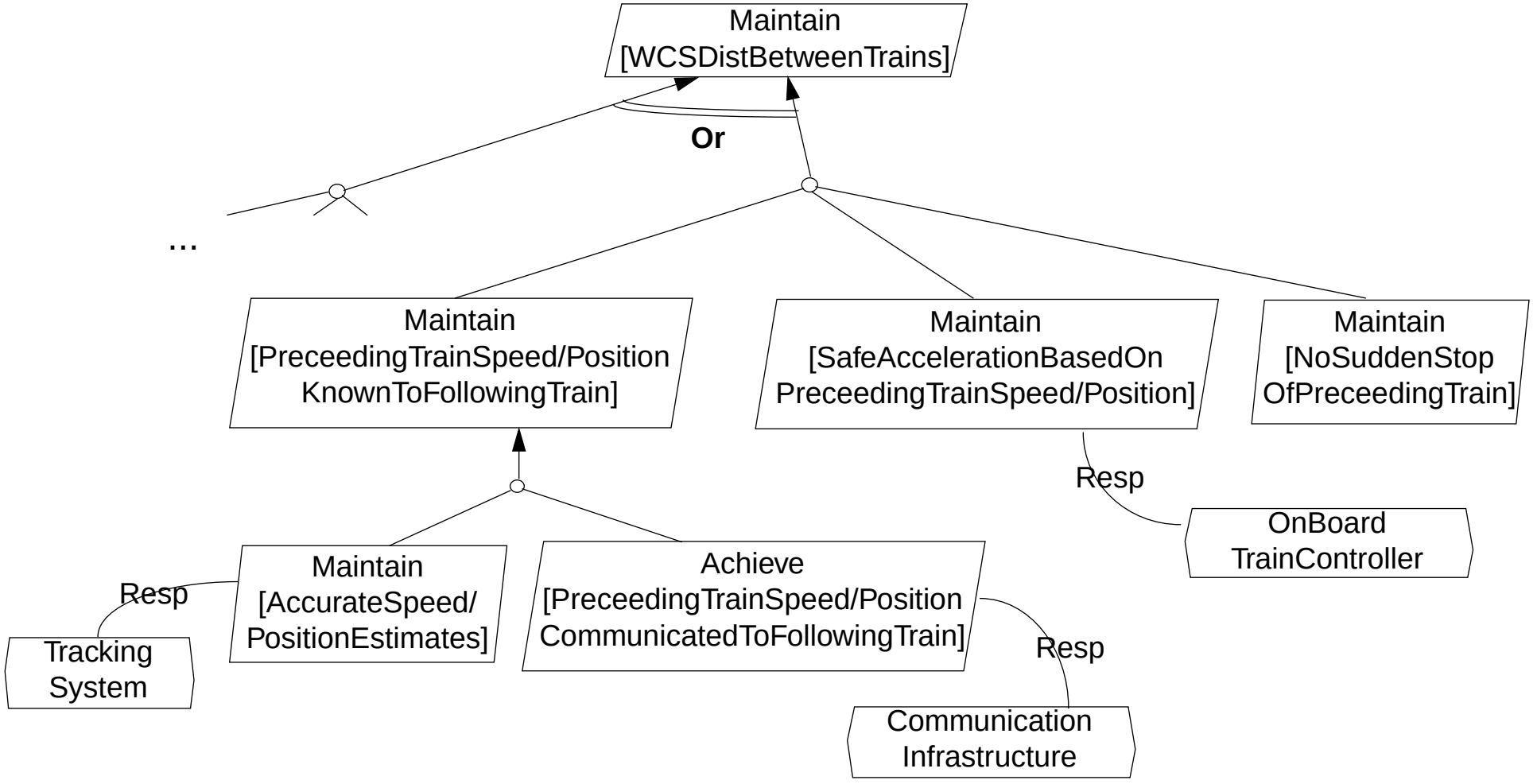
Speed/Acceleration ControlSystem



Controlling Agent → Object.Attribute → Monitoring Agent

Alternative Goal Refinements and Responsibility Assignments

==> different design : fully distributed system



Identifying Operations and DomPre/Post

= Identify state transitions relevant to goals

Goal Maintain[SafeCmdMsg]
  FormalDef ∀ cm: CommandMessage, ti1, ti2: TrainInfo
    cm.Sent ∧cm.TrainID = ti1.TrainID
    ∧ FollowingInfo(ti1, ti2)
    ⇒
    cm.Acc ≤ F (ti1.Loc+ ti1.LDev, ti2.Loc - ti2.Ldev,
                ti1.Speed + ti.Sdev, ti2.Speed - ti2.Sdev)
    ∧ cm.Speed > ti1.Speed+ ti1.Sdev

==>

Operation SendCommandMessage
  Input Train {**arg** tr}
  Output ComandMessage {**res** cm}
  DomPre ¬ cm.Sent
  DomPost cm.Sent ∧ cm.TrainID = tr.ID

Operationalizing Goals

Goal Maintain[SafeCmdMsg]
  FormalDef ∀ cm: CommandMessage, ti1, ti2: TrainInfo
    cm.Sent ∧cm.TrainID = ti1.TrainID
    ∧ FollowingInfo(ti1, ti2)
    ⇒
    cm.Acc ≤ F (ti1.Loc+ ti1.LDev, ti2.Loc - ti2.Ldev,
                ti1.Speed + ti.Sdev, ti2.Speed - ti2.Sdev)
    ∧ cm.Speed > ti1.Speed+ ti1.Sdev

==>

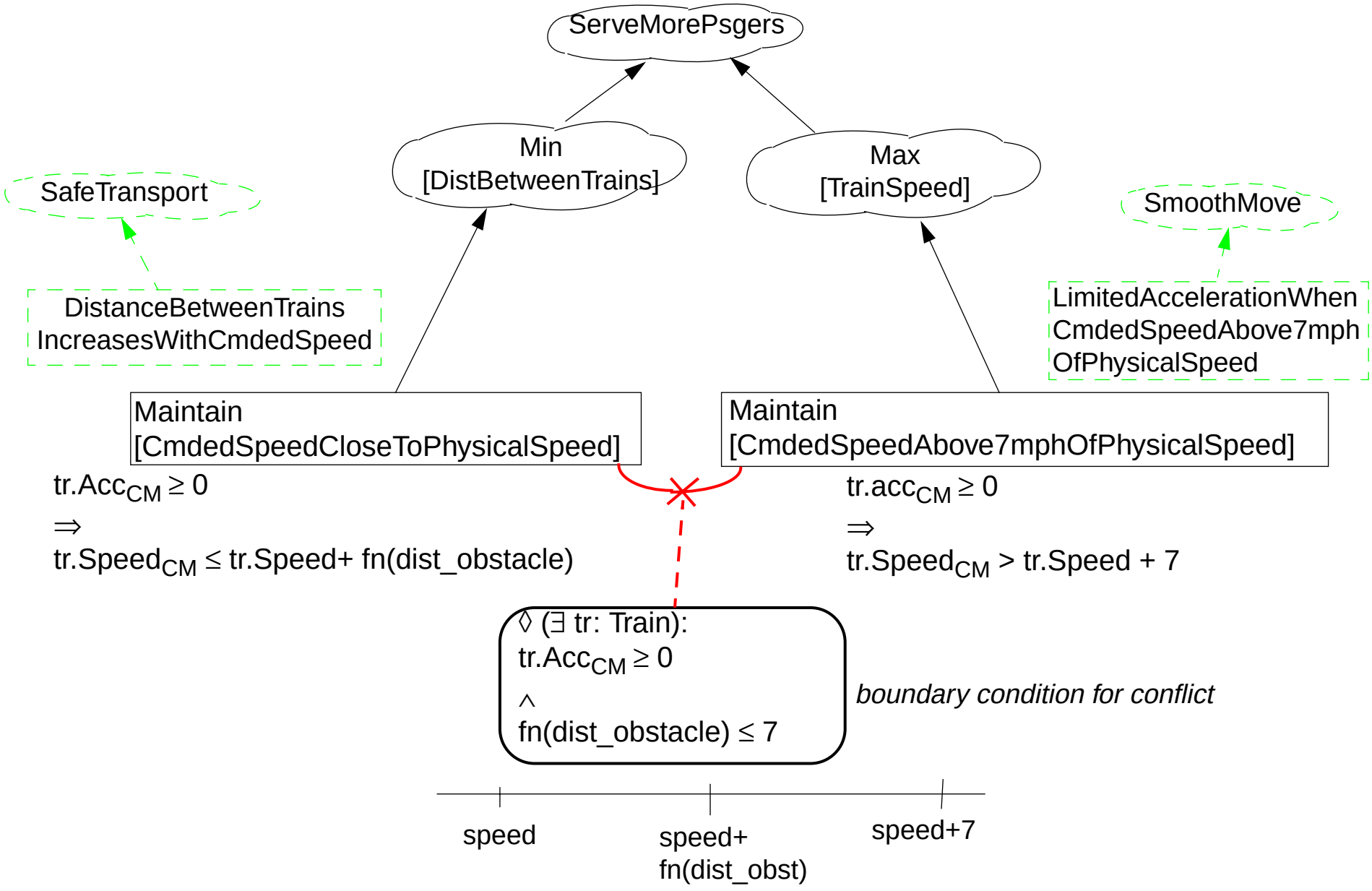
Operation SendCommandMessage
  Input Train {**arg** tr}
      TrainInfo
  Output ComandMsg {**res** cm}
  DomPre ¬ cm.Sent
  DomPost cm.Sent ∧ cm.TrainID = tr.ID

  ReqPostFor [SafeCmdMsg]
    Tracking(ti1, tr) ∧ Following(ti1, ti2)
    →
    cm.Acc ≤ F (ti1.Loc+ ti1.LDev, ti2.Loc - ti2.Ldev, ti1.Speed + ti.Sdev, ti2.Speed - ti2.Sdev)
    ∧ cm.Speed > ti1.Speed+ ti1.Sdev

  ReqTrigFor [CmdMsgSentInTime]
    $\blacksquare_{\leq 1/2 \text{ sec}}$ ¬ ∃ cm2: CommandMessage: cm2.Sent ∧ cm2.TrainID = tr.ID

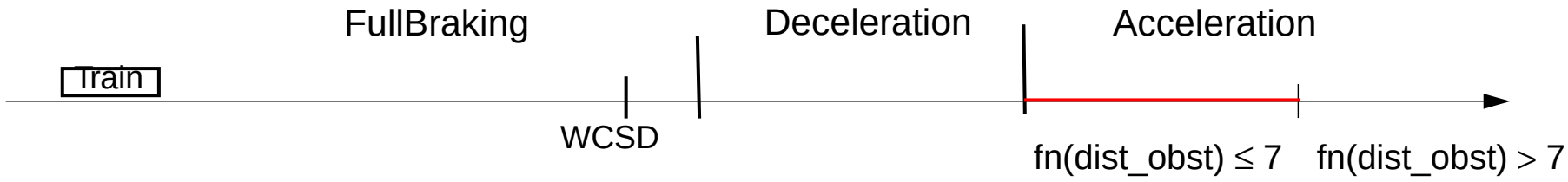
This is not the end of the story ...

Conflict Analysis: An Example



Conflict Resolution

Note: fn(dist_obst) increases with dist(obst)



==> **Conflict Resolution**:

Weaken *Maintain [CmdedSpeedAbove7mphOfPhysicalSpeed]*

$tr.Acc_{CM} \geq 0$

$\Rightarrow$

$tr.Speed_{CM} > tr.Speed + 7 \vee$ **fn(dist_obst) ≤ 7**

Rationale: if boundary condition is true, priority is to avoid going into deceleration mode

Obstacle Analysis

- Obstacle = high-level exception

Obstacle *O* obstructs goal *G* iff

1. {*O, Dom* } |== ¬ *G* (Obstruction)
2. *Dom* |=/= ¬ *O* (Domain Consistency)

- Handle obstacles at RE time

  ==> identification of new requirements

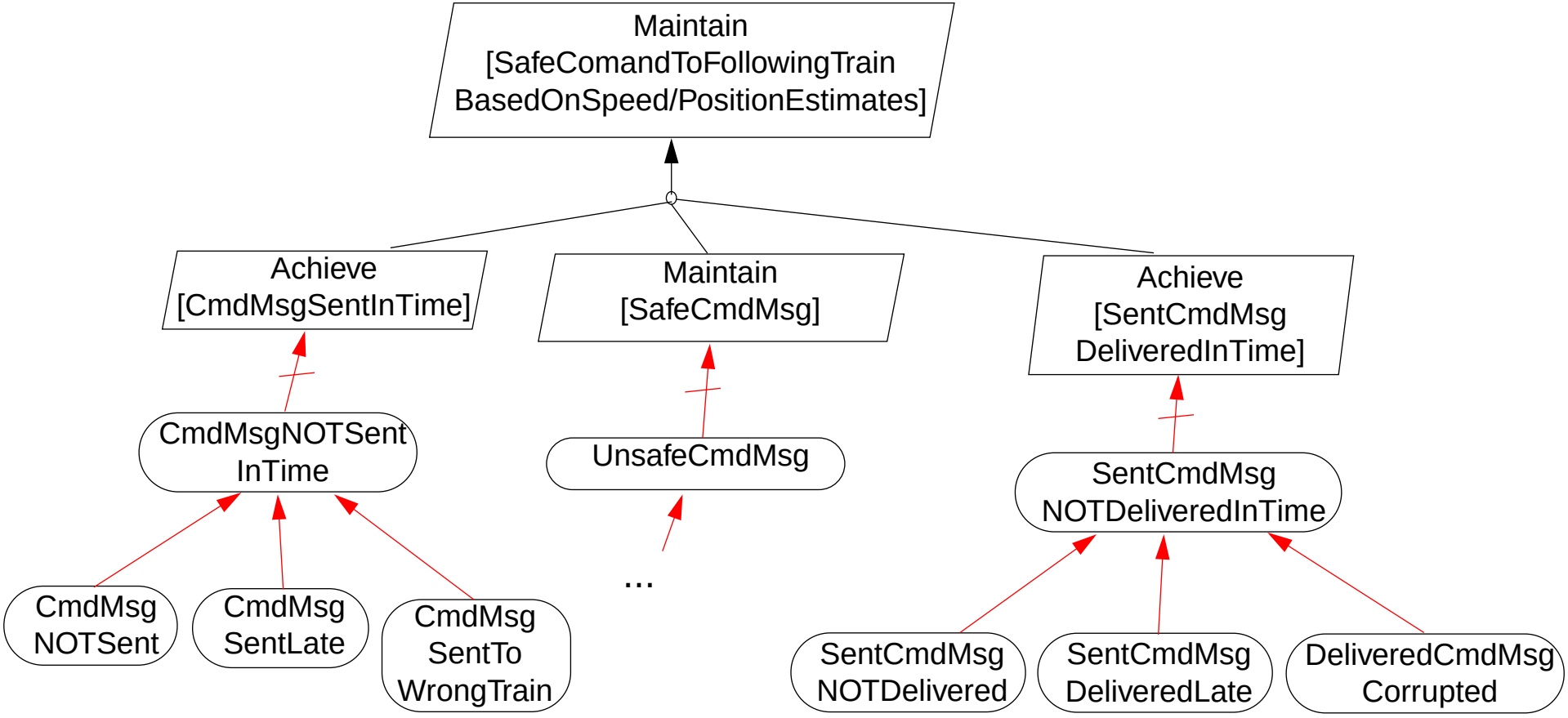
  ==> more robust system

Handling obstacles during goal-oriented requirements elaboration

1. Identify obstacles
   -> formal techniques for generating obstacles from goal formulations
   -> heuristics as ligthweight rules of thumb
2. Generate alternative obstacle resolutions
   -> resolution operators ==> new goals/requirements

3. Alternative evaluation and selection

Obstacle Identification



- Goal-anchored form of Fault-Tree construction
- Formal techniques to generate obstacles from goal formulations

A. van Lamsweerde and E. Letier, Handling Obstacles in Goal-Oriented Requirement Engineering, to appear in *IEEE-TSE, Special Issue on Exception Handling*, 2000.

Generating Alternative Obstacle Resolutions

Goal Substitution
= choose alternative goal
CmdMsgSentLate **Obstructs** *Achieve[CmdMsgSentInTime]*
UnderResponsibilityOf *Speed/AccelerationControlSystem*
==> alternative design : acceleration calculated by on-board train controller

Agent Substitution
= change responsibility assignment for obstructed goal
UnsafeAccelerationInCmdMsg **Obstructs** *SafeAccelerationInCmdMsg*
UnderResponsibilityOf *Speed/AccelerationControlSystem*
==> UnderResponsibilityOf *VitalStationComputer*

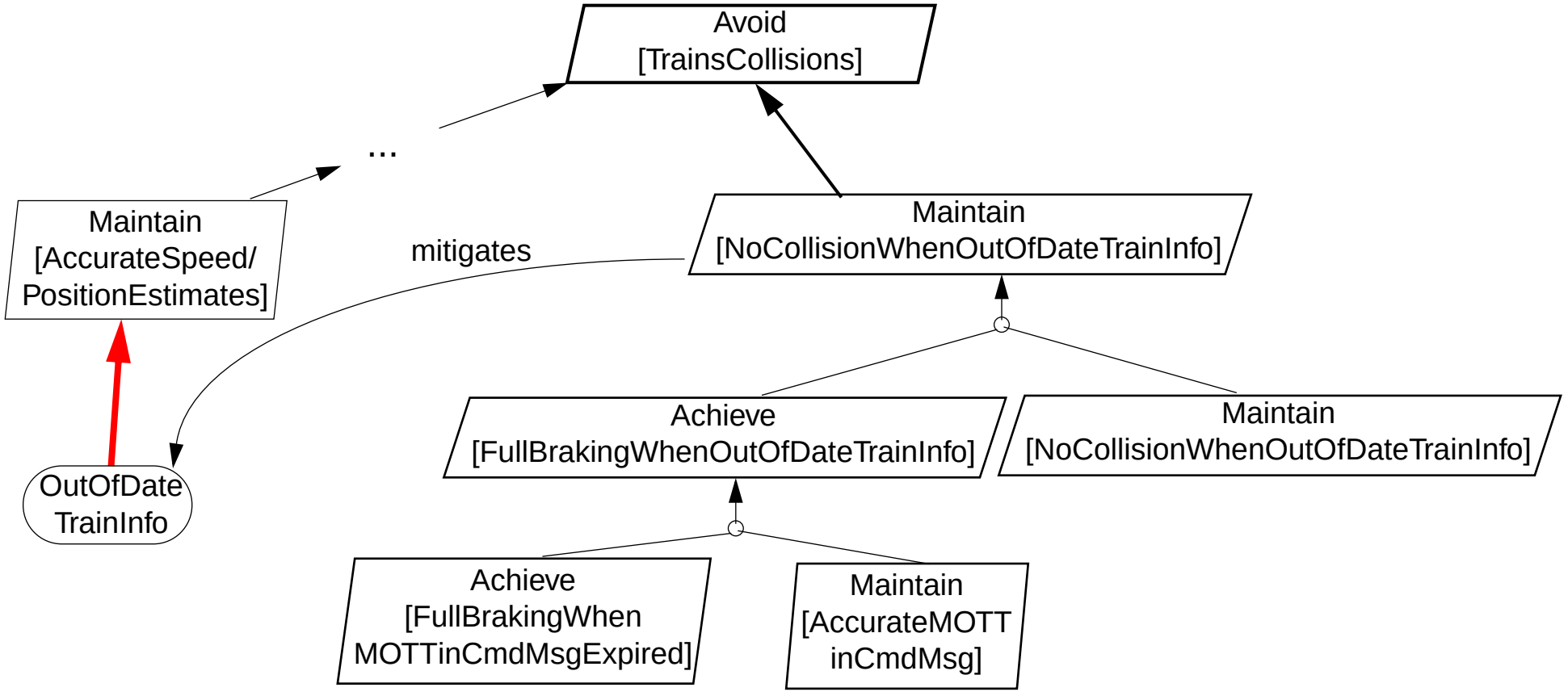
Obstacle Prevention
= add new goal: ¬ *O*
ImpossibleChangeInTrainSpeed/PositionEstimates **Obstructs** *AccurateSpeed/PositionEstimates*
==> New Goal: *Avoid[ImpossibleTrainInfoChange]*
(to be assigned as responsibility of TrackingSystem OR StationComputer)

Goal Deidealization
= weakening goal to make obstruction disappear

Obstacle Mitigation

= add new goal that tolerates obstacle but mitigates its consequences

e.g.



==> derivation of new requirements

Message Origination Time Tag attribute

Conclusions

- Systematic derivation of requirements from goals

  (required pre/post/trigger conditions, monitored/controlled objects)

- Goal formalization

  ==> refinement correctness proof

  ==> conflict identification/resolution

  ==> obstacle generation/resolution

- Goal-oriented explanation of requirements
- Goal structure provides structure for requirements document
- Separation of concerns: requirements vs. assumptions vs. domain properties
- Exploration of alternative system proposals