Goal-oriented requirements with KAOS

Riccardo Scandariato

Institute of Software Security, TUHH, Germany

ric***do . scanda***to @ tuhh.de

Learning objectives

- What is goal-oriented requirements engineering?
- How to formalize security goals via security specification patterns?
- What are anti-goals and threat analysis at requirements level

Reading material
Axel van Lamsweerde, Elaborating Security Requirements by Construction of Intentional Anti-Models, International Conference on Software Engineering, 2004

GOAL-ORIENTED REQUIREMENTS

What are goals?

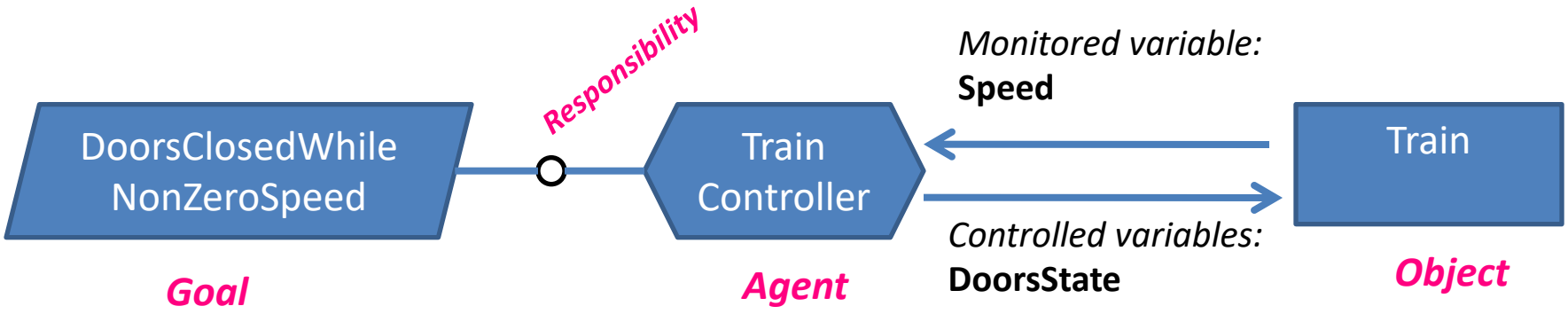
- A *goal* is a prescriptive statement of intent that the system should satisfy through the cooperation of its agents
- An *agent* is an active system component playing a specific role in the goal satisfaction
  - Human such as operator and users
  - Devices such as sensors, actuators, communication media, measurement instruments
  - Existing SW components such as legacy, off-the-shelf or foreign
  - New SW components forming the software to be

What are agents?

- A system component playing a role in goal satisfaction
  - Role rather than individual
- Active object
  - **Responsibilities** (goals)
  - **Capabilities** (monitor/control)
  - **Behavior** (performs operations)
- To play such role, agents need to restrict their behavior by adequate control of system items

Capabilities and responsibilities

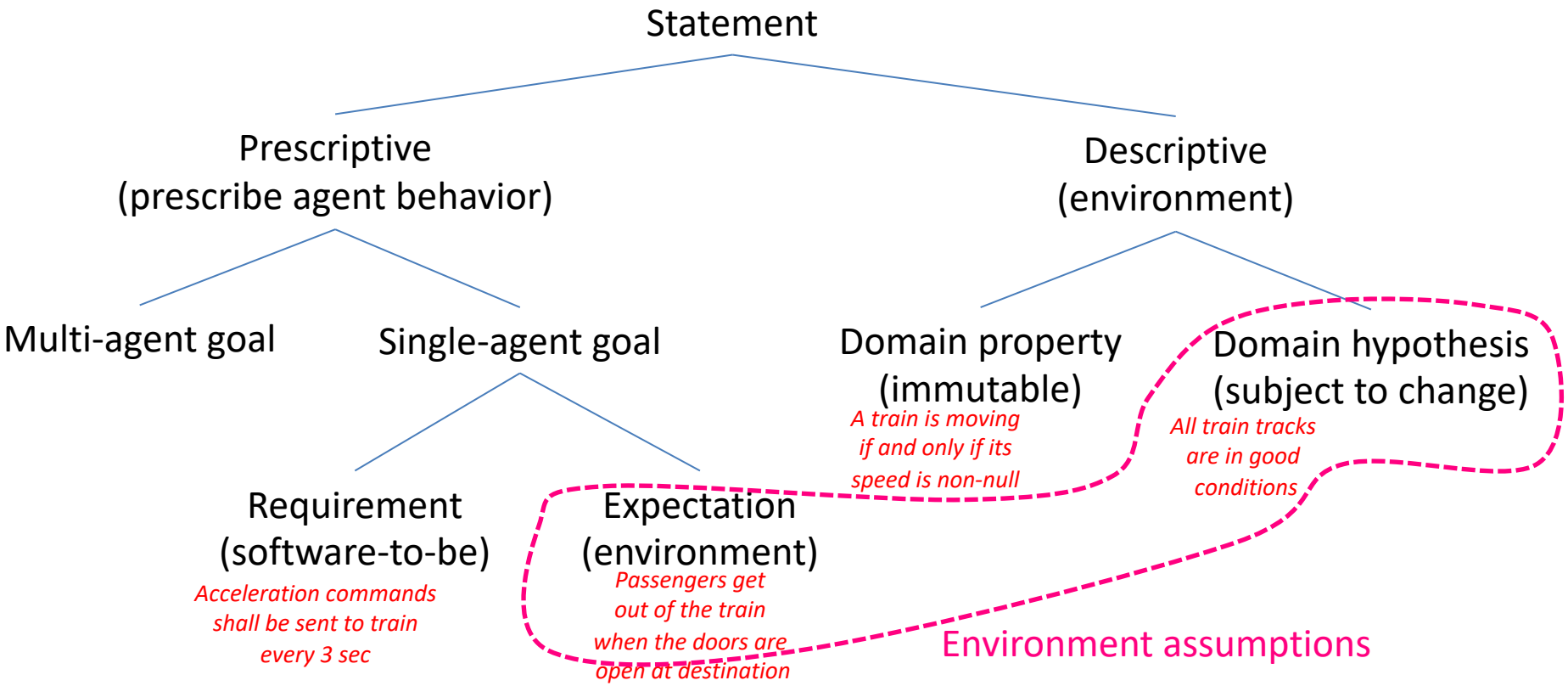
- Capabilities are the monitoring links and control links to objects
  - Attributes (get or set values)
  - Associations (check or create/delete)
- Responsible for a goal if its instances are the only ones required to restrict their behavior, through adequate setting of their behavior, so to satisfy the goal



What are goals? And what they are not!

- Meetings shall be scheduled so as to maximize the attendance of invited participants
  - Participants, initiator (in the environment)
  - Scheduler (in the software-to-be)
- Make user happy
  - Out of reach
- To initiate the meeting, the initiator needs to prompt the scheduler, authenticate, fill in a form and then confirm the request
  - Not prescriptive statement of intent (declarative vs operational)

Statements

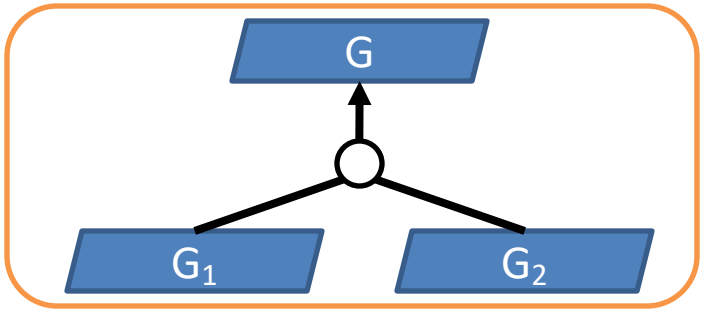

Statement
Prescriptive
(prescribe agent behavior)
Descriptive
(environment)
Multi-agent goal
Single-agent goal
Domain property
(immutable)
Domain hypothesis
(subject to change)
A train is moving
if and only if its
speed is non-null
All train tracks
are in good
conditions
Requirement
(software-to-be)
Acceleration commands
shall be sent to train
every 3 sec
Expectation
(environment)
Passengers get
out of the train
when the doors are
open at destination
Environment assumptions

Goal granularity

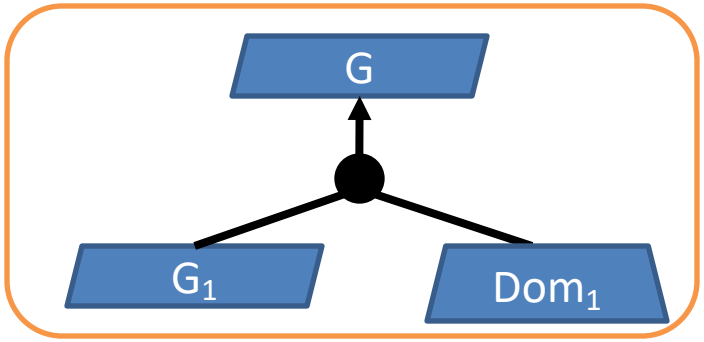
- High-level: strategic objectives
  - Larger cooperation needed
  - Ex: Meetings shall be scheduled so as to maximize the attendance of invited participants
- Low-level: technical objectives
  - Fewer agents
  - Ex: Reminders for upcoming meetings shall be issued

Goal refinement in KAOS

- AND-refinement
  - ‘Necessary’ to achieve G

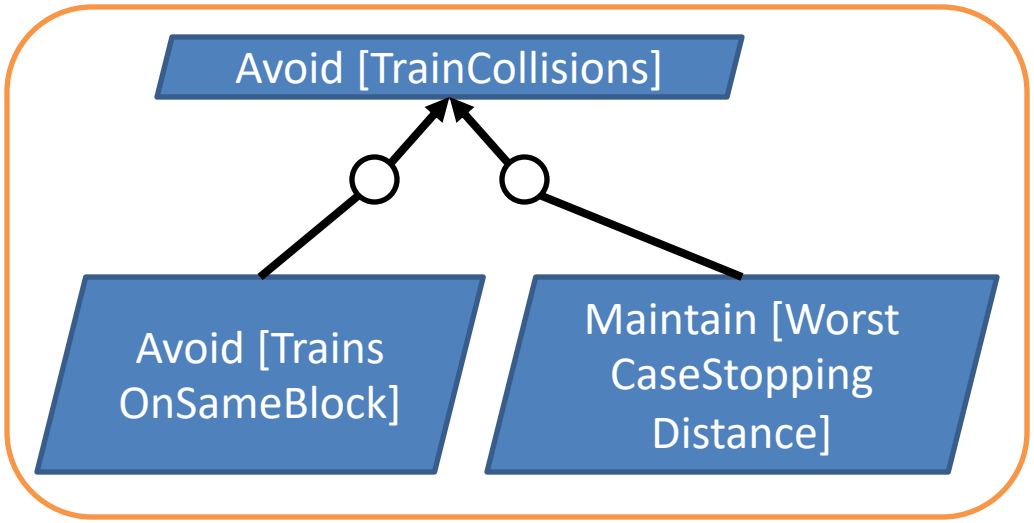


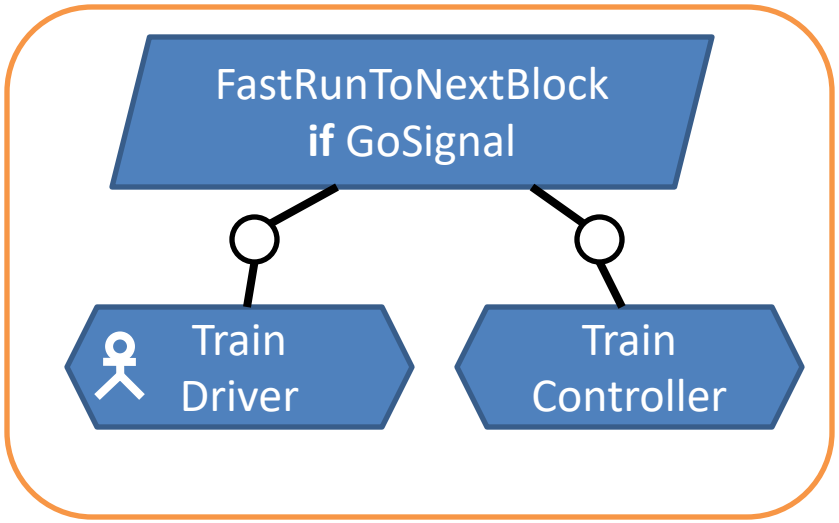
  - Complete refinement
    - ‘Sufficient’ to achieve G
    - Often uses domain properties and hypotheses



Alternatives in KAOS

- OR-refinement
  - Goal refinement
  - Goal assignment
- Generally result in different system designs

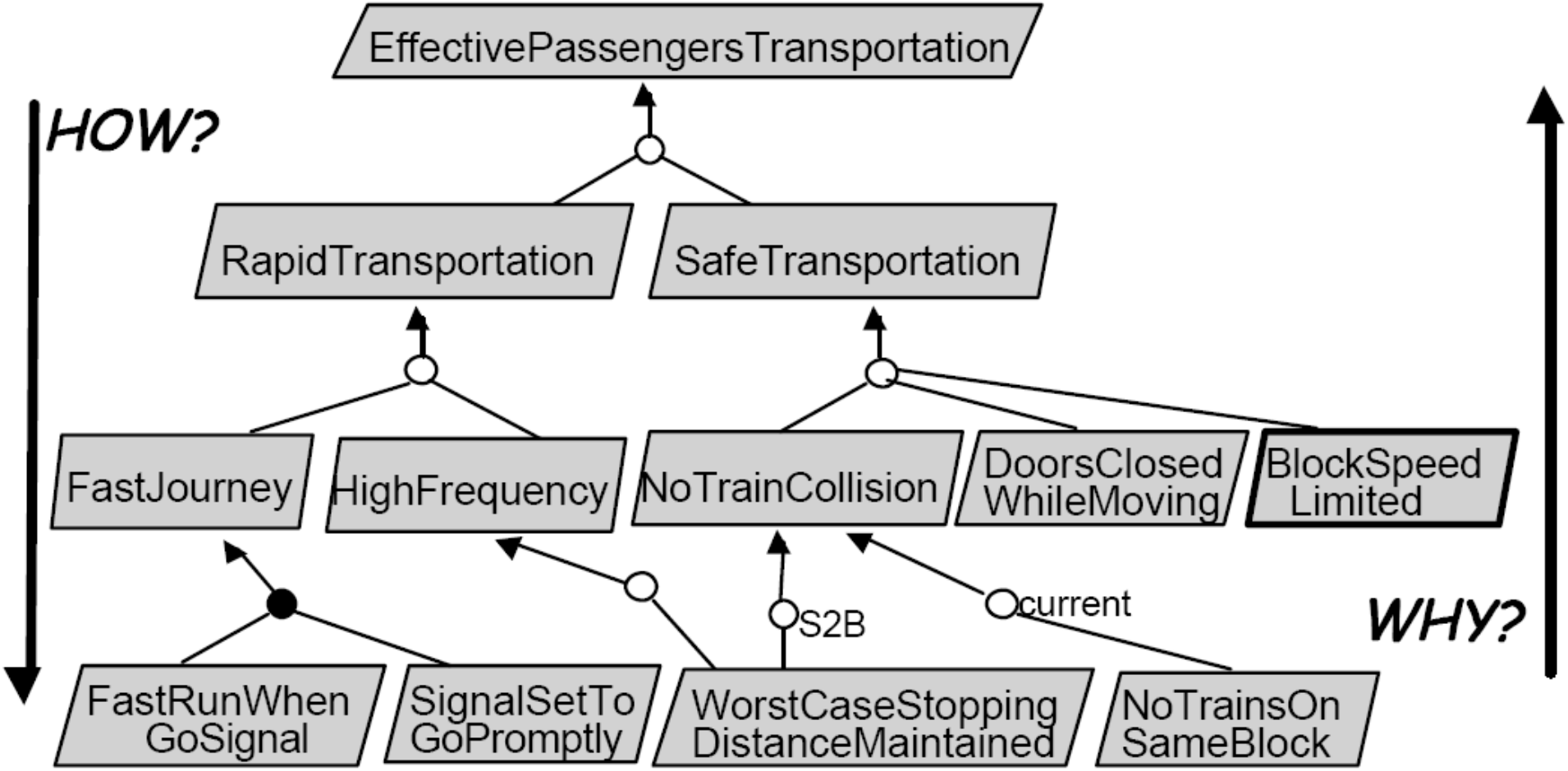




Building goal models

- Early discovery
  - Analysis of current system
  - Search for intentional and prescriptive keywords in documents
- Later discovery
  - By abstraction (“Why?”, bottom-up)
    - Until boundary of system capabilities is reached (system scope!)
  - By refinement (“How?”, top-down)
    - Until assignable to single agent as requirement or expectation

Building the goal model



Behavioral goals

Achieve [TargetCondition]

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

◊ "in some future state"

if a train is at some platform **then within 5 minutes** the train is at the next platform

Cease [TargetCondition]

[if CurrentCondition **then] sooner-or-later *not*** TargetCondition

Behavioral goals

Maintain [GoodCondition]

[if CurrentCondition **then] always** GoodCondition

□ "in every future state"

Maintain [DoorsClosedWhileMoving]

always (**if** a train is moving **then** its doors are closed)

Avoid [BadCondition]

[if CurrentCondition **then] always *not*** BadCondition

Avoid [TrainsOnSameBlock]

always not (more than one train at one block)

Only for your info

Some Linear Temporal Logic

Future

o P	P shall hold in the next state
◇ P	P shall hold in some future state (sooner or later)
□ P	P shall hold in every future state (always)
P ***U*** N	P shall hold until N becomes true (always until) (N will eventually become true)
P ***W*** N	P shall hold unless N becomes true (always unless) (N might not become true)

Past

● P	
◆P	
■ P	
P ***S*** N	
P ***B*** N	

Other

$\Box_{\leq d}$ P	P shall hold in every future state up to deadline d
$\Diamond_{\leq d}$ P	P shall hold within deadline d
P $\Rightarrow$ Q	□ (P $\rightarrow$ Q) (entailment)
@P	● ($\neg$ P) $\wedge$ P ('P just became true in the current state')

Soft goals

- Improve [TargetCondition]
- Increase/Reduce [TargetQuantity]
- Maximize/Minimize [ObjectiveFunction]

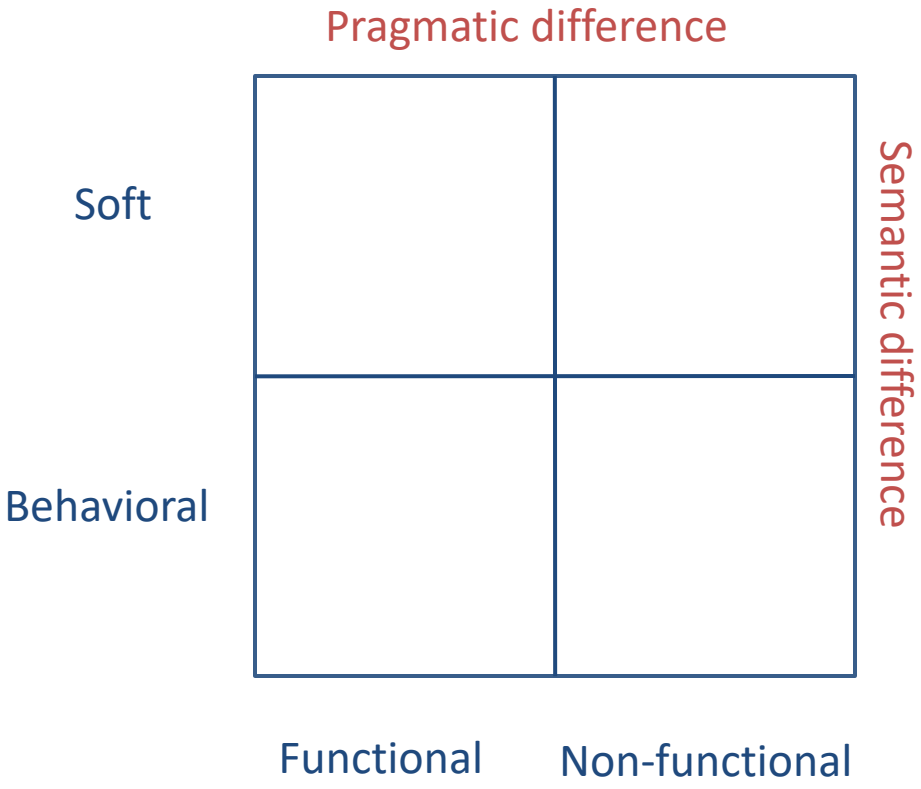
Goal types and categories

TYPE

- Behavioral
  - Clear cut sense (in isolation)
- Soft goals
  - Preferences among alternatives

CATEGORY

- Functional
  - Intent underpinning a system service
- Non functional
  - Quality or constraint on service provisioning or development



SECURITY GOALS

Application-level security analysis in KAOS

- A threat is the possibility of an asset in the system going unprotected against unintended behavior
  - **Obstacle analysis**: unintentional threats
  - **Threat analysis**: intentional threats

KAOS terminology

- Unintentional obstruction: possibility of inadvertent violation of a security goal
- Intentional obstruction: possibility of proactive violation of a security goal by exploitation of unprotected data and system knowledge acquired through malicious behaviors, calculations, deductive inferences, etc.

Identify security goals

Two complementary methods

a) Security specification patterns

b) Threat analysis and anti-goals (i.e., converse of asset-related achieve goals)

Security specification patterns (1/2)

Confidentiality
Goal *Avoid* [SensitiveInfoKnownByUnauthorizedAgent]
FormalSpec ∀ ag: Agent, ob: Object
¬ Authorized (ag, ob.info) ⇒ ¬ $KnowsV_{ag}$ (ob.info)

Agent knowledge must be modeled. LTL extended with epistemic operator

$Knows_{ag}$ (P) ≡ **$Belief_{ag}$** (P) ∧ P (knows property)

↑
"P is in ag's local memory"

$KnowsV_{ag}$ (x) ≡ ∃ v: $Knows_{ag}$ (x=v) (knows value)

↑
state variable

Security specification patterns (2/2)

Confidentiality
Goal *Avoid* [SensitiveInfoKnownByUnauthorizedAgent]
FormalSpec ∀ ag: Agent, ob: Object
¬ Authorized (ag, ob.info) ⇒ ¬ $KnowsV_{ag}$ (ob.info)

Authorized is generic predicate and needs to be instantiated through a domain-specific definition. E.g.

∀ ag: Agent, acc: Account
Authorized (ag, acc) ≡ **Owner** (ag, acc) ∨ **Proxy** (ag, acc) ∨ **Manager** (ag, acc)

Spec patterns for other security properties

Privacy

Goal *Maintain*[PrivateInfoKnownOnlyIfConsentedByOwner]

FormalSpec ∀ ag, ag': Agent, ob: Object

KnowsV_{ag} (ob.info) ∧ OwnedBy (ob.info, ag') ∧ ag ≠ ag'

⇒ Consent (ag, ob.info, ag')

Integrity

Goal *Maintain*[ObjectInfoChangeOnlyIfCorrectAndAuthorized]

FormalSpec ∀ ag: Agent, ob: Object, v: Value

ob.info = v ∧ **o** (ob.info ≠ v) ∧ UnderControl (ob.info, ag)

⇒ Authorized (ag, ob.info) ∧ o Integrity (ob.info)

"in the next state"

Availability

Goal *Achieve*[ObjectInfoUsableWhenNeededAndAuthorized]

FormalSpec ∀ ag: Agent, ob: Object

Needs (ag, ob.info) ∧ Authorized (ag, ob.info)

⇒ $\Diamond_{\leq d}$ Using (ag, ob.info)

"soner or later before the deadline d"

Instantiate pattern

Confidentiality
Goal *Avoid* [SensitiveInfoKnownByUnauthorizedAgent]
FormalSpec ∀ ag: Agent, ob: Object
¬ Authorized (ag, ob.info) ⇒ ¬ $KnowsV_{ag}$ (ob.info)

Agent …
Object …
Authorized …

Examples

Goal *Avoid* [AccountNumber&PinDisclosedToUnauthorized]
FormalSpec ∀ p: Person, acc: Account
¬ (Owner (p, acc) ∨ Proxy (p, acc) ∨ Manager (p, acc))
⇒ ¬ ($KnowsV_p$ (acc.Acc#) ∧ $KnowsV_p$ (acc.PIN))

Goal *Avoid* [ePurseBalanceDisclosedtoUnauthorized]
FormalSpec ∀ p: Person, ep: ePurse
¬ Owner (p, ep) ⇒ ¬ $KnowsV_p$ (ep.Balance)

Instantiate pattern

a) Instantiating
   meta-classes (such as Object, Agent) and
   generic attributes (such as Info*)*
   to application-specific sensitive classes, attributes and associations in the object model

b) Specializing
   predicates (such as Authorized, UnderControl)
   through substitution by application-specific definitions

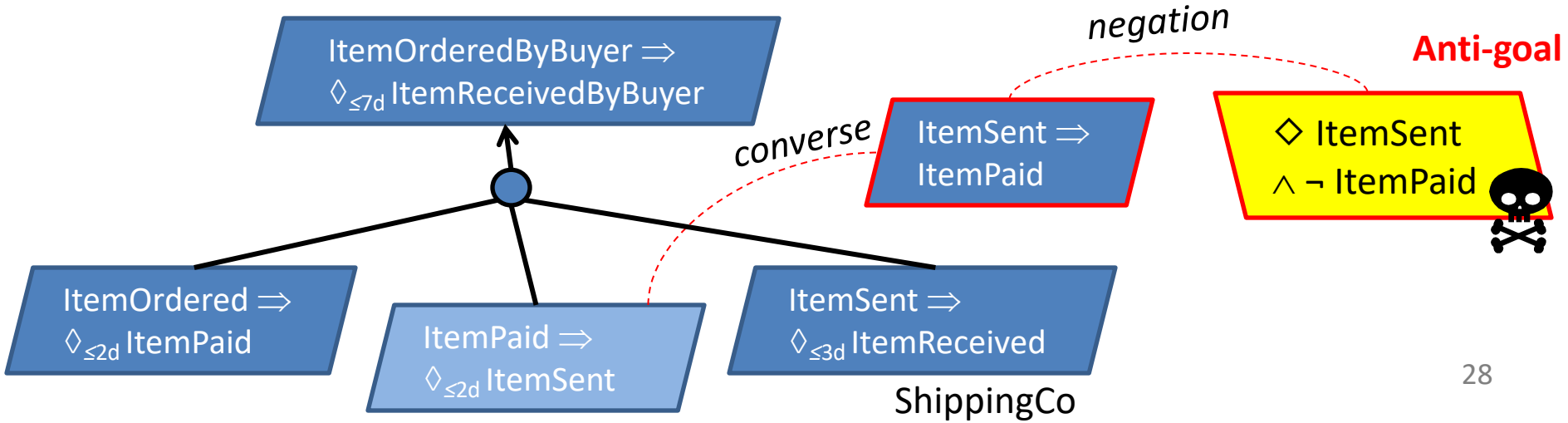
THREAT ANALYSIS

Starting point: anti-goals

- Check converse of asset-related **Achieve** goals

PreCondition $\Rightarrow \Diamond$ TargetCondition	(Achieve goal)
has converse	
TargetCondition $\Rightarrow$ PreCondition	(Maintain goal)

- Example



Threat analysis

1. Get initial anti-goals to be refined/abstracted
2. Identify attackers wishing them and their capabilities
3. Build threat graph
4. Derive new security goals as countermeasures

1. Initial anti-goals

$$AG \Leftrightarrow \neg\, SG$$

Anti goal

Security goal

1. Initial anti-goals

Security goal

Avoid[AccountNumber&PinDisclosedToUnauthorized]
$\forall$ 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)]

Negate goal

Achieve[AccountNumber&PinDisclosedToUnauthorized]
$\Diamond \exists$ p: Person, acc: Account
  $\neg$ Authorized(p, acc) $\wedge$ KnowsV$_p$(acc.Acc#) $\wedge$ KnowsV$_p$(acc.PIN)

Who would benefit from this?

2. Identify attackers and capabilities

- Who might benefit from satisfaction of anti-goal
  - Agent classes (insider/outside, hacker, thief, terrorist)

- What atomic conditions from the goal model the attacker can monitor/control

Anti-agent

- Attacker (malicious agent) has objectives
  - Anti-goals (threats as goals)
- Attacker has capabilities
  - Conditions he can monitor and control
- Attacker has system knowledge (anti-Dom)
  - Domain properties and goal model ("*most knowledgeable attacker*" assumption)
  - Software-to-be as part of *anti-environment*
  - Anti-domain properties include *requirements* and *vulnerabilities*

3. Build threat graph

- For each (initial anti-goal, attacker) build anti-goal refinement/abstraction graph

- Techniques
  - HOW questions to refine, WHY questions to find missing anti-goals
  - (Refinement patterns)
  - (Regression)

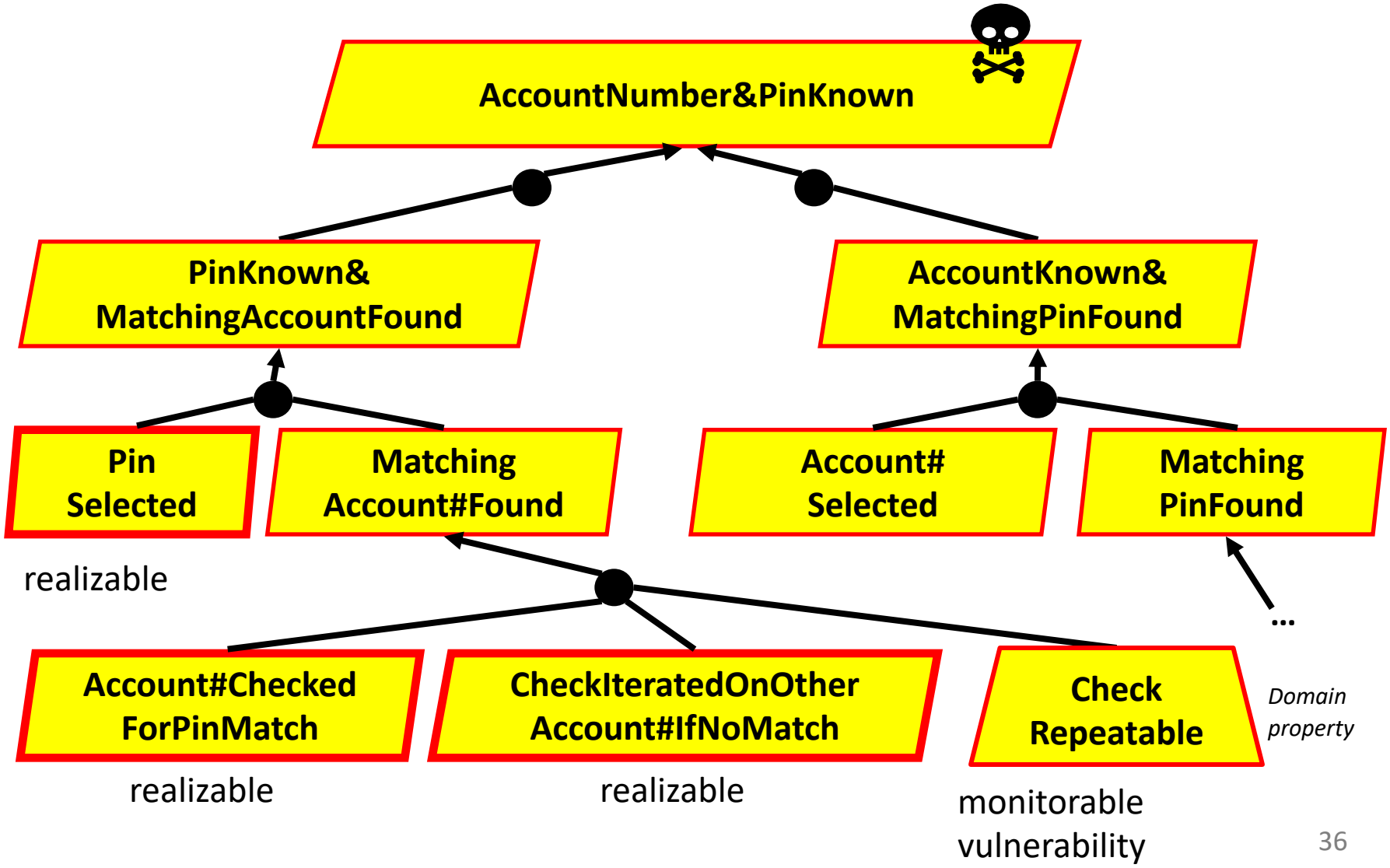
We do not cover these

Threat graph

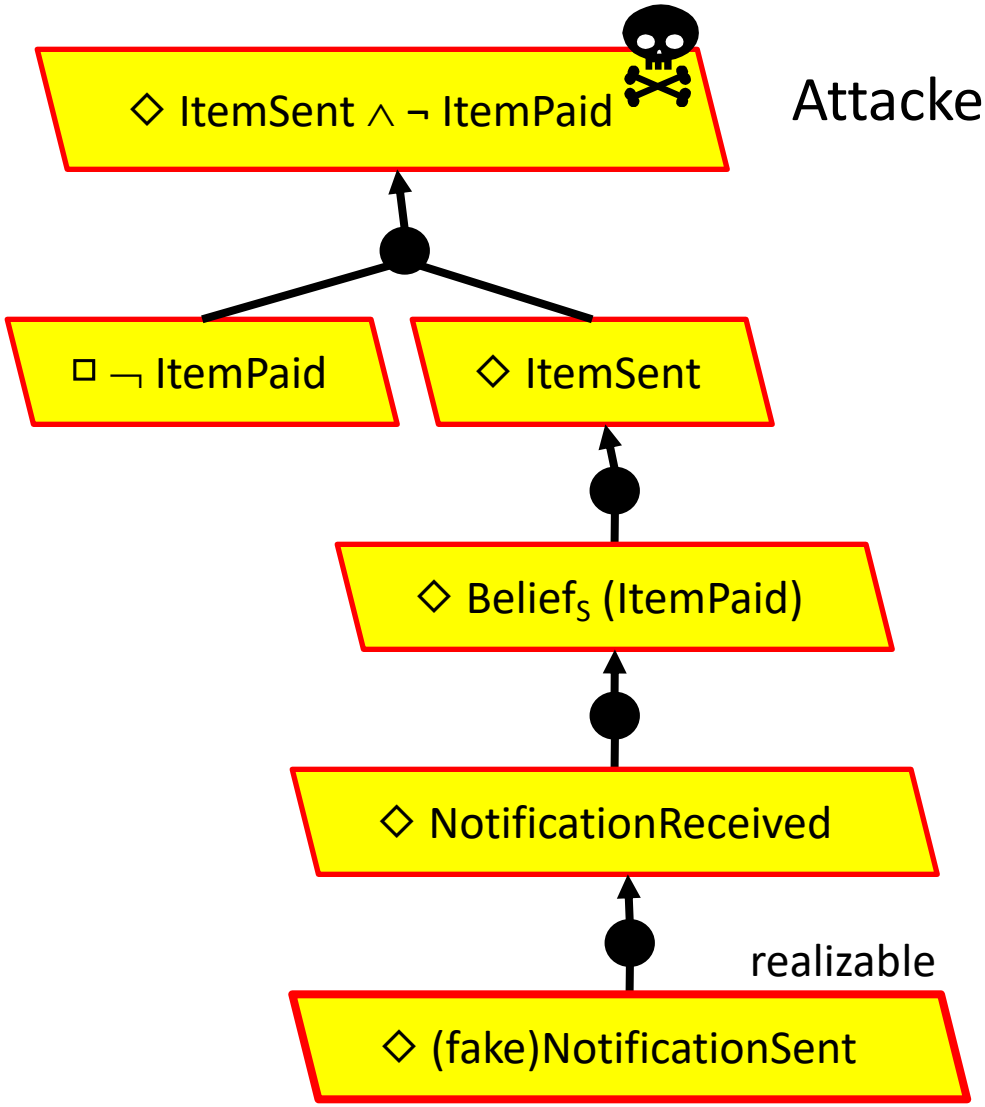
- Refinement of anti-goals ⇒ threat graph
- Terminal condition
  - Leaf anti-goals realizable by attacker agents (anti-requirements) with their capabilities, given their knowledge
  - Properties of the anti-domain (vulnerabilities of the attackee)

- Vulnerability
  - Anti-goal pre-condition to be satisfied by the attacked software-to-be and its environment

Threat graph



Threat graph (another example)



4. Countermeasures

- Avoid vulnerability (or anti-goal): add a new goal requiring the software vulnerability condition (or anti-goal) to be avoided

- New goals must be further refined
- A new cycle of threat analysis may be needed for these new goals !!!