STAF 2019

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

On Softening OCL invariants

Martin Gogolla[1] and Antonio Vallecillo[2]

[1]University of Bremen, [2]University of Málaga

ECMFA 2019

Eindhoven, Netherlands, July 2019

Uncertainty (in Science and Engineering)

Uncertainty: Quality or state that involves imperfect and/or unknown information

1. **Uncertainty** applies to predictions of future events, estimations, physical measurements, or unknown properties of a system, due to:
   - Underspecification
   - Lack of knowledge of the system actual behavior or underlying physics → **Epistemic U.**
   - Variability and lack of precision in measurements
   - Numerical approximations because values are too costly to measure
   - Associated properties not directly measurable/accessible (Estimations) → **Aleatory U.**

2. **Measurement Uncertainty**: A kind of aleatory uncertainty that refers to a set of possible states or outcomes where probabilities are assigned to each possible state or outcome.

3. **Occurrence Uncertainty**: a kind of epistemic uncertainty that refers to the degree of belief (***confidence***) that we have on the actual existence of an entity, i.e., the real entity that a model element represents.

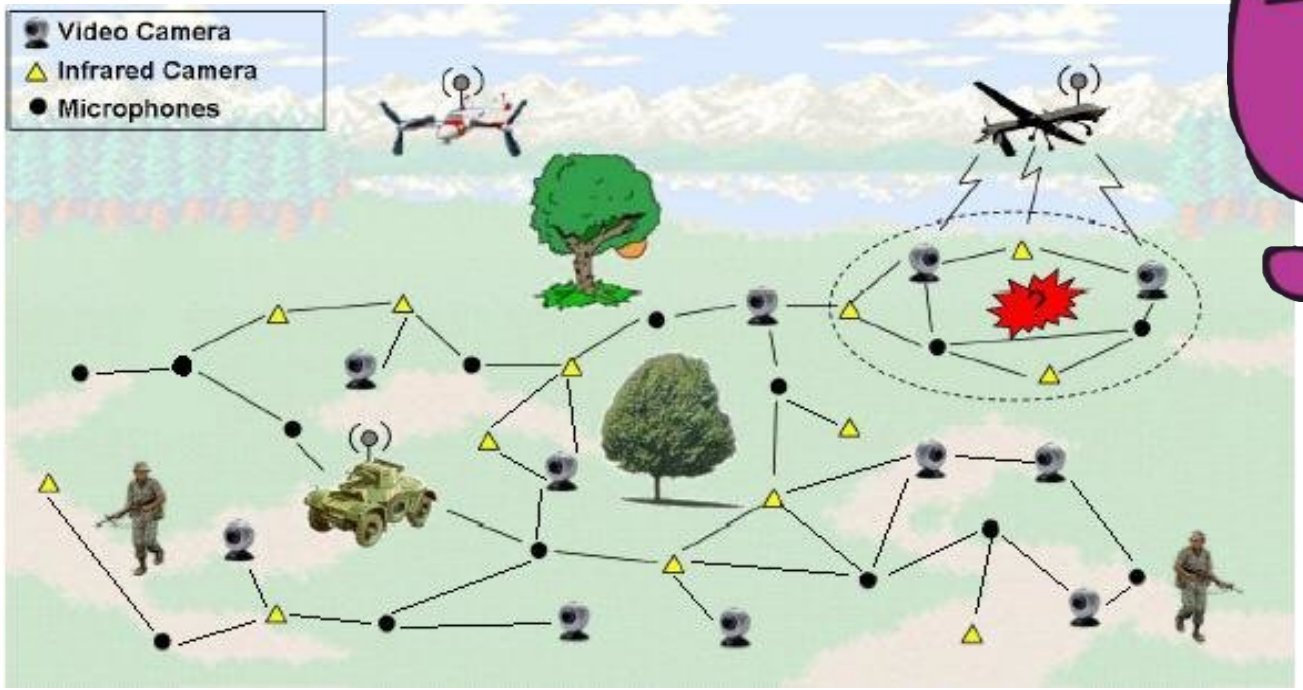
[Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and Roland Norgren. "Understanding uncertainty in cyber-physical systems: A conceptual model." In Proc. ECMFA'16]

Uncertainty related to OCL invariants (system integrity constraints)

Constraints

Sensor.*allInstances()*->*size() >= 3000*

Enemy.*allInstances()*->select(e|*e.distanceTo(self)*)->*size() < 27*



[Image borrowed from Mihai Lica Pura "Ad Hoc Networks and Their Security: A Survey", 2012]

- **Degree of fulfilment** of an OCL invariant
- **Ocurrence uncertainty** of the elements of the system (confidence)

Questions

- Q1. How do we **soften (i.e., relax) OCL invariants**
  - to express some degree of uncertainty about their level of fulfillment
  - To express some degree of confidence on the system elements
- Q2. How do we **represent soft invariants** at the model level
  - so that existing modeling tools can properly operate with them, and modelers and other system stakeholders can reason about them
- Q3. How do we **compose soft invariants** using logical connectives
  - such as and, or, or implies, to produce further soft invariants
  - How is uncertainty propagated through composition?
- Q4. What is the semantics of **invariant independence** in this context?
  - Two invariants are said to be independent if none of them implies the other.

USE: carDriverCRISP.use

File Edit State View Plugins Help

CarDriver
- Classes
  - Person
- Associations
- Invariants
  - Person::AllowDrive
- Pre-/Postconditions

context p : Person **inv** AllowDrive:
(p.age >= 18)

Class diagram

Person
age : Integer

Object diagram

ada:Person
age=17

cyd:Person
age=18

bob:Person
age=17

dan:Person
age=18

Evaluation browser

context p : Person **inv** AllowDrive:
(p.age >= 18)

Person.*allInstances*()->*forAll*(p:Person | (p.age >= 18)) = false
- Person.*allInstances*() = Set{ada,bob,cyd,dan}
- p = @ada
  - (p.age >= 18) = false
    - p.age = 17
- p = @bob
  - (p.age >= 18) = false
    - p.age = 17
- p = @cyd
  - (p.age >= 18) = true
- p = @dan
  - (p.age >= 18) = true

Expand all false | Close

Ready.

USE: carDriverSOFT.use

File Edit State View Plugins Help

CarDriver
- Classes
  - ProbableElement
  - Person
  - Dashboard
- Associations
- Invariants
  - Dashboard::AllowDrive
- Pre-/Postconditions

context self : Dashboard **inv** AllowDrive:
(self.AllowDriveCSL >=
self.AllowDriveRSL)

Class diagram

Dashboard
PersonConfidTh : Integer
AllowDriveRSL : Integer
/AllowDriveCSL : Integer

ProbableElement
confid : Integer

Person
age : Integer

AllowDriveCSL:Integer derive =
let Yes=Person.allInstances->select(p|
p.confid>=PersonConfidTh and p.age>=18)->size in
let No=Person.allInstances->select(p|
p.confid>=PersonConfidTh and not(p.age>=18))->size in
(Yes*10) div (Yes+No)

Object diagram

ada:Person
confid=5
age=17

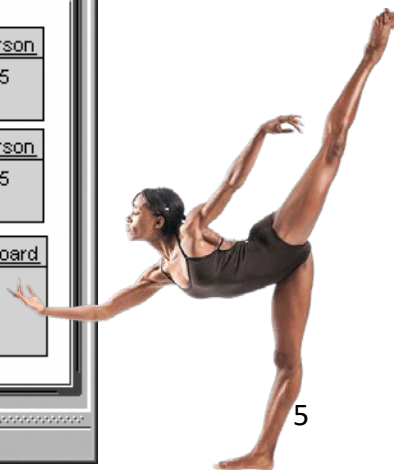
cyd:Person
confid=5
age=18

bob:Person
confid=5
age=17

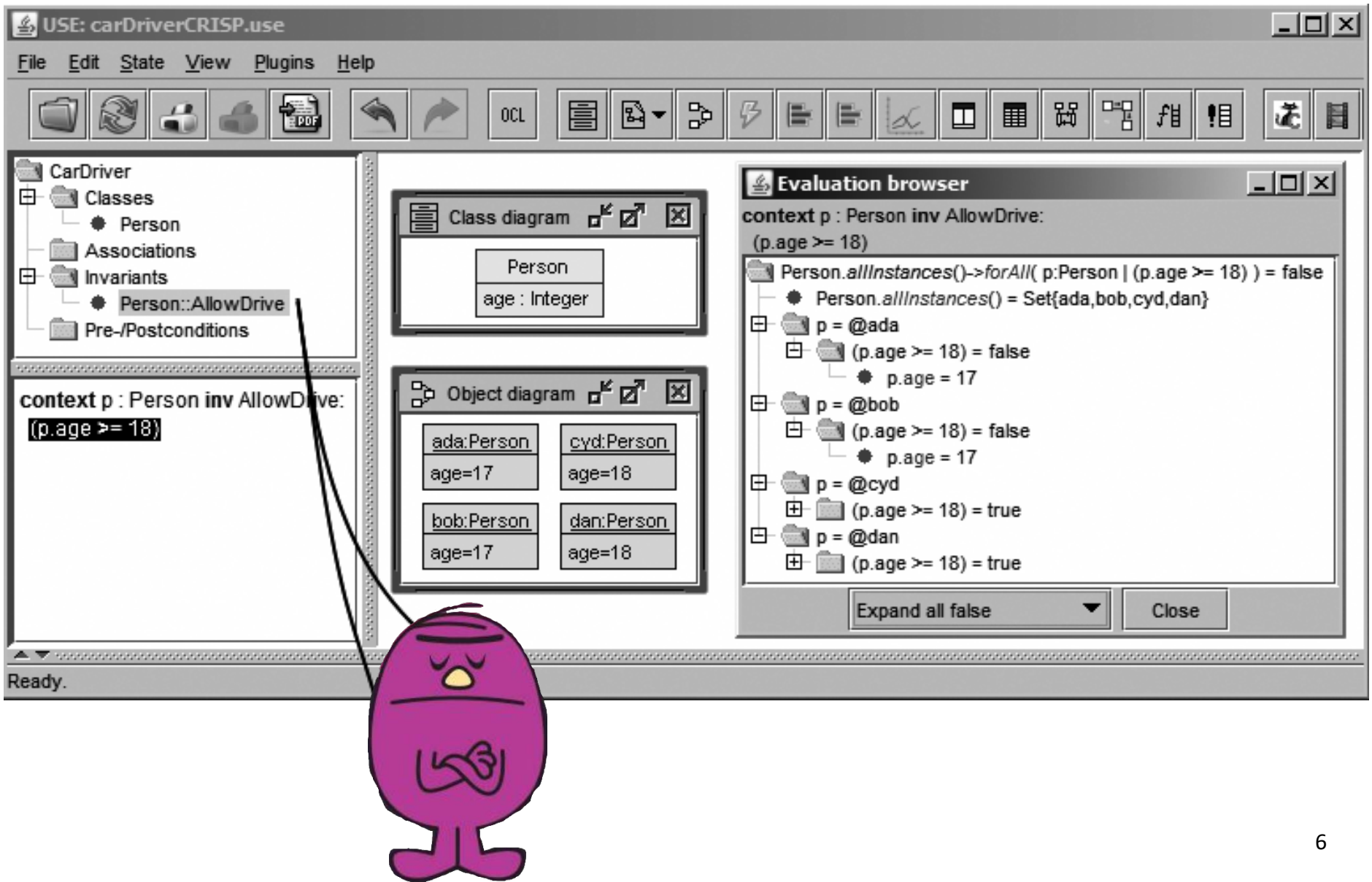
dan:Person
confid=5
age=18

USEDashboard:Dashboard
PersonConfidTh=5
AllowDriveRSL=5
/AllowDriveCSL=5

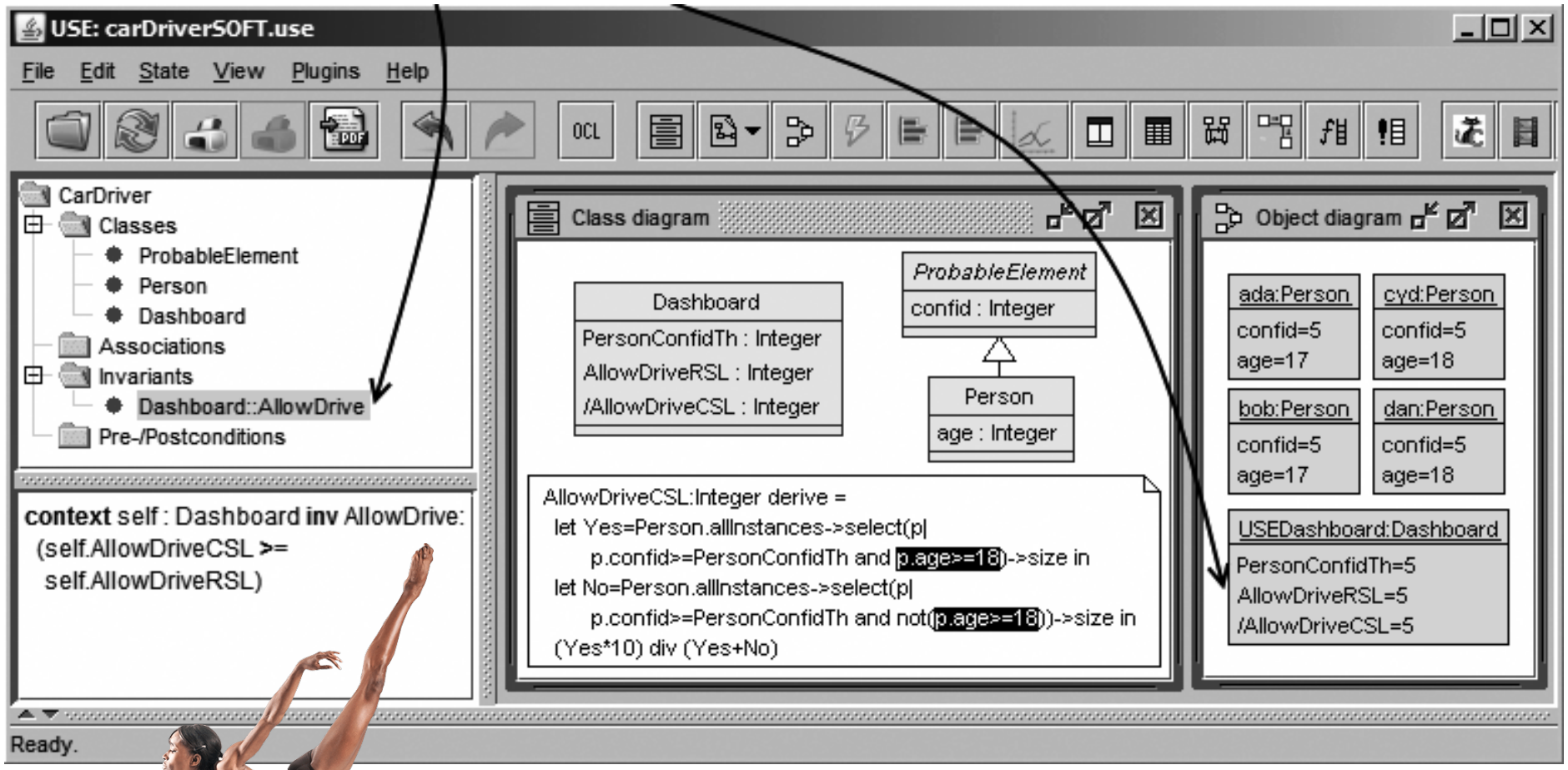
Ready.

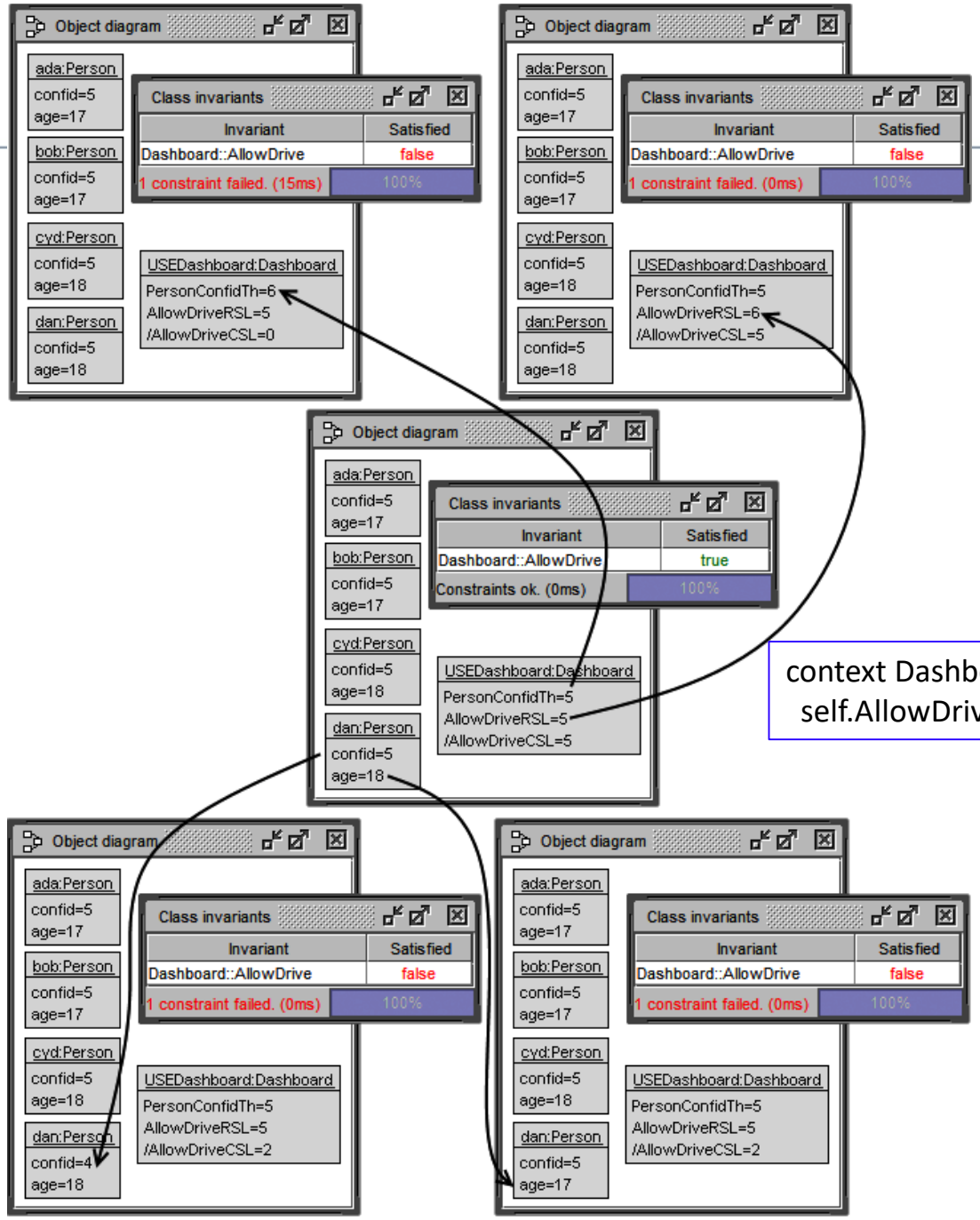


An example of a rigid and inflexible constraint



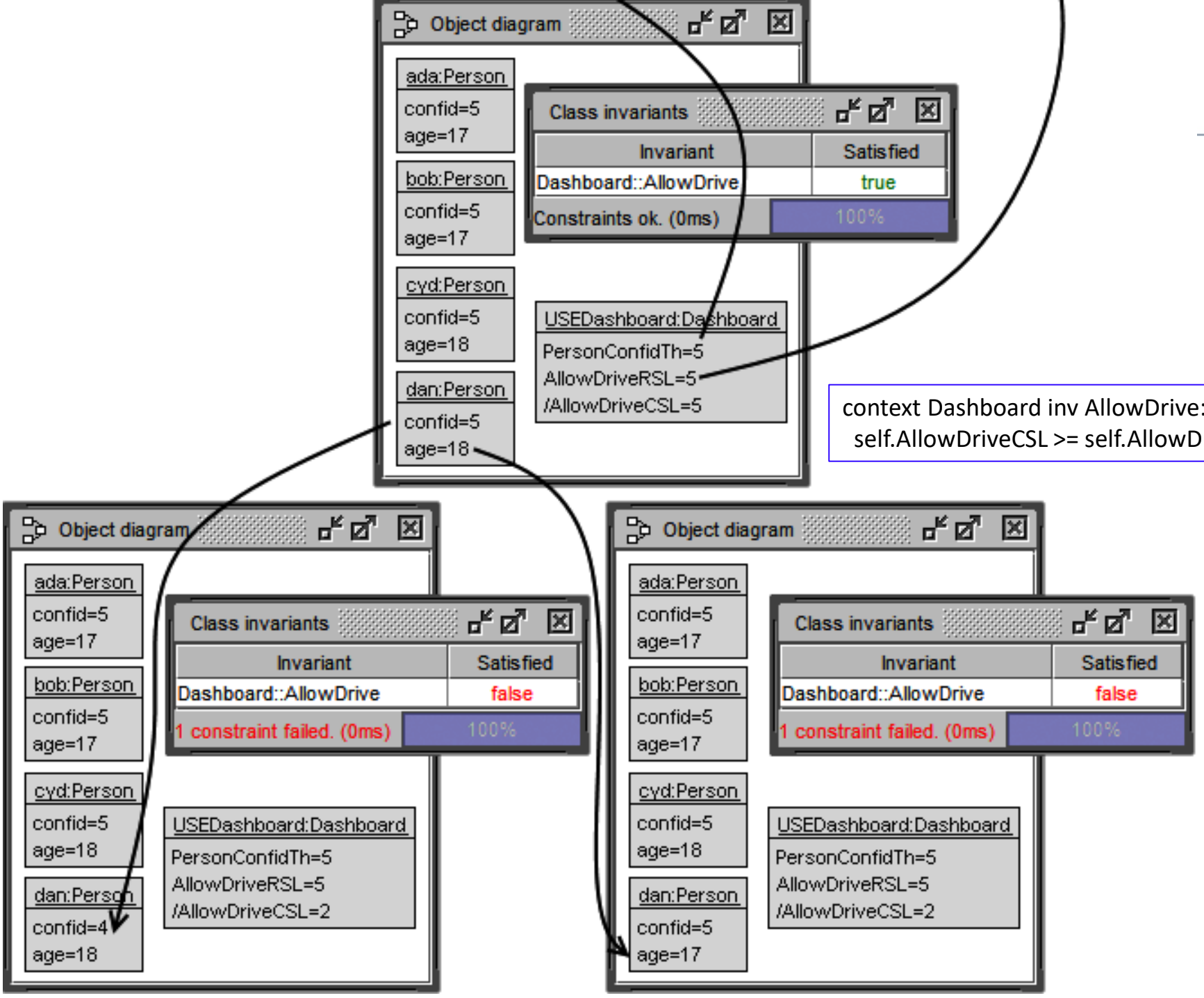
Softening the invariant



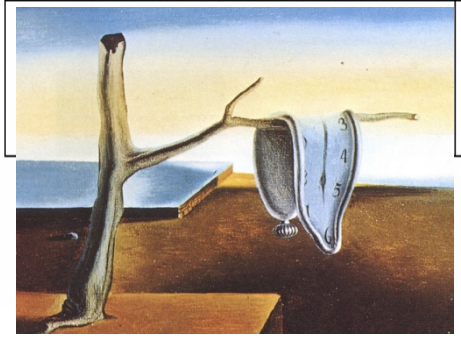


context Dashboard inv AllowDrive:
self.AllowDriveCSL >= self.AllowDriveRSL


Object diagram
ada:Person
confid=5
age=17
bob:Person
confid=5
age=17
cyd:Person
confid=5
age=18
dan:Person
confid=5
age=18
Class invariants
Invariant
Satisfied
Dashboard::AllowDrive
false
1 constraint failed. (15ms)
100%
USEDashboard:Dashboard
PersonConfidTh=6
AllowDriveRSL=5
/AllowDriveCSL=0
Object diagram
ada:Person
confid=5
age=17
bob:Person
confid=5
age=17
cyd:Person
confid=5
age=18
dan:Person
confid=5
age=18
Class invariants
Invariant
Satisfied
Dashboard::AllowDrive
false
1 constraint failed. (0ms)
100%
USEDashboard:Dashboard
PersonConfidTh=5
AllowDriveRSL=6
/AllowDriveCSL=5
Object diagram
ada:Person
confid=5
age=17
bob:Person
confid=5
age=17
cyd:Person
confid=5
age=18
dan:Person
confid=5
age=18
Class invariants
Invariant
Satisfied
Dashboard::AllowDrive
true
Constraints ok. (0ms)
100%
USEDashboard:Dashboard
PersonConfidTh=5
AllowDriveRSL=5
/AllowDriveCSL=5
context Dashboard inv AllowDrive:
self.AllowDriveCSL >= self.AllowDriveRSL


Object diagram
ada:Person
confid=5
age=17
bob:Person
confid=5
age=17
cyd:Person
confid=5
age=18
dan:Person
confid=5
age=18
Class invariants
Invariant
Satisfied
Dashboard::AllowDrive
true
Constraints ok. (0ms)
100%
USEDashboard:Dashboard
PersonConfidTh=5
AllowDriveRSL=5
/AllowDriveCSL=5
context Dashboard inv AllowDrive:
self.AllowDriveCSL >= self.AllowDriveRSL
Object diagram
ada:Person
confid=5
age=17
bob:Person
confid=5
age=17
cyd:Person
confid=5
age=18
dan:Person
confid=4
age=18
Class invariants
Invariant
Satisfied
Dashboard::AllowDrive
false
1 constraint failed. (0ms)
100%
USEDashboard:Dashboard
PersonConfidTh=5
AllowDriveRSL=5
/AllowDriveCSL=2
Object diagram
ada:Person
confid=5
age=17
bob:Person
confid=5
age=17
cyd:Person
confid=5
age=18
dan:Person
confid=5
age=17
Class invariants
Invariant
Satisfied
Dashboard::AllowDrive
false
1 constraint failed. (0ms)
100%
USEDashboard:Dashboard
PersonConfidTh=5
AllowDriveRSL=5
/AllowDriveCSL=2

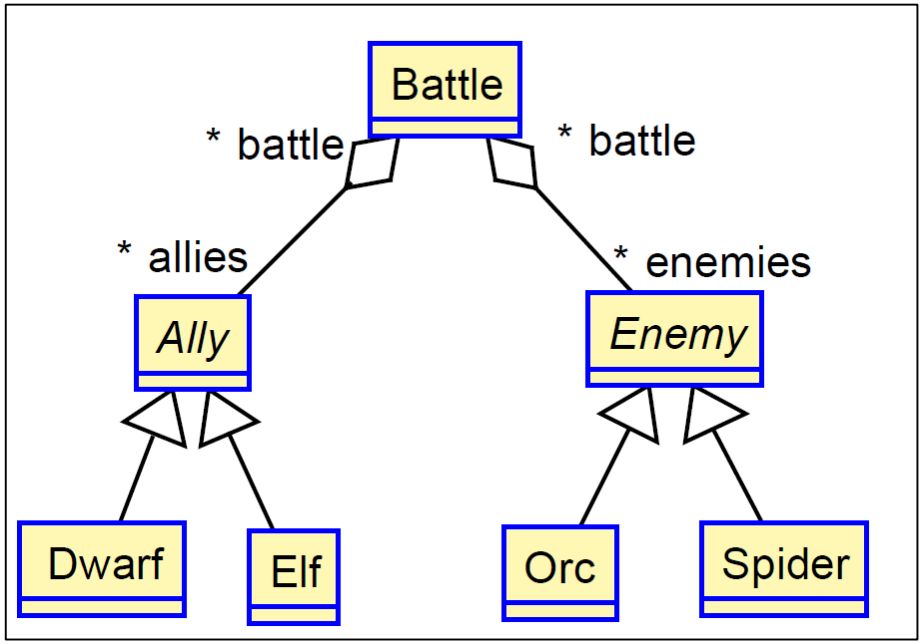
Achieving levels of flexibility

Model element uncertainty: Class confidence threshold (Th), Object confidence (confid)	*Informal invariant classification*			
10			*Completely crisp*	
. . .				
0	*Completely soft*			
	0	. . .	10	Constraint uncertainty: Required invariant satisfaction level (RSL)

Confidence about model element

Percentage of population required to fulfill invariant

Soft Invariant “Patterns”



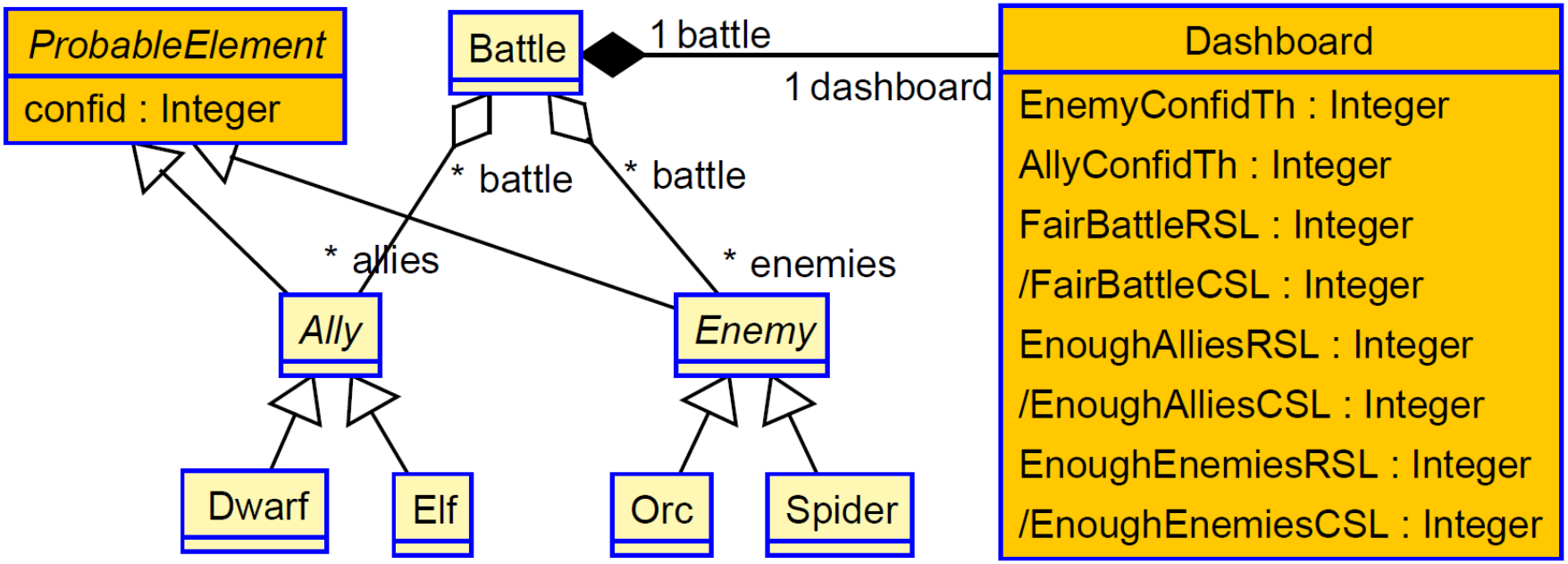
context Battle inv FairBattle: self.enemies->size = self.allies->size

context Battle inv EnoughAllies: self.allies->notEmpty

context Battle inv EnoughEnemies: self.enemies->notEmpty

Soft Invariant “Patterns”

context Battle inv FairBattle: self.enemies->size = self.allies->size



FairBattleRSL : Integer -- Required satisfaction level (user defined)

FairBattleCSL : Integer derive = -- Current satisfaction level

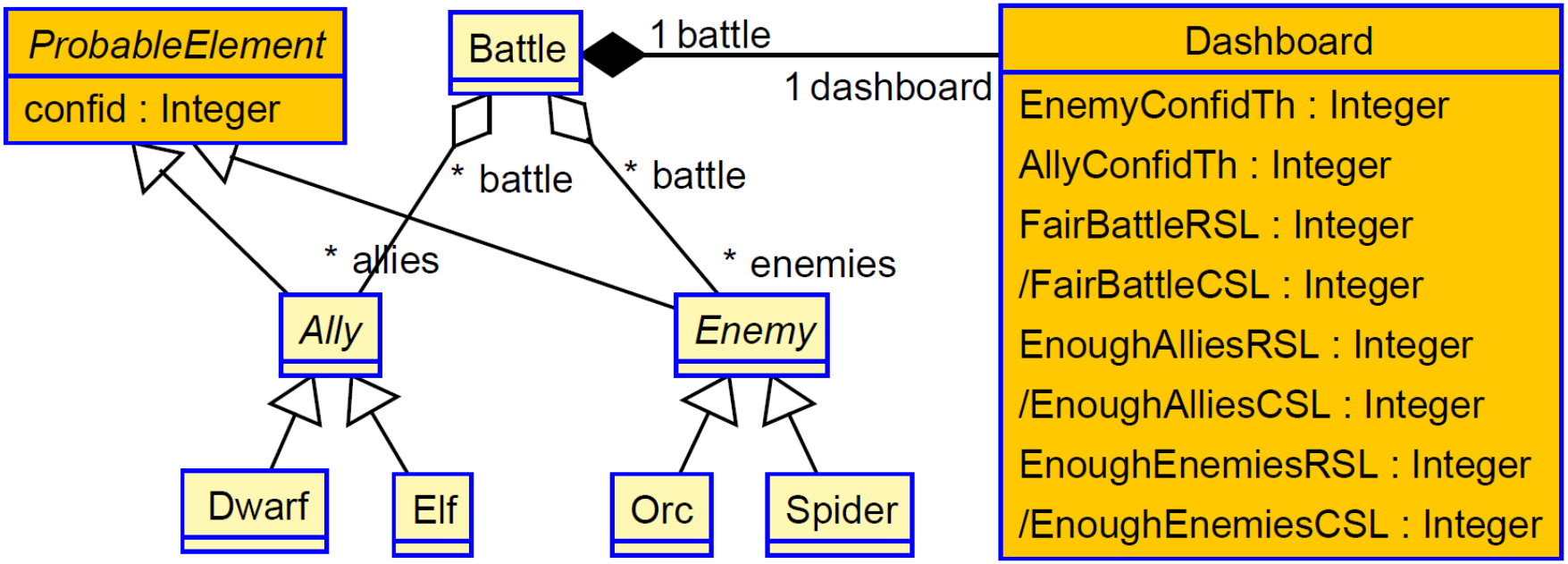
let YesE=battle.enemies->select(e|e.confid>=EnemyConfidTh)->size in -- # real enem

let YesA=battle.allies->select(a|a.confid>=AllyConfidTh)->size in -- # real allies

1 - (YesA - YesE).abs()/(YesE + YesA)

Soft Invariant “Patterns”

context Battle inv FairBattle: self.enemies->size = self.allies->size

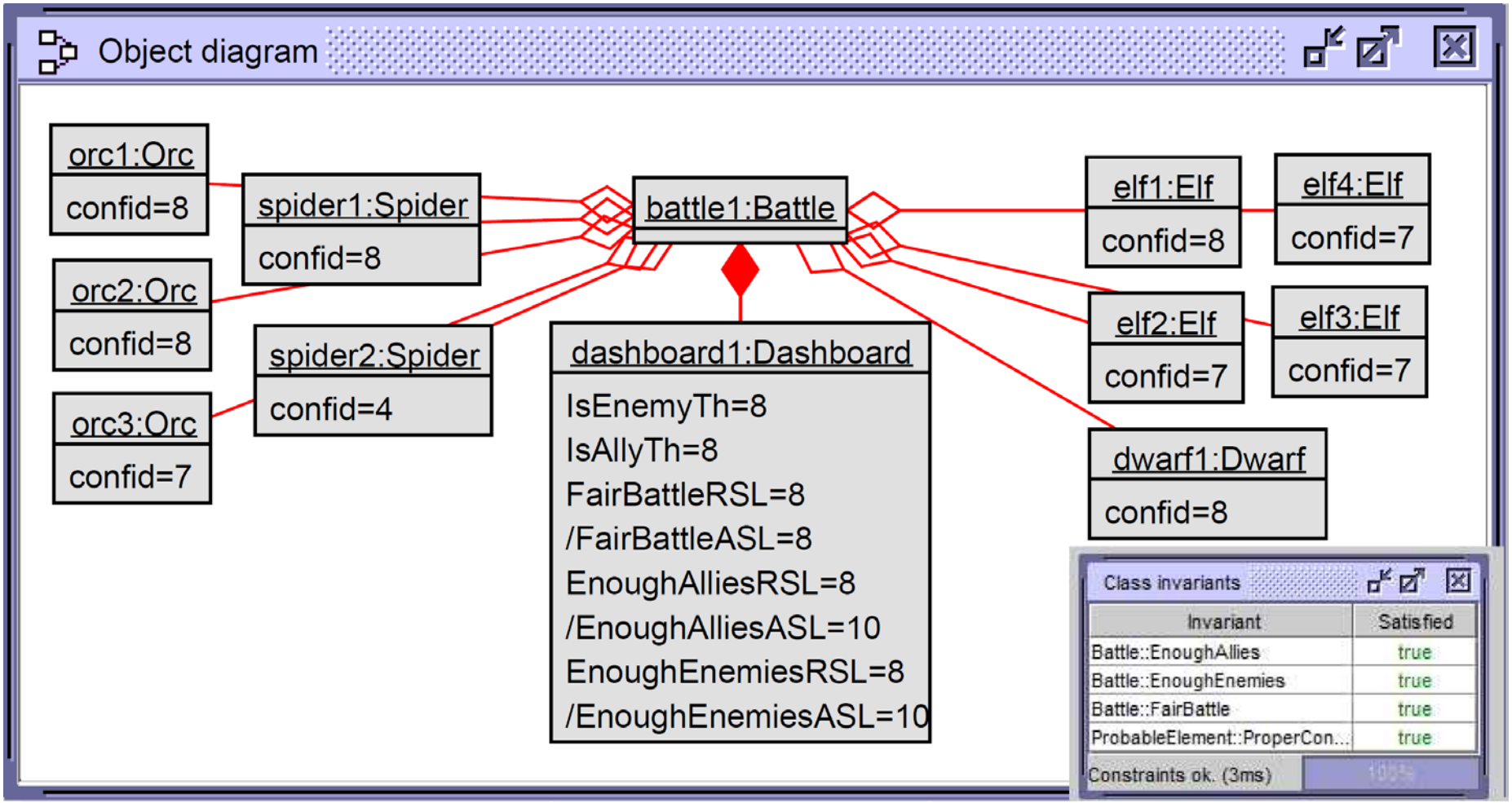


FairBattleRSL : Integer -- Required satisfaction level (user defined)

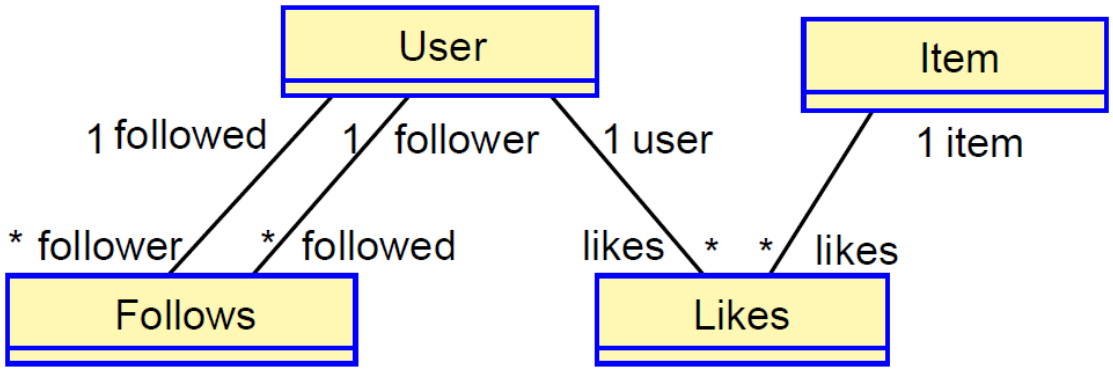
FairBattleCSL : Integer derive = -- Current satisfaction level

```
let YesE=battle.enemies->select(e|e.confid>=EnemyConfidTh)->size in -- # real enem
let YesA=battle.allies->select(a|a.confid>=AllyConfidTh)->size in -- # real allies
let diff = if YesA>=YesE then YesA-YesE else YesE-YesA endif in
  10 - ((diff * 10) div (YesE + YesA))
```

Asking the model validator to create valid object models

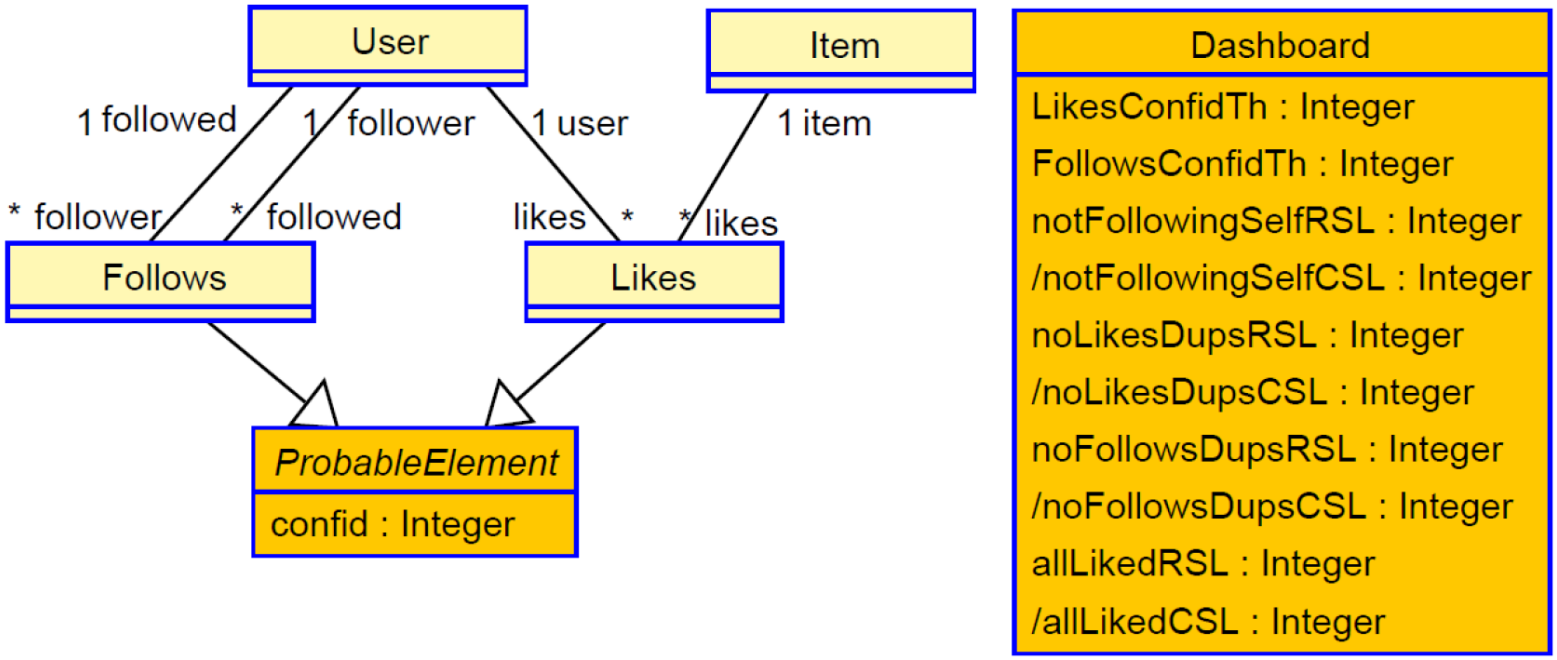


More Soft Invariant “Patterns”




```
context u:User inv notFollowingSelf: u.follower->forAll(fR | fR.follower<>u)
context i:Item inv allLiked: i.likes->notEmpty
context l1:Likes inv noLikesDups:
    not Likes.allInstances->exists(l2 |
            not (l1<>l2 implies (l1.user<>l2.user or l1.item<>l2.item)))
context f1:Follows inv noFollowsDups:
    Follows.allInstances->forAll(f2 |
            f1<>f2 implies (f1.follower<>f2.follower or f1.followed<>f2.followed))
```

More Soft Invariant "Patterns"



context u:User inv **notFollowingSelf**: -- *"few" followers with "enough" confidence*
context i:Item inv **allLiked**: -- *"few" not liked with "enough" confidence*
context l1:Likes inv **noLikesDups**: -- *"few" dups with "enough" confidence*
context f1:Follows inv **noFollowsDups**: -- *"few" dups with "enough" confidence*

Soft Invariant Patterns Summary

Crisp invariant pattern	Corresponding soft invariant
Invariant1 ≡ X->size = Y->size or X->size >= Y->size	let a=X->select(x|x.confid>=XConfidTh)->size in let b=Y->select(y|y.confid>=YConfidTh)->size in let Invariant1CSL = if (a+b)=0 then 0 else (1-((a-b).abs()/(a+b))) endif in Invariant1CSL >= Invariant1RSL
Invariant2 ≡ X.allInstances->notEmpty()	let a=X.allInstances->select(x|x.confid>=XConfidTh)->size in let b=X.allInstances->size in let Invariant2CSL = if b=0 then 0 else (a/b) endif in Invariant2CSL >= Invariant2RSL
Invariant3 ≡ X->select(x|P(x))->size()>0	let a=X->select(x|x.confid>=XConfidTh and P(x))->size in let b=X->select(x|x.confid>=XConfidTh and not P(x))->size in let Invariant3CSL = if (a+b)=0 then 0 else a/(a+b) endif in Invariant3CSL >= Invariant3RSL
Invariant4 ≡ not (OtherInvariant)	let Invariant4CSL = 1.0 - OtherInvariantCSL in Invariant4CSL >= Invariant4RSL
Invariant5 ≡ InvX and InvY	let Invariant5CSL = InvXCSL * InvYCSL in Invariant5CSL >= Invariant5RSL
Invariant6 ≡ InvX or InvY	let Invariant6CSL = InvXCSL + InvYCSL - InvXCSL*InvYCSL in Invariant6CSL >= Invariant6RSL

Reasoning about degrees of satisfaction

- Extending the "satisfaction" operator "$\models$" to a new one: "degree of satisfaction"

$$\models\!\!\!\sim \; : \mathcal{M} \times \mathcal{LP} \rightarrow [0..1]$$

- Properties*
  - $m \models\!\!\!\sim (\neg\phi_1) = 1.0 - m \models\!\!\!\sim \phi_1$
  - $m \models\!\!\!\sim (\phi_1 \wedge \phi_2) = m \models\!\!\!\sim \phi_1 * m \models\!\!\!\sim \phi_2$
  - $m \models\!\!\!\sim (\phi_1 \vee \phi_2) = m \models\!\!\!\sim \phi_1 + m \models\!\!\!\sim \phi_2 - (m \models\!\!\!\sim \phi_1 * m \models\!\!\!\sim \phi_2)$

(*) Assuming that the predicates that define the invariants are "independent"

Invariant independence (traditional definitions)

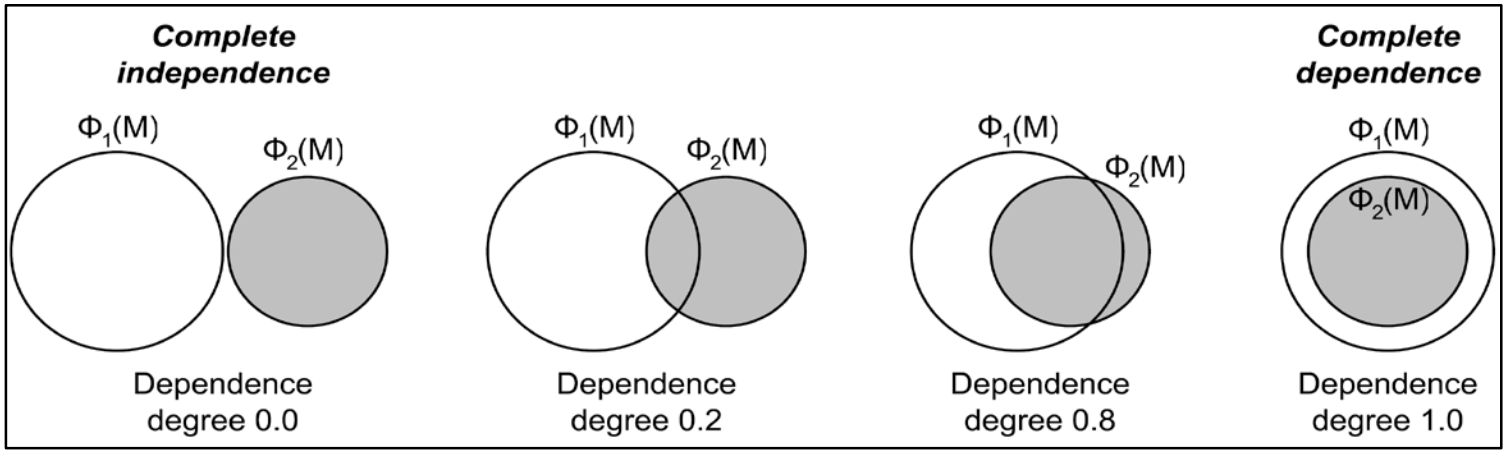
- Traditional definitions of "dependence" and "independence"
  - Dependence:

$$\forall m \in \mathcal{M} \bullet (m \models \phi_1 \Rightarrow m \models \phi_2)$$

  - Independent:

$$\exists\, m_1, m_2 \in \mathcal{M} \bullet (m_1 \models \phi_1 \wedge m_1 \not\models \phi_2) \wedge (m_2 \models \phi_2 \wedge m_2 \not\models \phi_1)$$

Invariant independence (new definitions)



- Dependence: $\forall m \in \mathcal{M} \bullet (m \sqsupseteq \phi_1) \leqslant (m \sqsupseteq \phi_2)$ or $\forall m \in \mathcal{M} \bullet \phi_1(m) \leqslant \phi_2(m)$
- Independent: $\exists\, m_1, m_2 \in \mathcal{M} \bullet\ \phi_1(m_1) > \phi_2(m_2)\ \wedge\ \phi_1(m_2) > \phi_2(m_1)$
- "Completely" independent $\forall\ m \in \mathcal{M} \bullet (\phi_1 \wedge \phi_2)(m) = \phi_1(m) * \phi_2(m)$
- *"Degree" of independence*: $i(\phi_1, \phi_2) = 1 - \dfrac{\#\{m \in \mathcal{M} \bullet (\phi_1 \wedge \phi_2)(m) > 0\}}{\#\{m \in \mathcal{M} \bullet (\phi_1 \vee \phi_2)(m) > 0\}}$

Checking independence in a practical way with the USE Model Validator

- Invariant dependence:
  - ask the MV to look for a model m such that: $\phi_2(m) < \phi_1(m)$
- Invariant independence:
  - ask the MV to look for two models m_1 and m_2 such that: $\phi_1(m_1) > \phi_2(m_1)$ and $\phi_2(m_2) > \phi_1(m_2)$
- Invariant complete independence:
  - ask the MV to look for a model m such that:

$$(\phi_1 \wedge \phi_2)(m) \neq \phi_1(m) * \phi_2(m)$$

Summary and outlook on future work

- Softened invariants to
  - Express the degree of satisfaction that is required for the invariant to be fulfilled
  - Express level of confidence in the elements in the system (occurrence uncert.)
- Reasoning about soft invariants
  - Algebra of operations on soft invariants (and, or, implies)
  - Analysis of dependency/independency between soft invariants
- What's next?
  - Capture further invariant "patterns"
  - Further "validation use cases" for soft invariants (satisfiability, completion, …)
  - Empirical studies with expert modelers
  - Covering further types of uncertainty in the invariants