Employing the tool USE for Model-Based Engineering

Martin Gogolla
University of Bremen, Germany
Database Systems Group

Outline

- View on modeling
- USE features
- Classifying terms
- Transformation models
- Filmstrip models

Outline

- View on modeling
- USE features
- Classifying terms
- Transformation models
- Filmstrip models

A View on Modeling

Why do engineers build models?

- To understand ... problems and solutions
  Models as "Means for Knowledge Acquisition"

- To communicate ... model and design intent
  Models as "Means for Knowledge Transfer"

- To predict ... interesting characteristics of the system under study
  Models as "Surrogates"

- To specify ... the implementation of the system under study
  Models as "Blueprints"

Building models is realized by selecting statements through abstraction, i.e., reduction of information preserving properties relative to a given set of concerns

From: Bran Selic, UML2 Tutorial @ MoDELS 2012

System under study

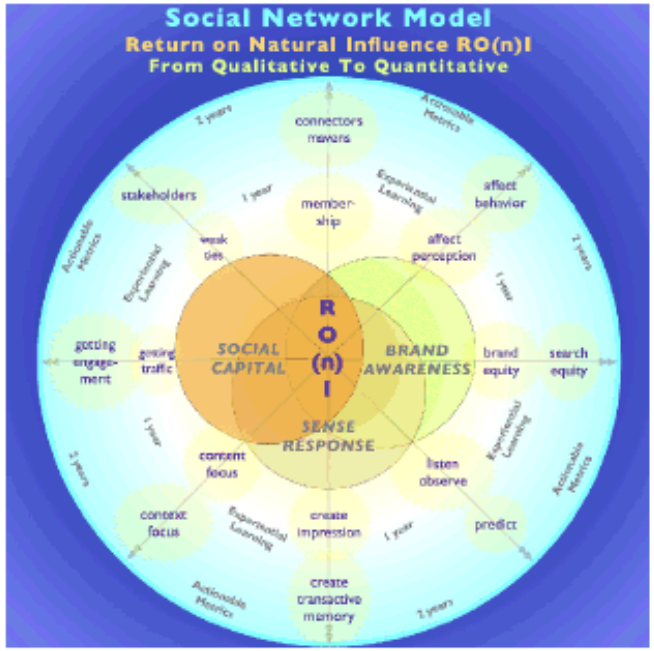
Abstraction

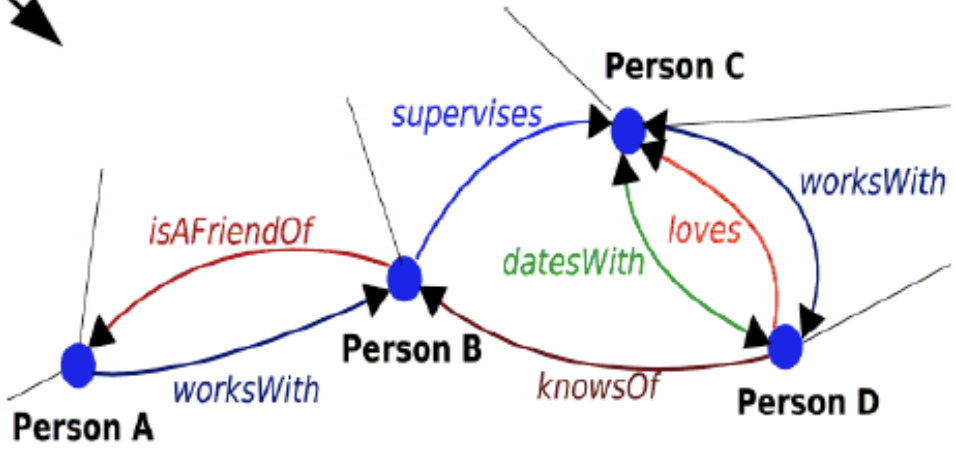
Statements (Models)

$$\frac{\partial k_i}{\partial t} = m_r\left(\frac{1}{t} + m_s\frac{k_i}{\sum k}\right) = \frac{1}{t}\left(m_r + \frac{m_s}{2(1+m_s)}k_i\right)$$

$$k_i(t) = B\left(\frac{t}{t_i}\right)^{1/A} - C$$

$$F(k_i) = P(\tilde{k} \leqslant k_i) = P(\tilde{t} \geqslant t_i) = \frac{1}{t}(t - t_i)$$





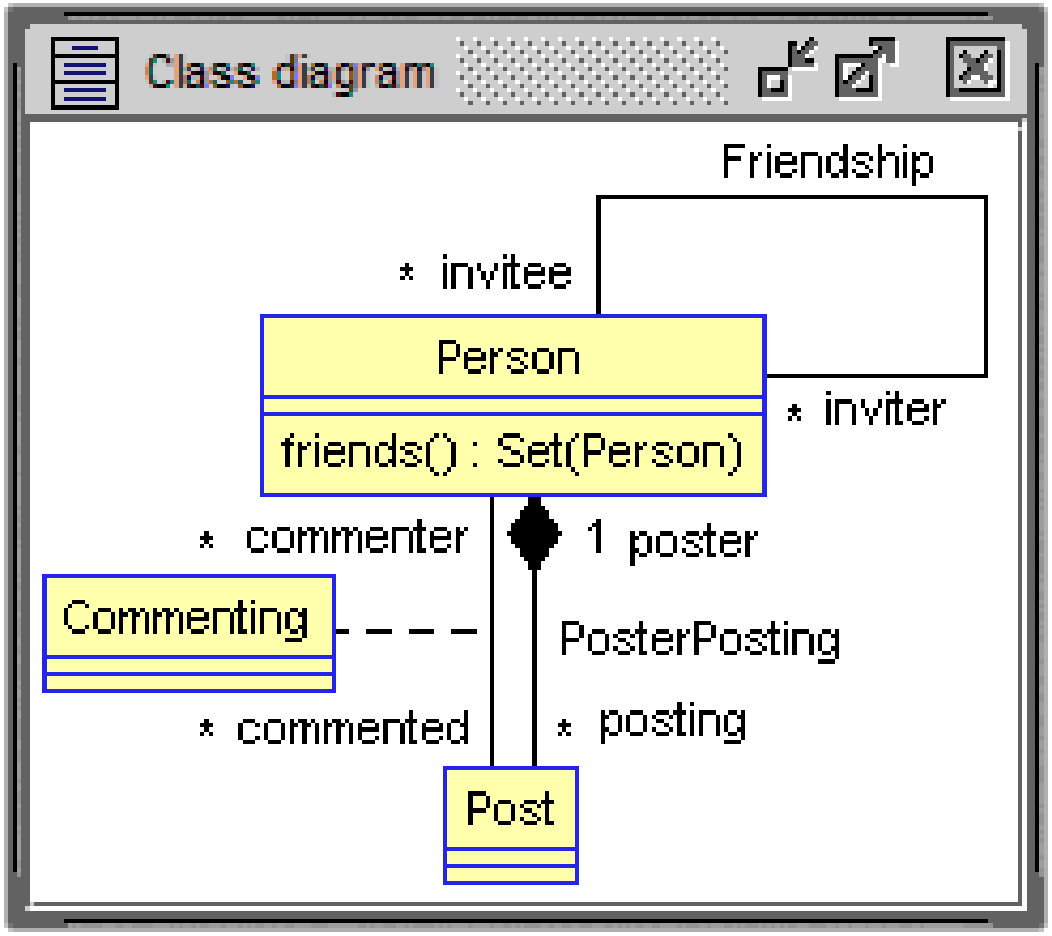
Outline

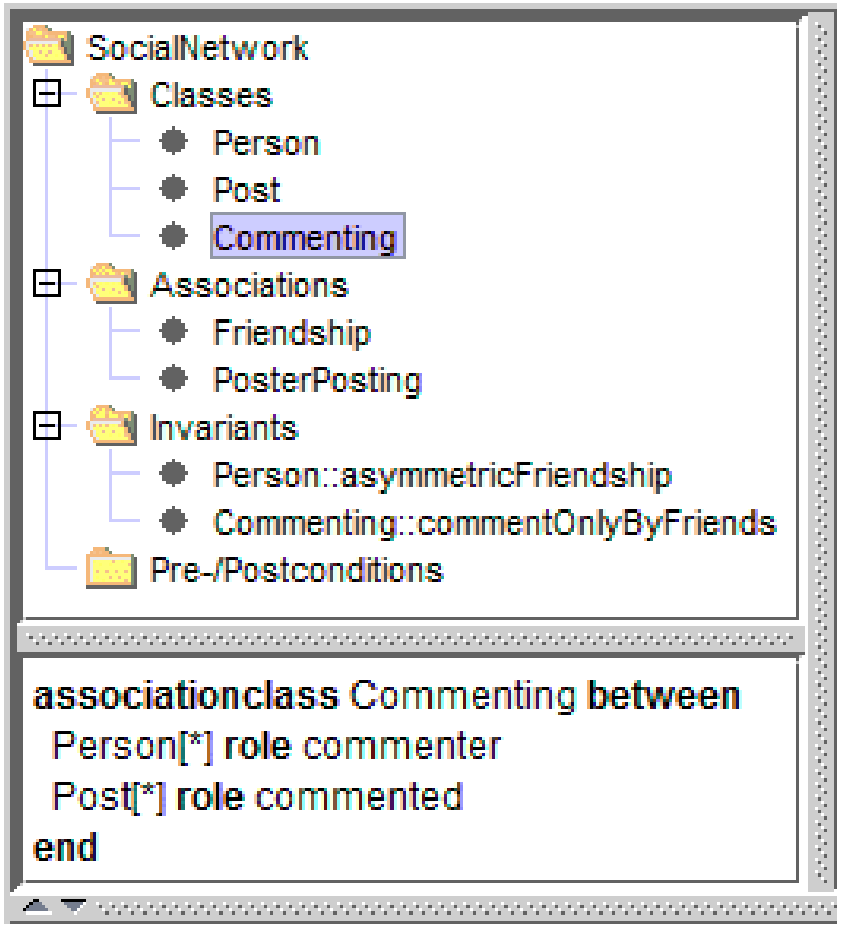
- View on modeling
- USE features
- Classifying terms
- Transformation models
- Filmstrip models

(Textual) Modeling with UML and OCL within USE (UML-based Specification Environment)

- Class diagrams (classes, associations, generalization, abstract classes, association clases, aggregation, composition, ...)
- Protocol State Machines (states and guarded transitions)
- Support for full OCL with all collection kinds (sets, bags, sequences, ordered sets) and operations (forAll, select, collect, iterate, closure, ...)
- Central assurance technique: Expressing scenarios (test cases); UML object and sequence diagrams
- Employing OCL for ad-hoc queries, class invariants, operation pre- and postconditions, operation definitions (for side-effect free operations), state invariants, transition pre- and postconditions
- Employing SOIL (Simple Ocl-like Imperative Language) for realizing operations manipulating the system state
- Validation: Are we building the right product?
  Verification: Are we building the product right?

Running Example



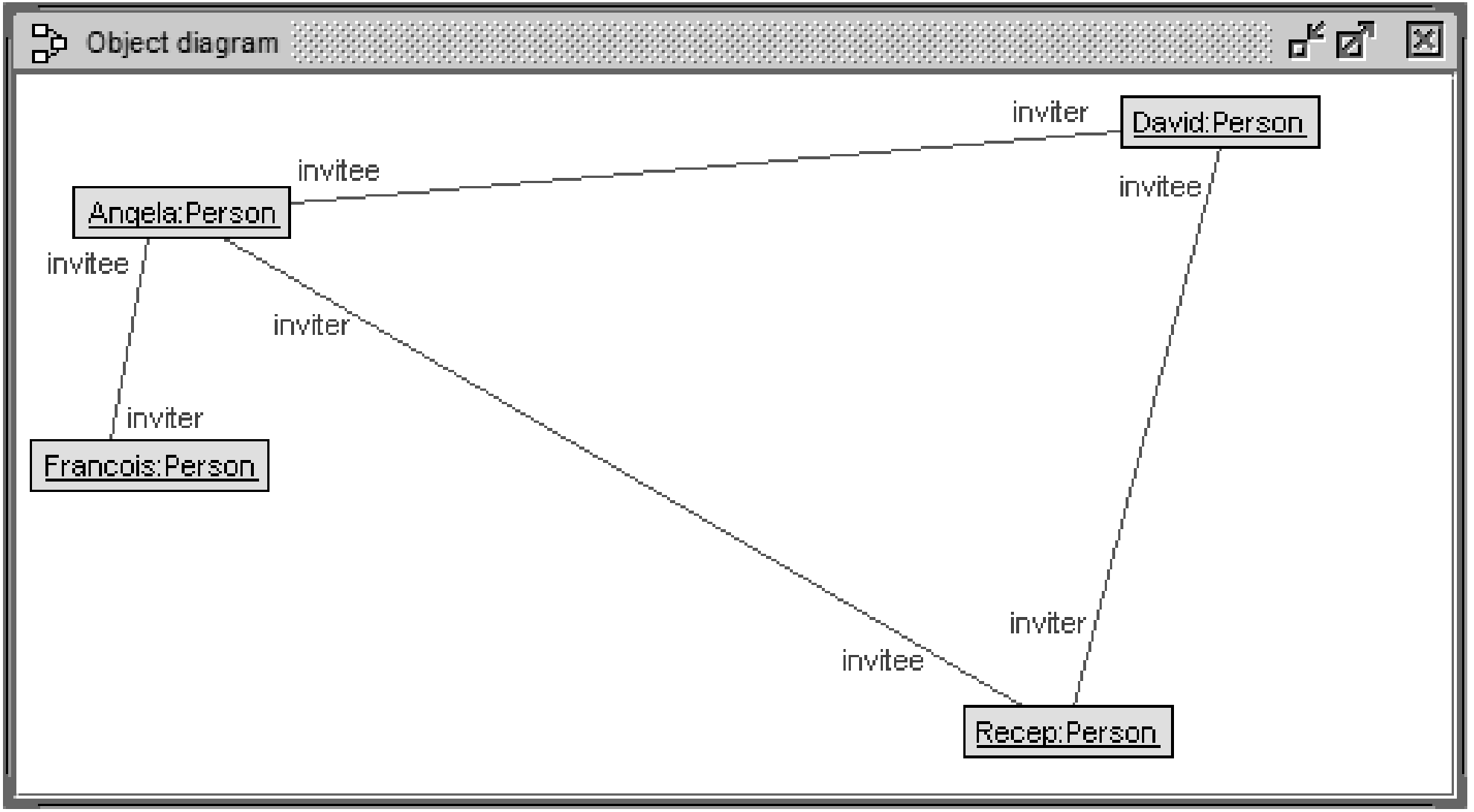



```
Person::friends():Set(Person)=invitee->union(inviter)

context Person inv asymmetricFriendship:
  invitee->intersection(inviter)->isEmpty()

context Commenting inv commentOnlyByFriends:
  commented.poster.friends()->includes(commenter)
```

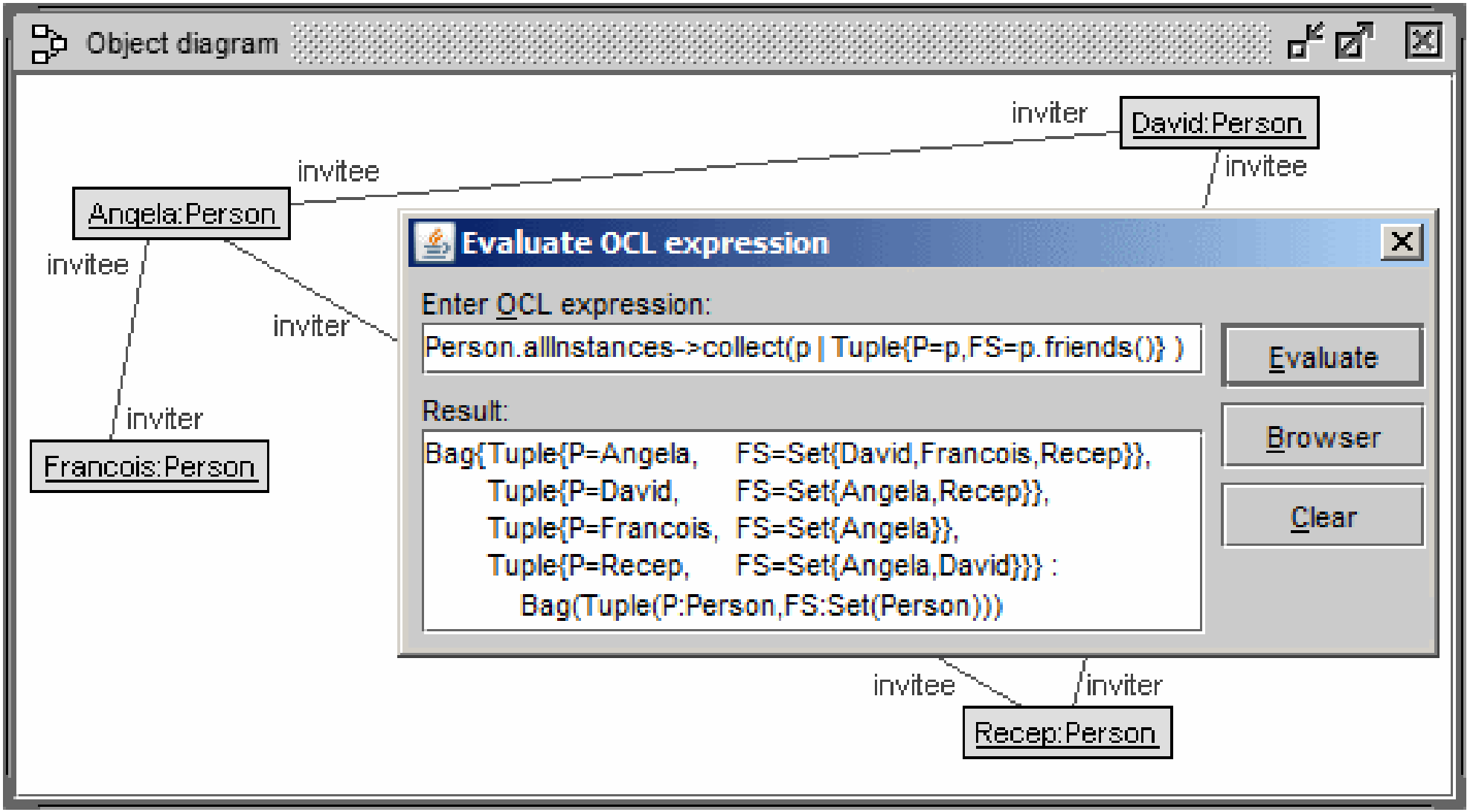
Running Example in Textual Form

```
model SocialNetwork

class Person
operations
  friends():Set(Person)=invitee->union(inviter)
end

association Friendship between
  Person [*] role inviter
  Person [*] role invitee
end

class Post
end

composition PosterPosting between
  Person [1] role poster
  Post   [*] role posting
end

associationclass Commenting between
  Person [*] role commenter
  Post   [*] role commented
end

context Person inv asymmetricFriendship:
  invitee->intersection(inviter)->isEmpty()

context Commenting inv commentOnlyByFriends:
  commented.poster.friends()->includes(commenter)
```

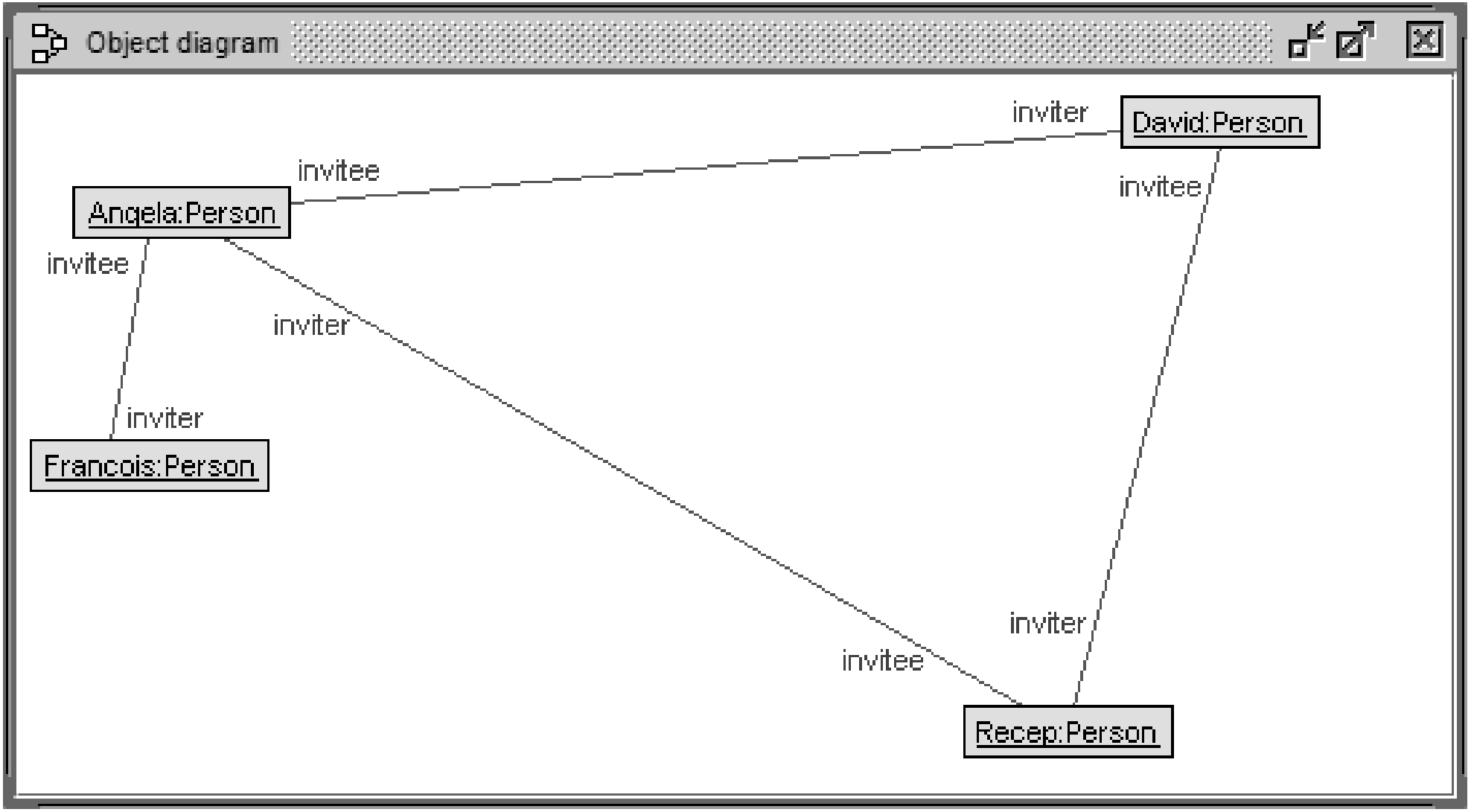
Constructing a System State



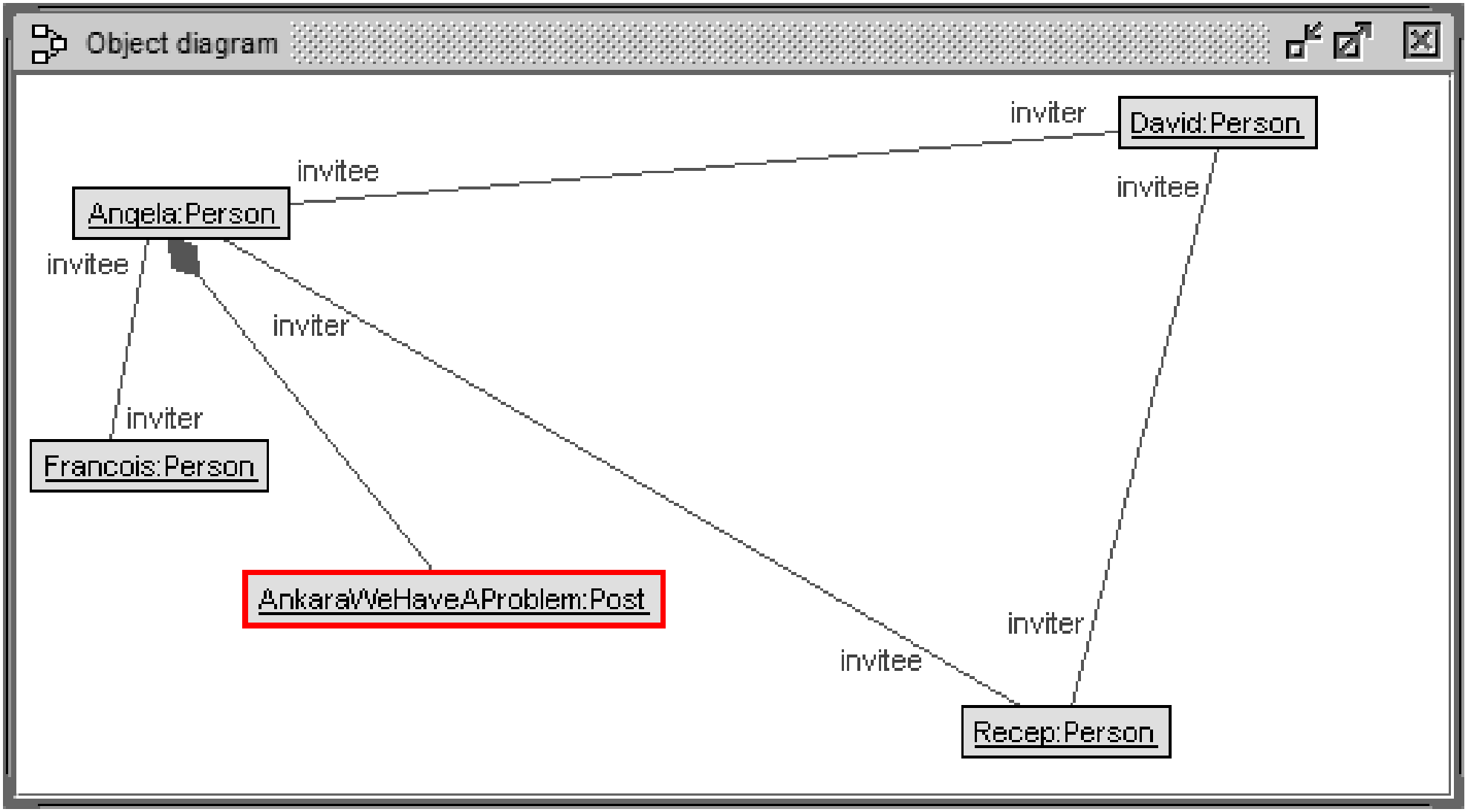
Constructing a System State



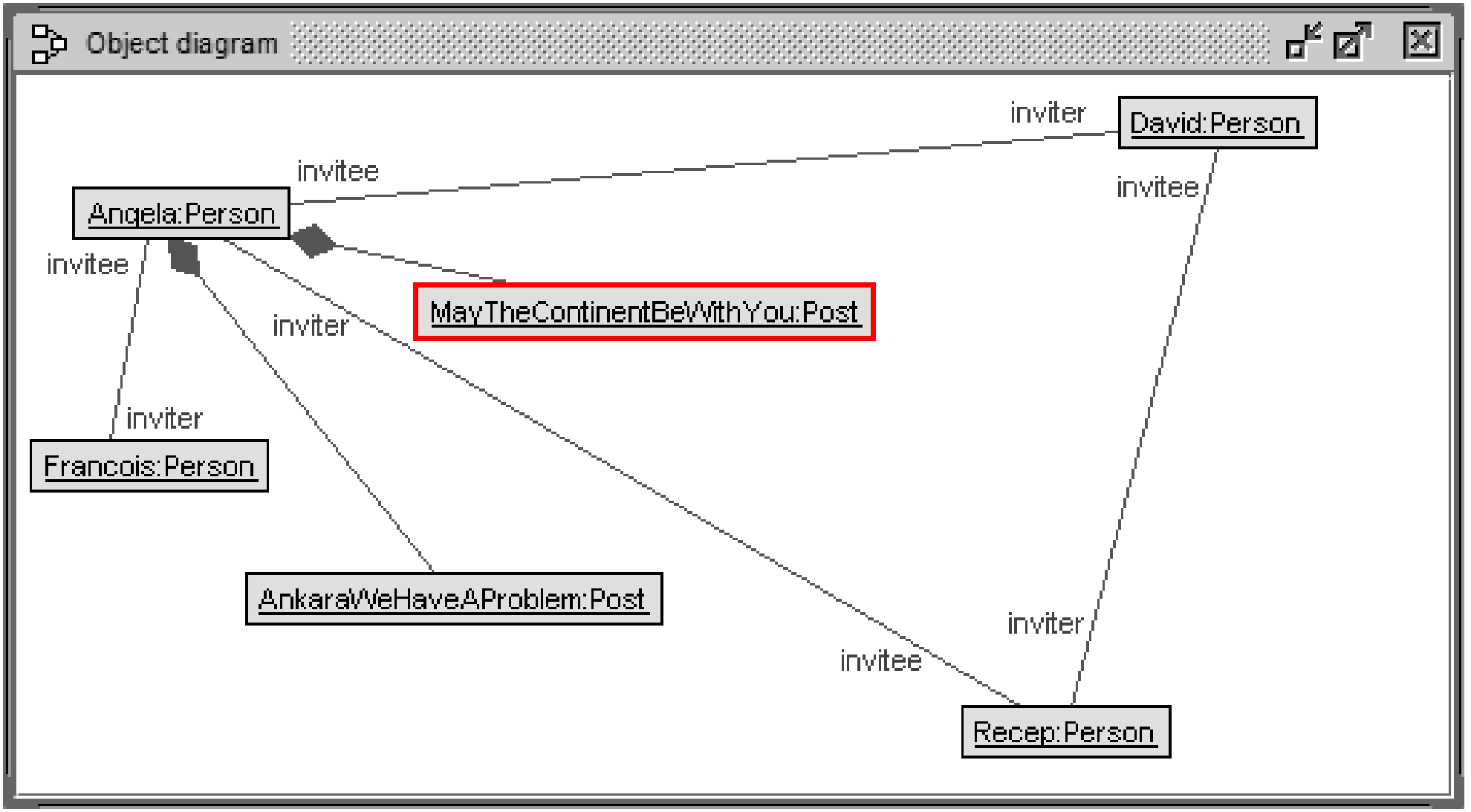
Constructing a System State



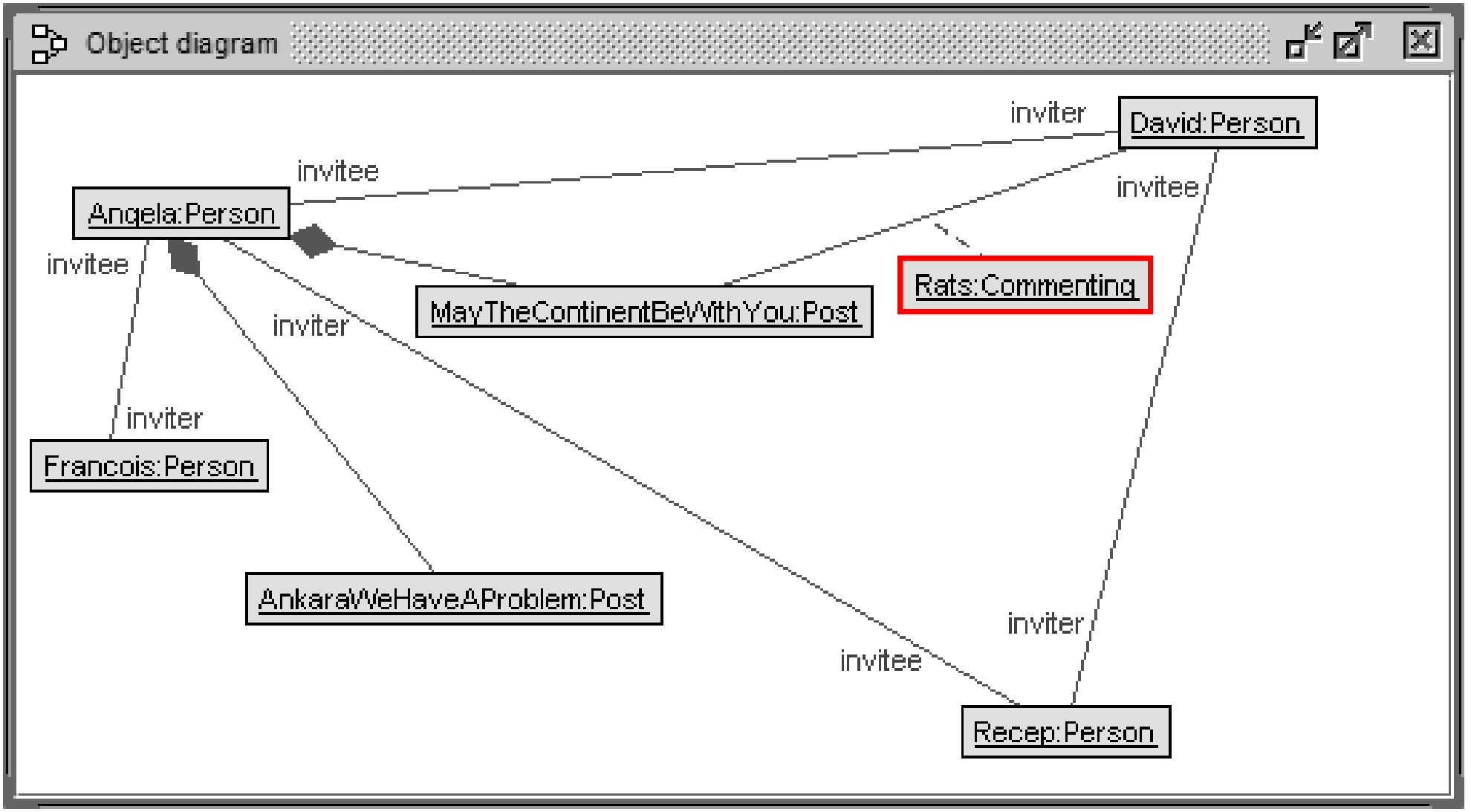
Constructing a System State



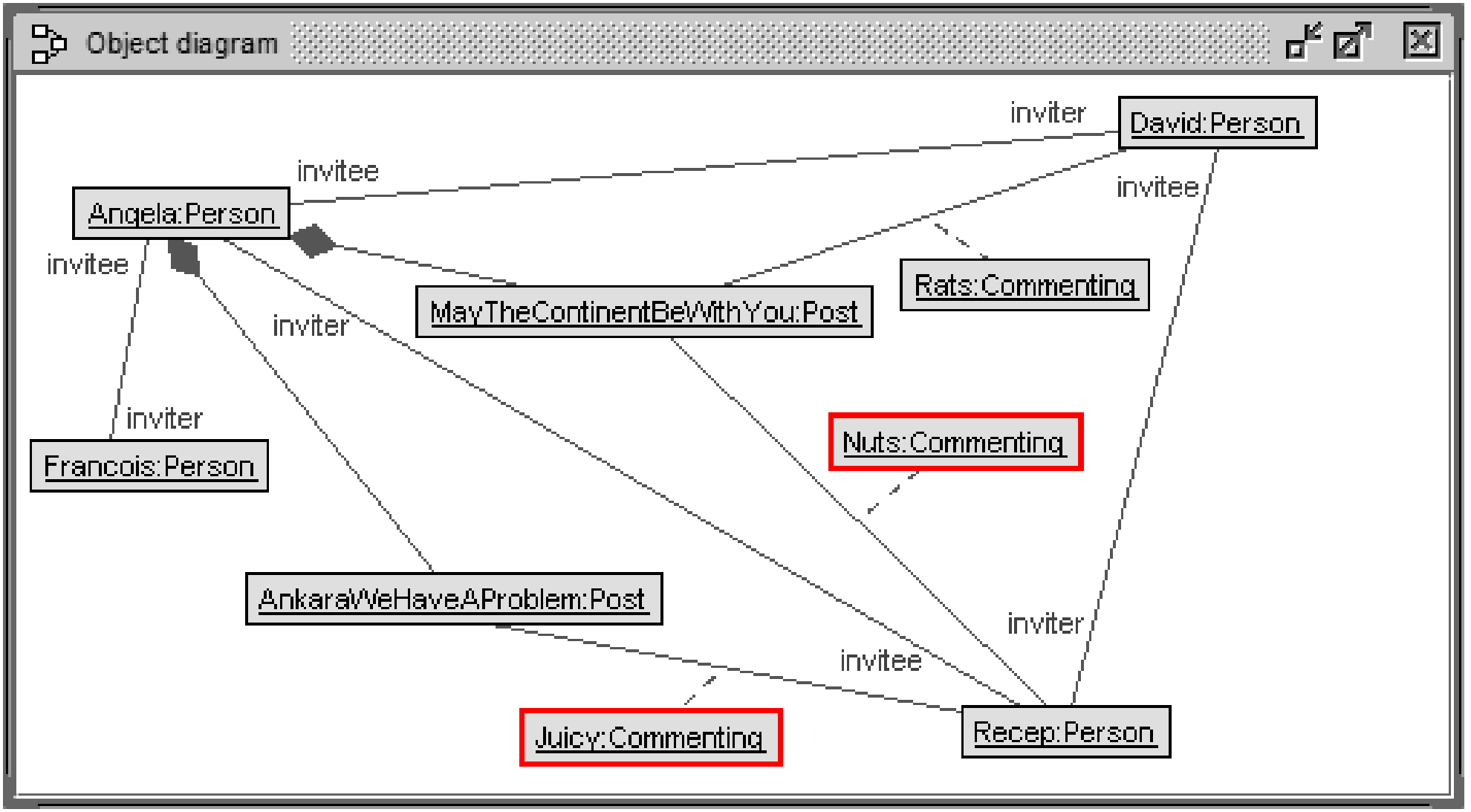
Constructing a System State



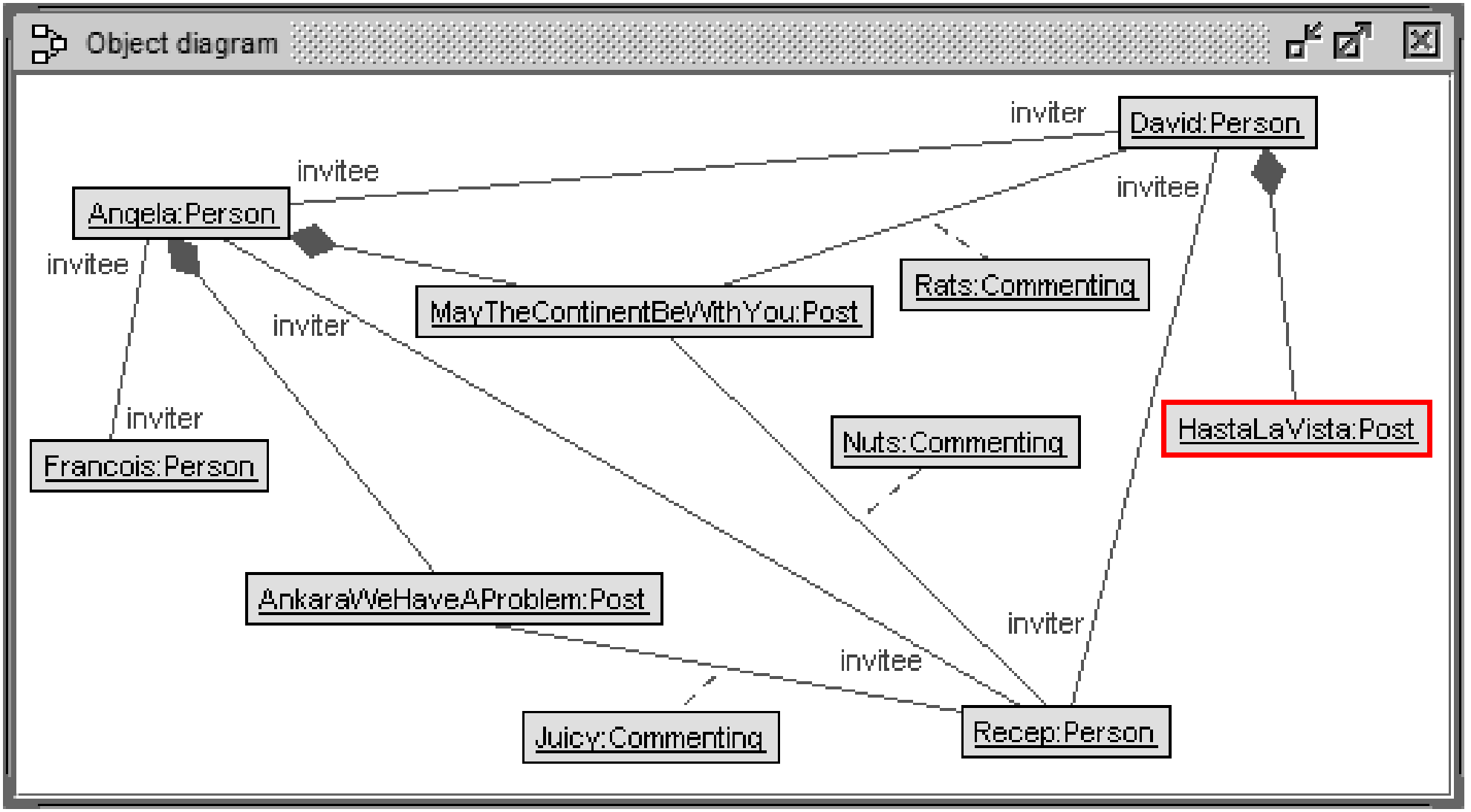
Constructing a System State



Constructing a System State



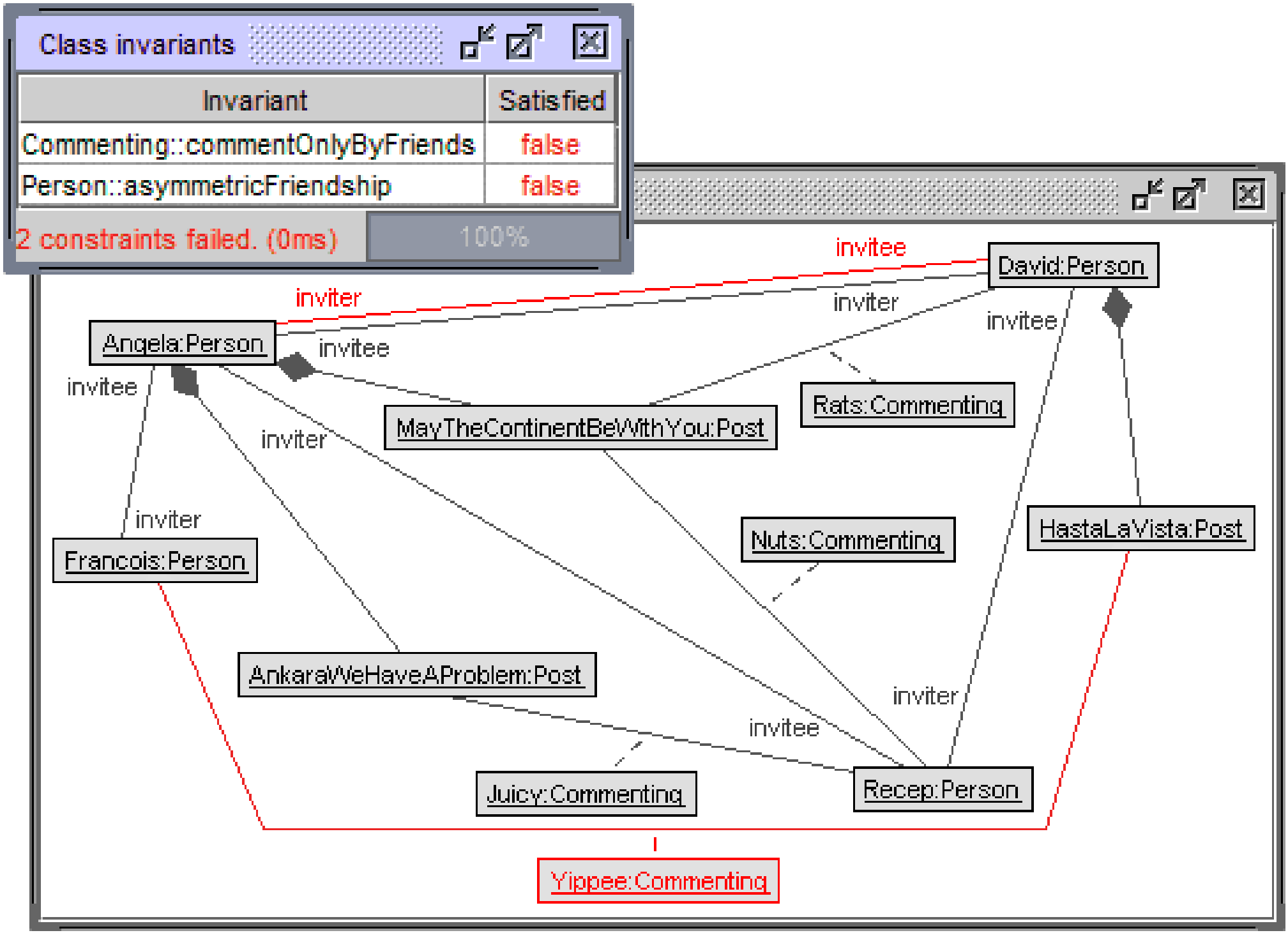
Constructing a System State



Constructing a System State with the Model Validator

- Object diagram (system state) automatically constructed by (so-called) Model Validator (MV)
- Based on a transformation of UML and OCL models into the relational logic of Kodkod/Alloy
- Configuration determines
  - Finite sets of data types values
  - Finite sets for population of classes and associations
  - Finite sets for attribute values
- (a) Construct single object diagram
  (b) Enumerate all object diagrams determined by configuration
- Various use cases
- Enumerations of all object diagrams guided by (so-called) classifying terms (OCL queries) in order to achieve few interesting, diverse object diagrams

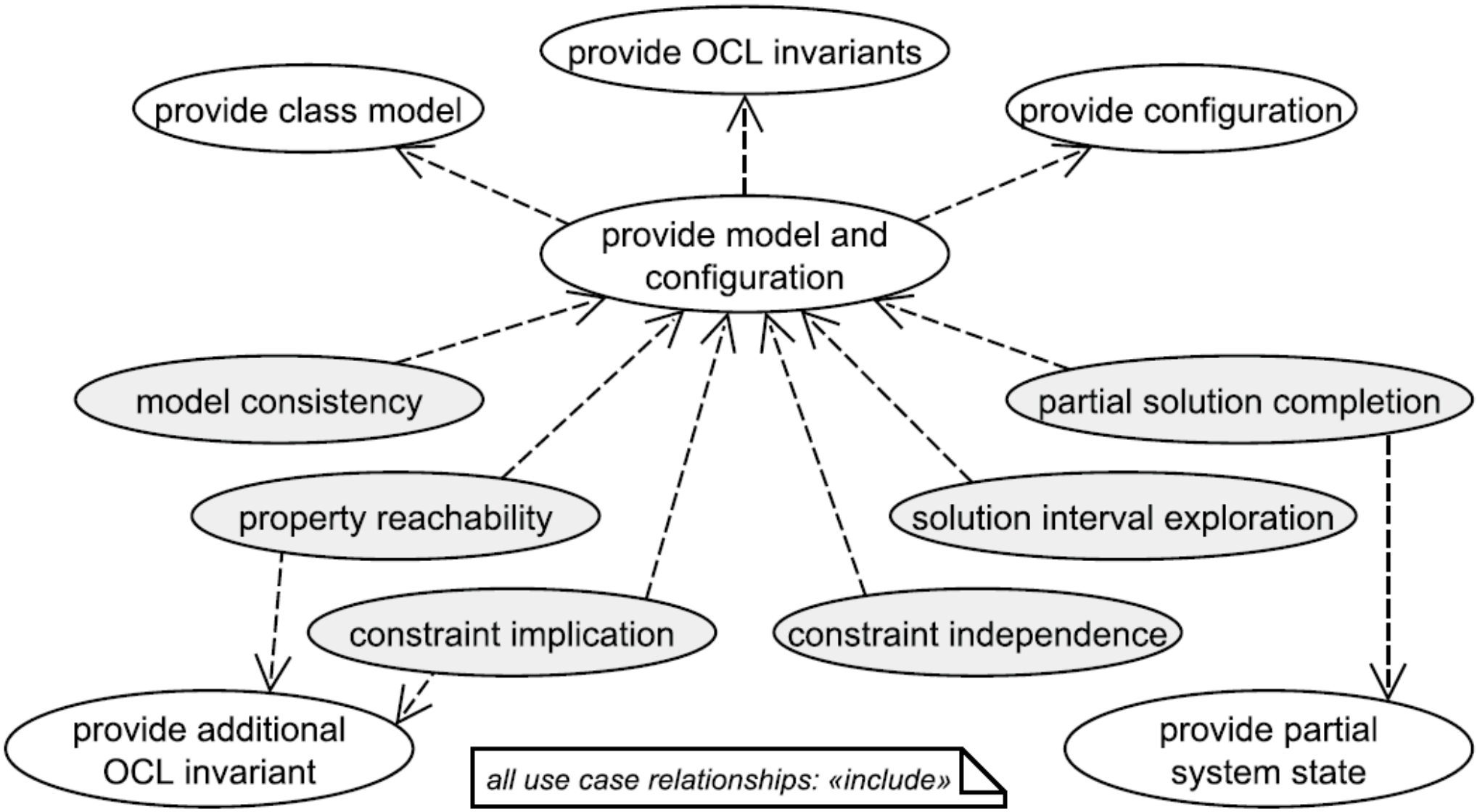
Violating the Invariants



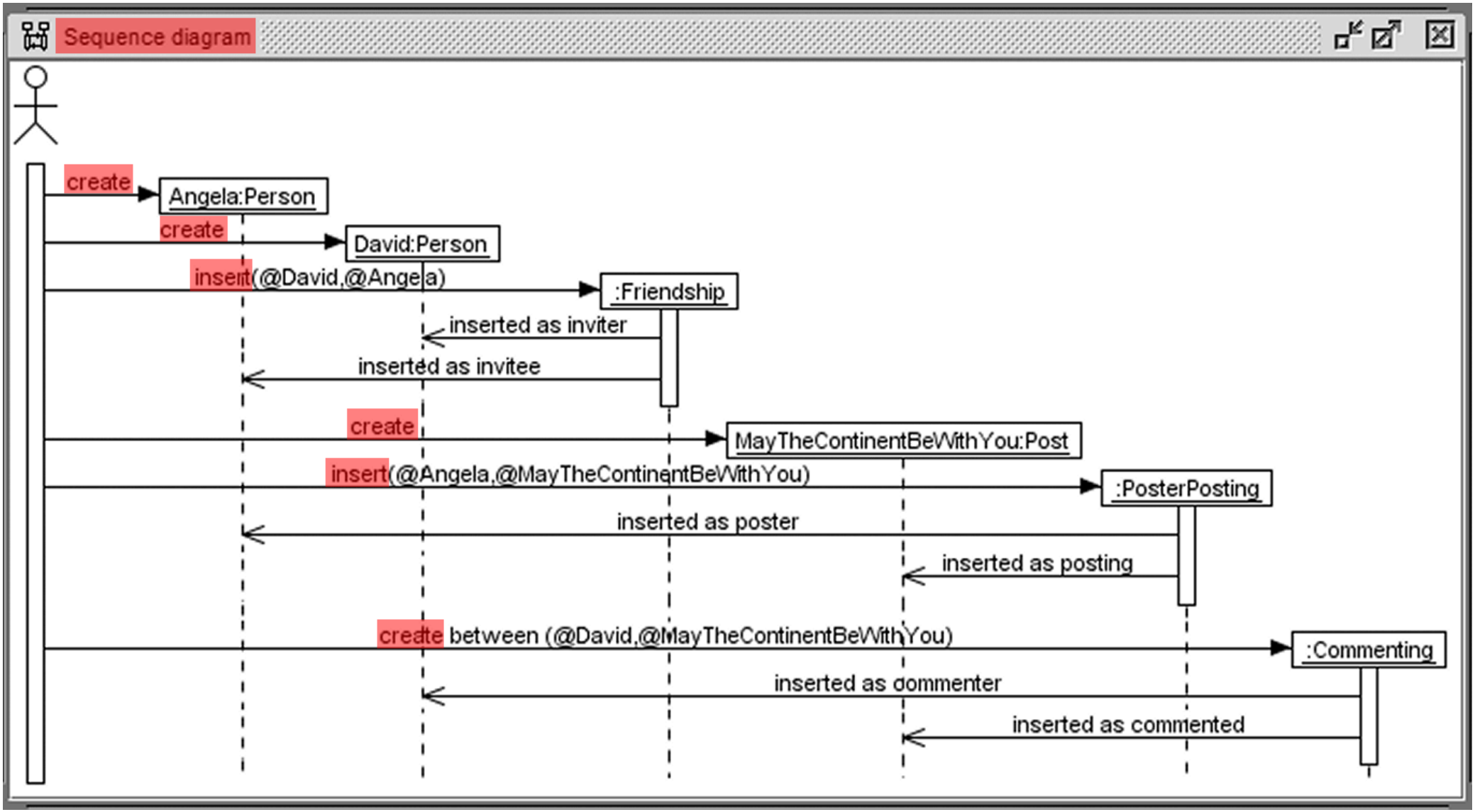
Reasoning about the model

- Can I comment my own postings?
- Can all classes be instantiated?
  Can all classes and associations be instantiated?
- Are the invariants independent from each other?
  Independent: neither (invA |= invB) nor (invB |= invA) holds
- Can I invite myself to be my own friend?
- Can invariant asymetricFriendship be equivalently expressed as
  context p1,p2:Person inv:
    not (p1.invitee->includes(p2) implies p2.invitee->includes(p1))
- Is this equivalent to:
  ... p1.inviter->includes(p2) implies p2.inviter->includes(p1) ...
- Can I comment a posting twice?
- Is the model correct in the sense that
  (a) a stated (partial) object diagram set is accepted as valid and
  (b) another stated (partial) object diagram set is considered invalid?

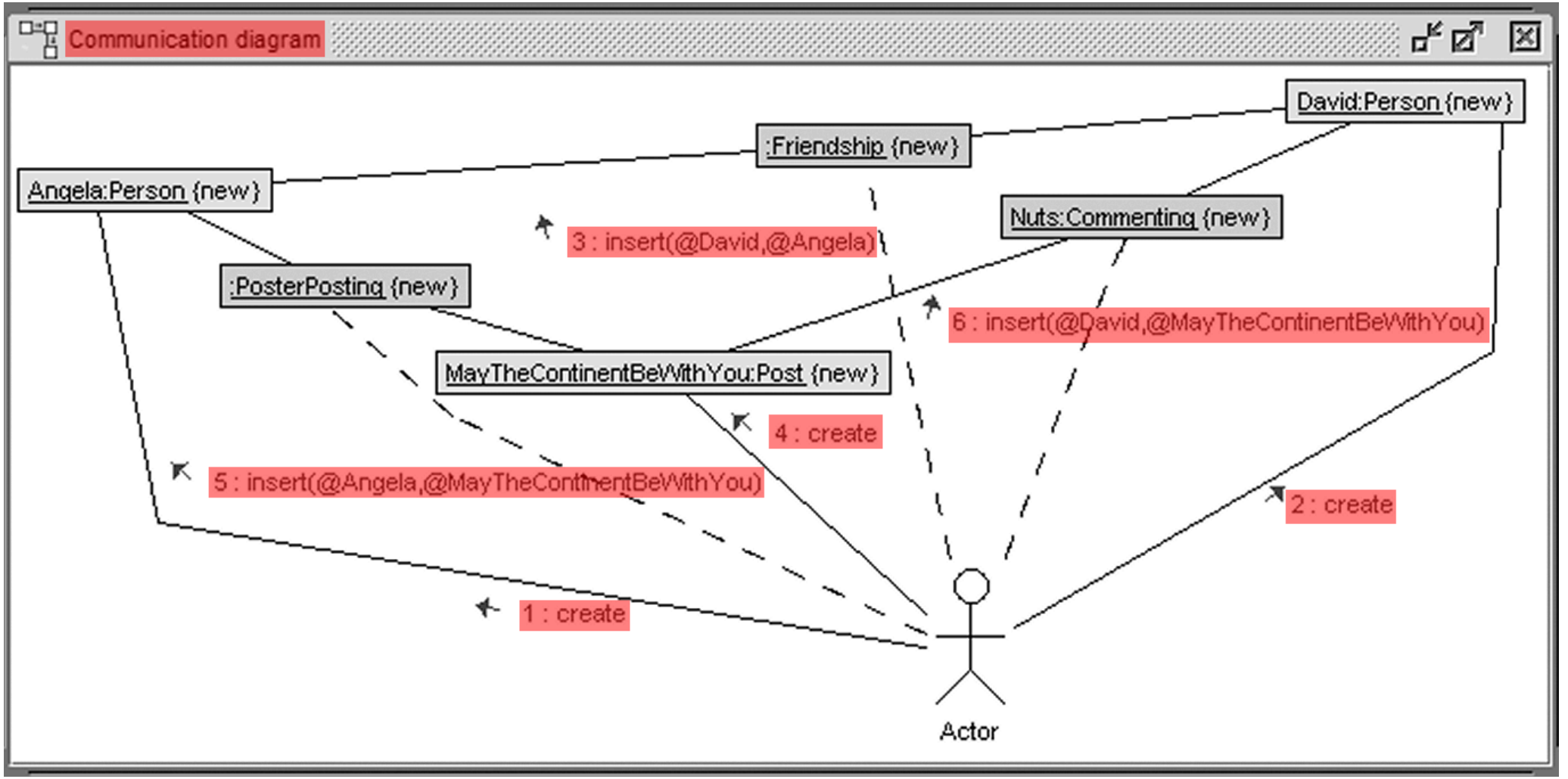
Model Validator Use Cases



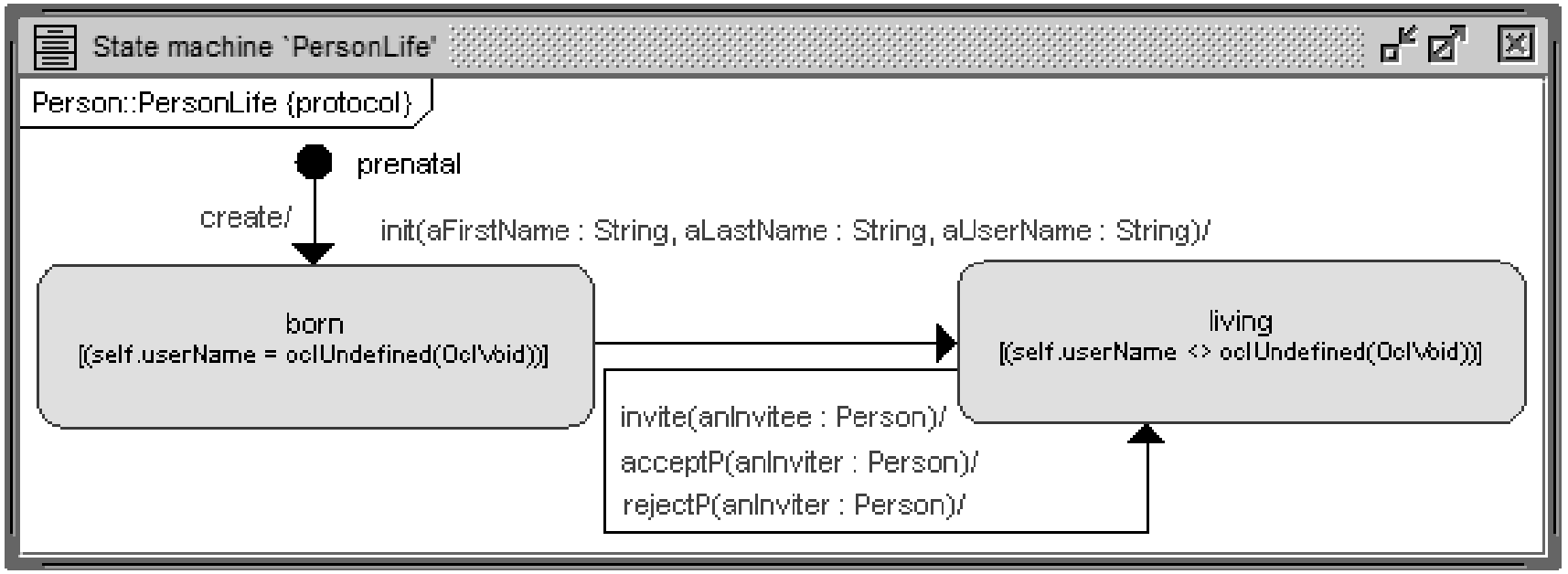
Sequence Diagram @ USE

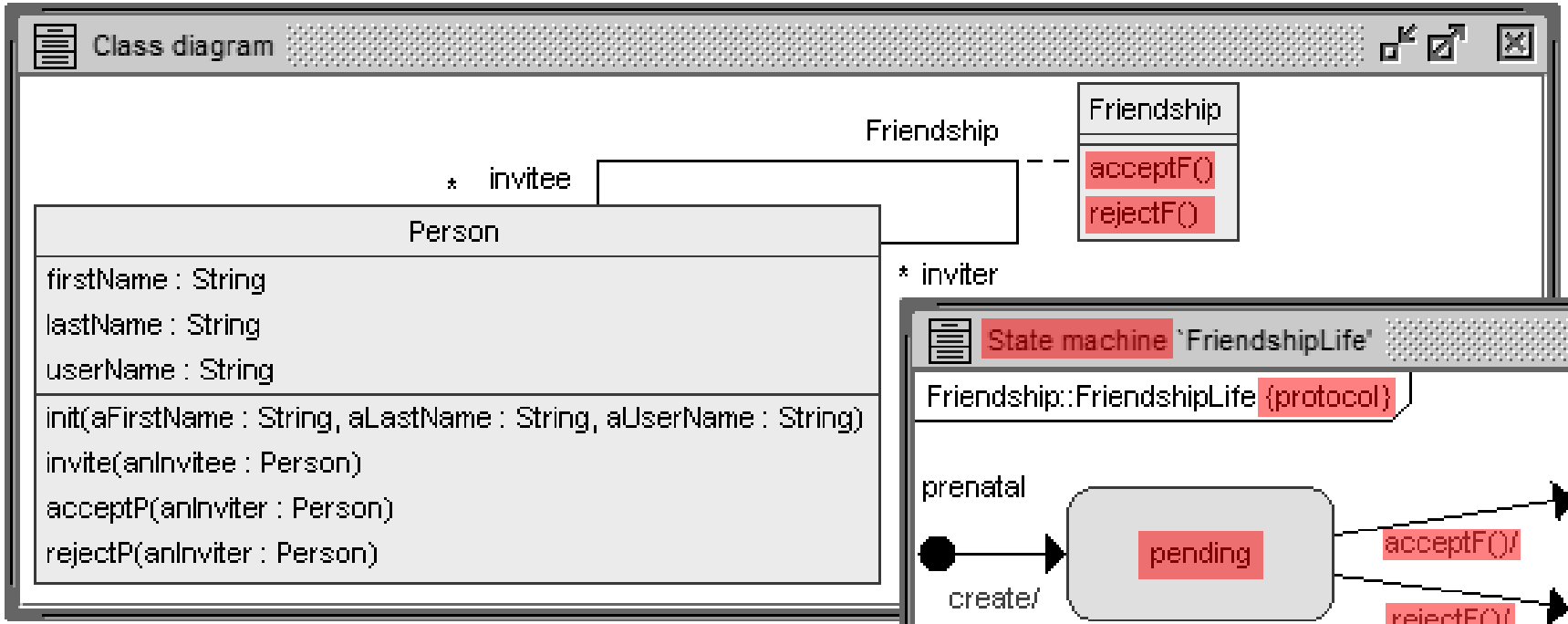


Communication Diagram @ USE



Protocol State Machine @ USE

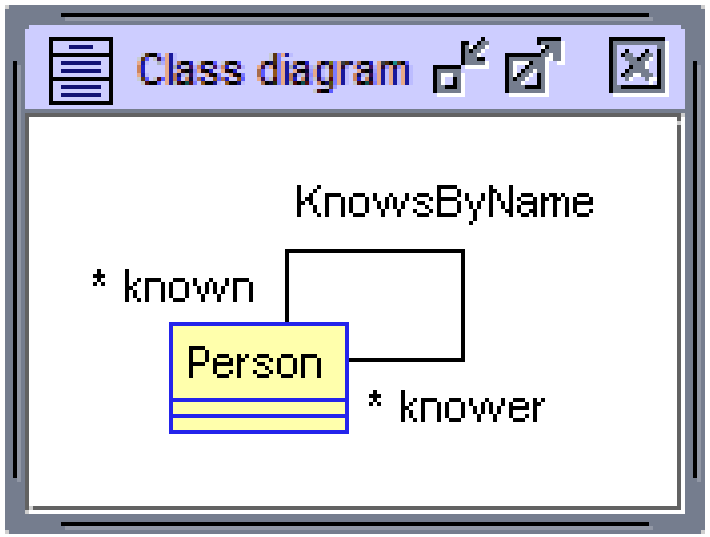


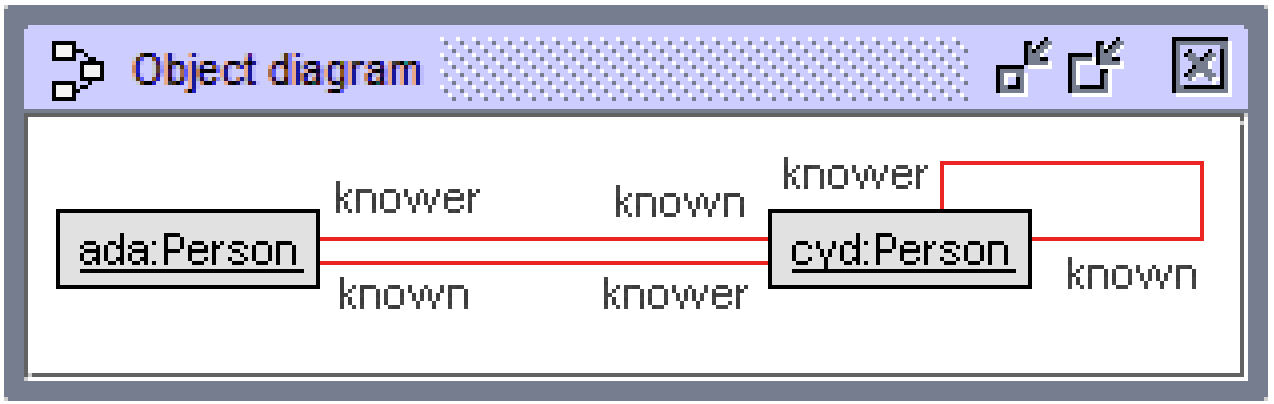


Outline

- View on modeling
- USE features
- Classifying terms
- Transformation models
- Filmstrip models

Multiple object diagram solutions employing one classifying term





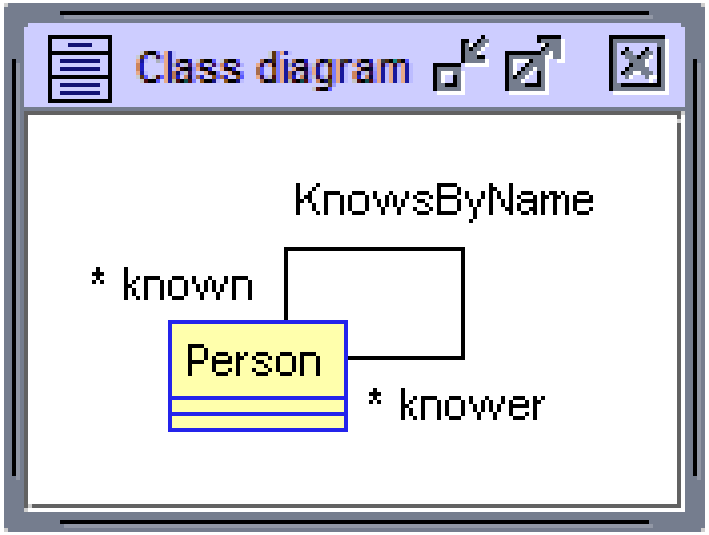
Configuration

- **Person_min = 1**
  Person_max = 3
  Person =
  Set{ada,bob,cyd}
- **KnowsByName_min = 0**
  KnowsByName_max = -1

Classifying term

- **Person.allInstances->size**

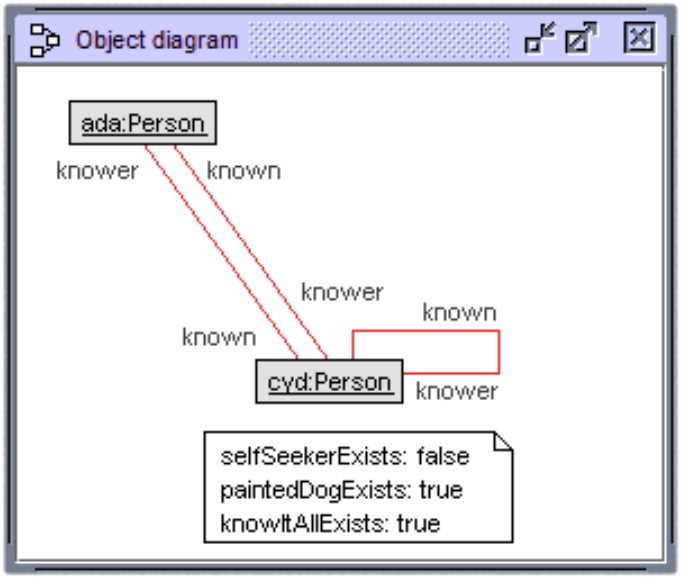
Multiple solutions with multiple classifying terms

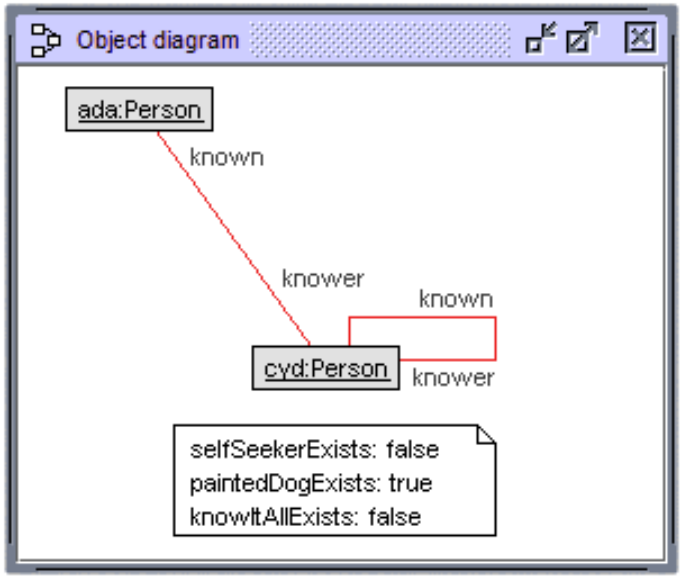


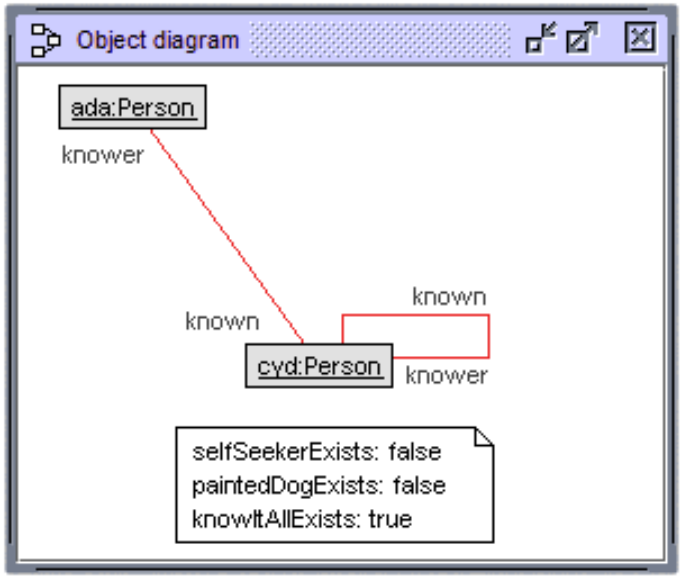
Classifying terms

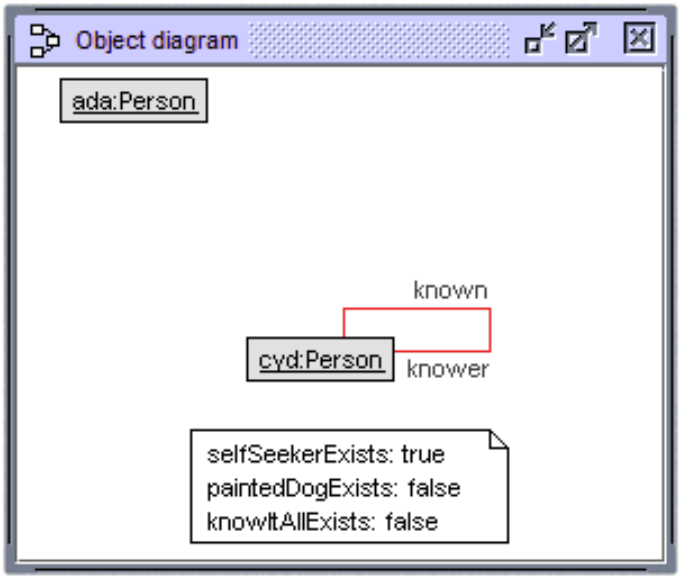
```
let selfSeekerExists= -- in German 'Egoist'
  Person.allInstances->exists(p | p.knower=Set{p} and p.known=Set{p})
  in selfSeekerExists

let knowItAllExists= -- in German 'Alleswisser'
  Person.allInstances->exists(p | p.knower=Person.allInstances)
  in knowItAllExists

let paintedDogExists= -- in German 'bunter Hund'
  Person.allInstances->exists(p | p.known=Person.allInstances)
  in paintedDogExists
```

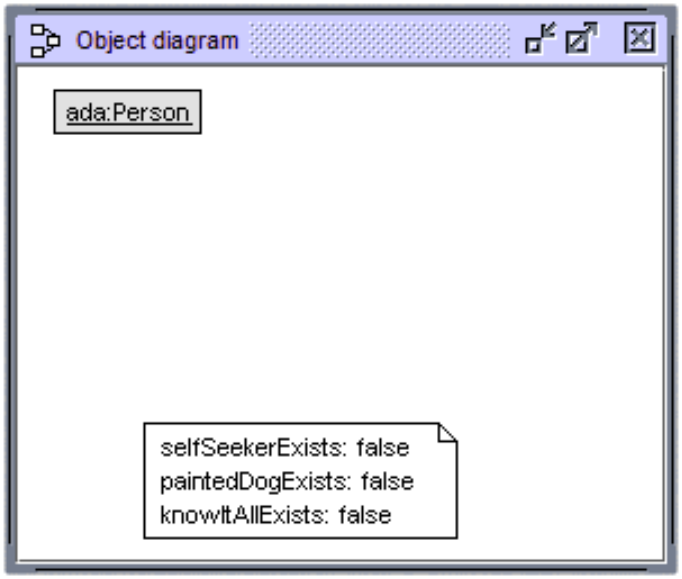
Multiple solutions with multiple classifying terms

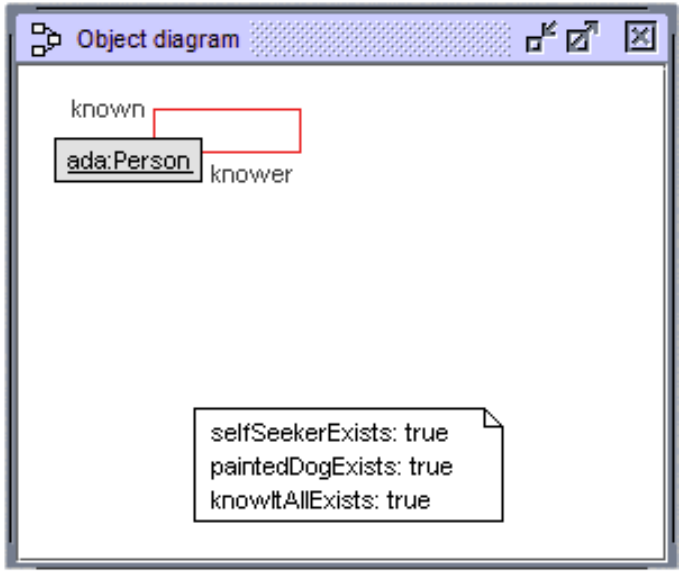












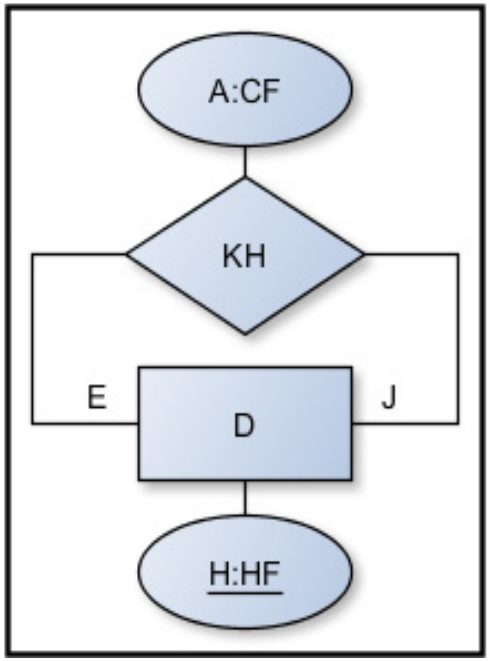
Outline

- View on modeling
- USE features
- Classifying terms
- Transformation models
- Filmstrip models

Transformation between ER schemas and relational DB schemas

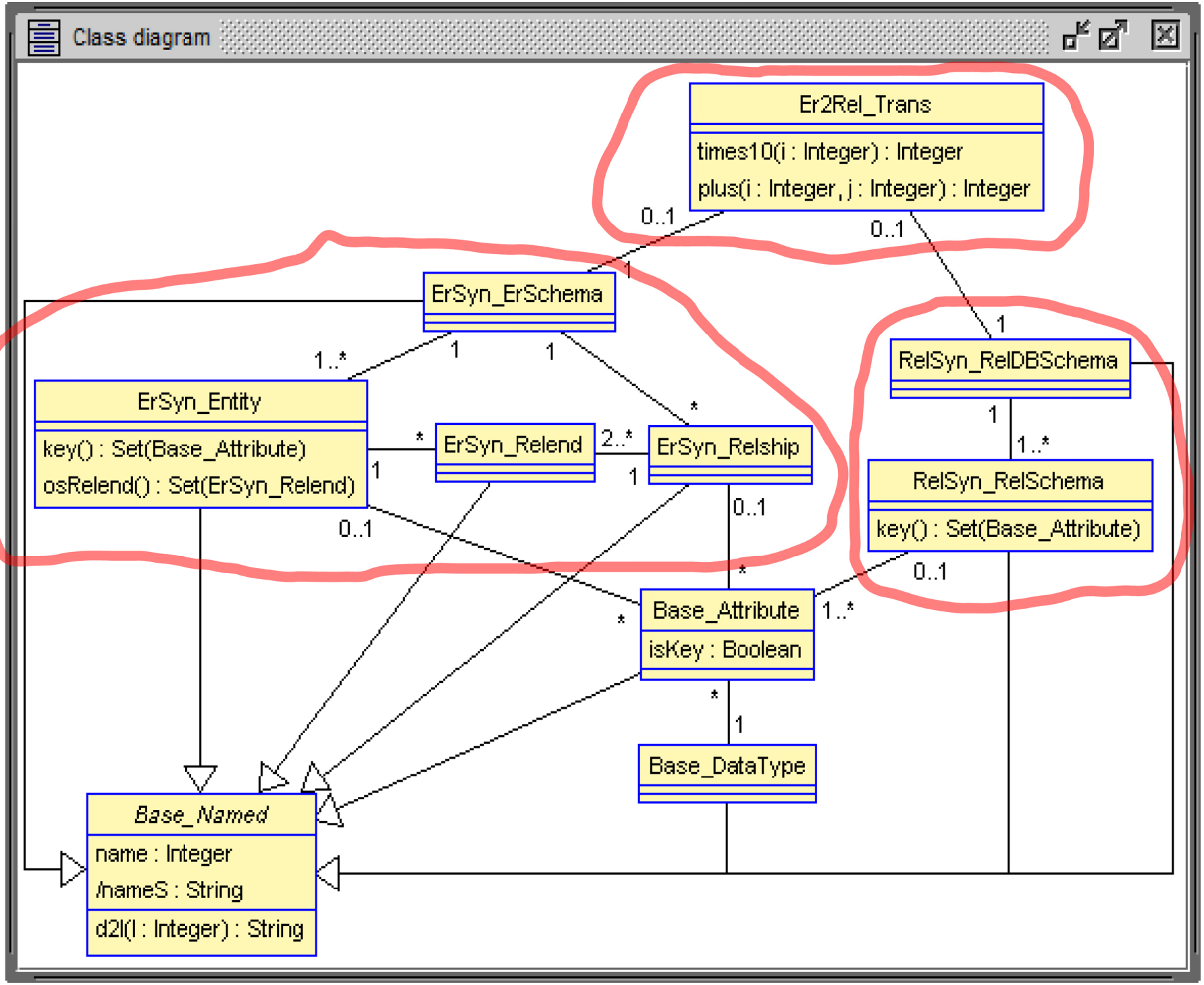
- Metamodel for Entity-Relationship (ER) and for relational datamodel
- Transformation model with single transformation class and invariants
- 23 OCL invariants, partly complex structured
- Verification task: Show transformation model consistency, i.e., automatically construct an example transformation that utilizes all concepts (classes and associations) from the metamodels

Constructed ER Diagram and Equivalent RelDB Schema

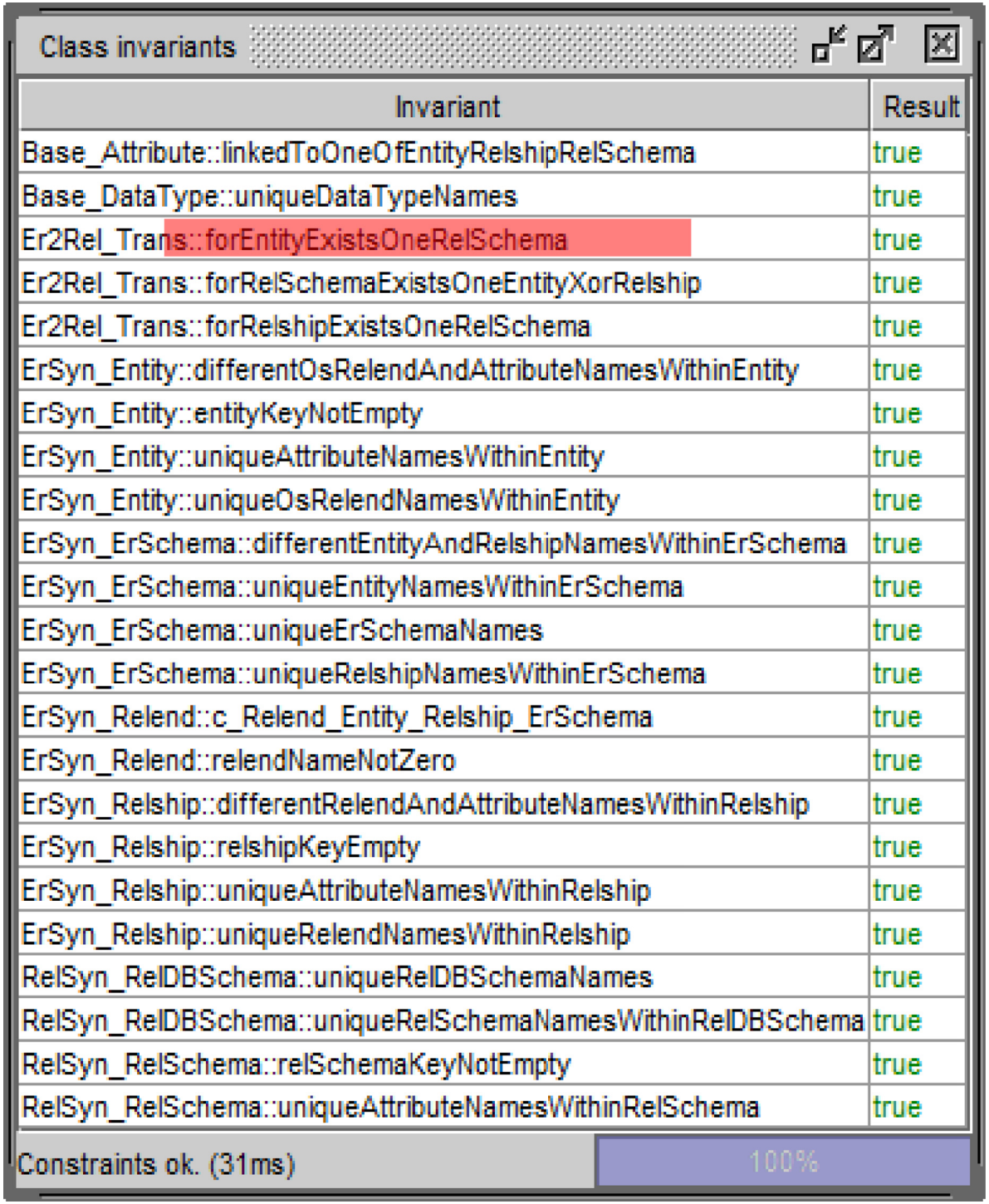



```
create table D(
  H HF primary key)
create table KH(
  EH HF,
  JH HF,
  A CF,
  primary key(EH,JH))
```

ER, RelDB and Transformation Metamodel



Metamodel Constraints



Non-trivial Example Constraint (6 Quantifier Nesting Levels)

```
context self:Er2Rel_Trans
  inv forRelSchemaExistsOneEntityXorRelship:
  self.relDBSchema.relSchema->forAll(rl |
    self.erSchema.entity->one(e |
      rl.name=e.name and
      rl.attribute->forAll(ra |
        e.attribute->one(ea |
          ra.name=ea.name and ea.dataType=ra.dataType and
          ra.isKey=ea.isKey)))
    xor
    self.erSchema.relship->one(rs |
      rl.name=rs.name and
      rl.attribute->forAll(ra |
        rs.relend->one(re |
          re.entity.key()->one(rek |
            ra.name=re.name.concat('_').concat(rek.name) and
            ra.dataType=rek.dataType and ra.isKey))
        xor
        rs.attribute->one(rsa |
          ra.name=rsa.name and ra.dataType=rsa.dataType and
          ra.isKey=false))))
```

Example Model Validator Configuration

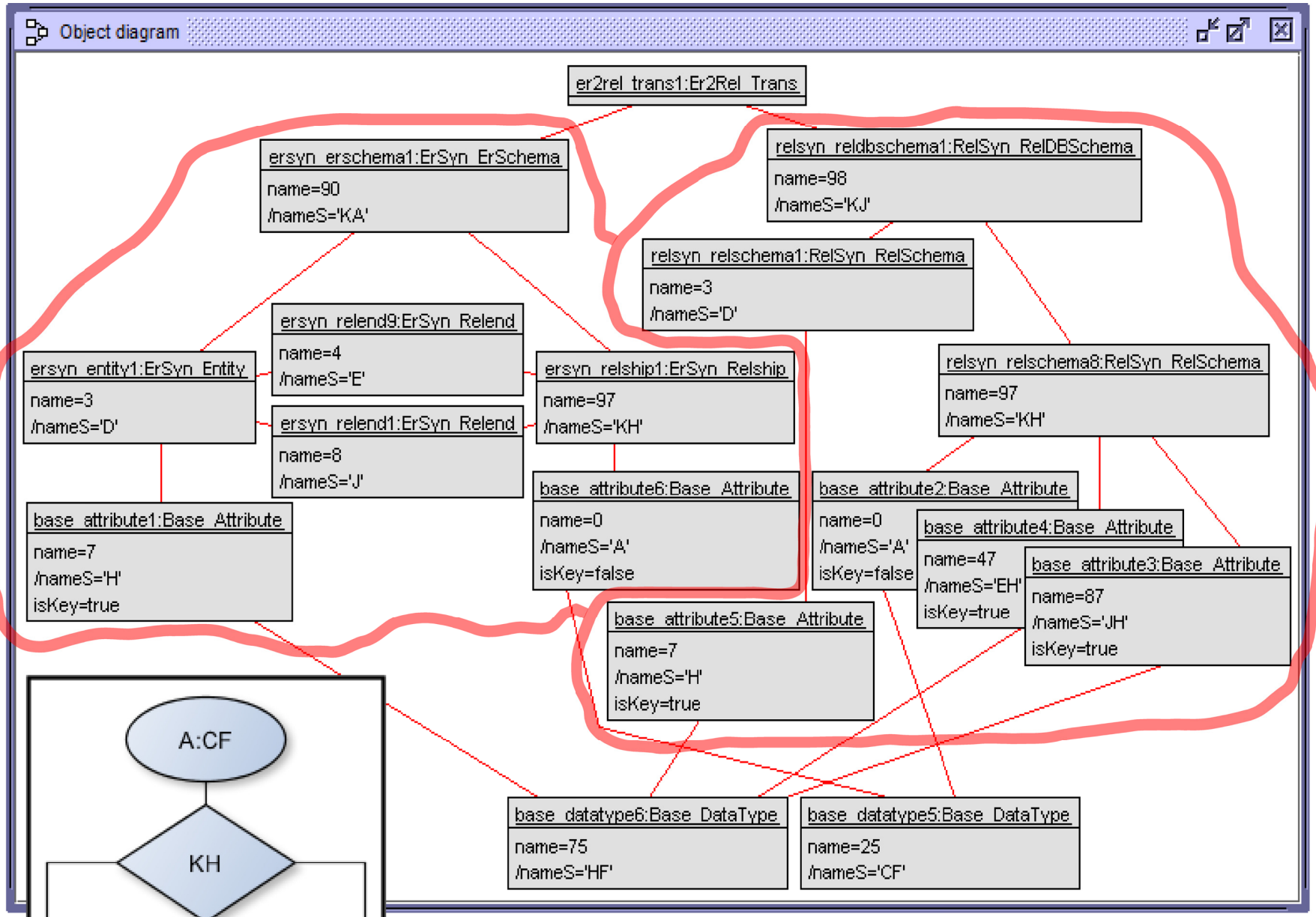
```
Er2Rel_Trans : 1..1

Er2Rel_OwnershipTransErSchema    : 1..1
Er2Rel_OwnershipTransRelDBSchema : 1..1
```

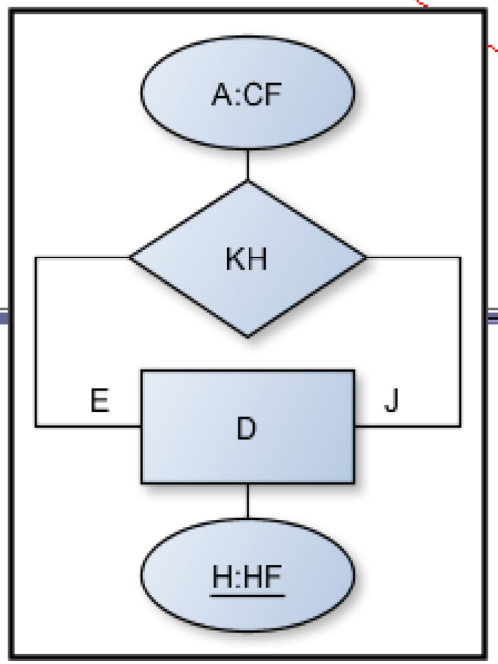
```
ErSyn_ErSchema : 1..1
ErSyn_Entity   : 1..9
ErSyn_Relship  : 1..9
ErSyn_Relend   : 1..9

ErSyn_OwnershipErSchemaEntity   : 1..9
ErSyn_OwnershipErSchemaRelship  : 1..9
ErSyn_OwnershipEntityAttribute  : 1..9
ErSyn_OwnershipRelshipAttribute : 1..9
ErSyn_OwnershipRelshipRelend    : 1..9
ErSyn_RelendTyping              : 1..9
```

```
RelSyn_RelDBSchema : 1..1
RelSyn_RelSchema   : 1..9

RelSyn_OwnershipRelDBSchemaRelSchema : 1..9
RelSyn_OwnershipRelSchemaAttribute   : 1..9
```

```
Base_Attribute       : 1..9
Base_DataType        : 1..9
Base_Named_name      : Set{0,1,2,3,4,5,6,7,8,9,10,11,..., 97,98,99}
Base_Attribute_isKey : Set{false,true}

Base_AttributeTyping : 1..9

Real      : 0..0
Real_step : 1
String    : 0..0
Integer   : 0..127
```

```
Class       black-on-white
Association black-on-light-grey
```



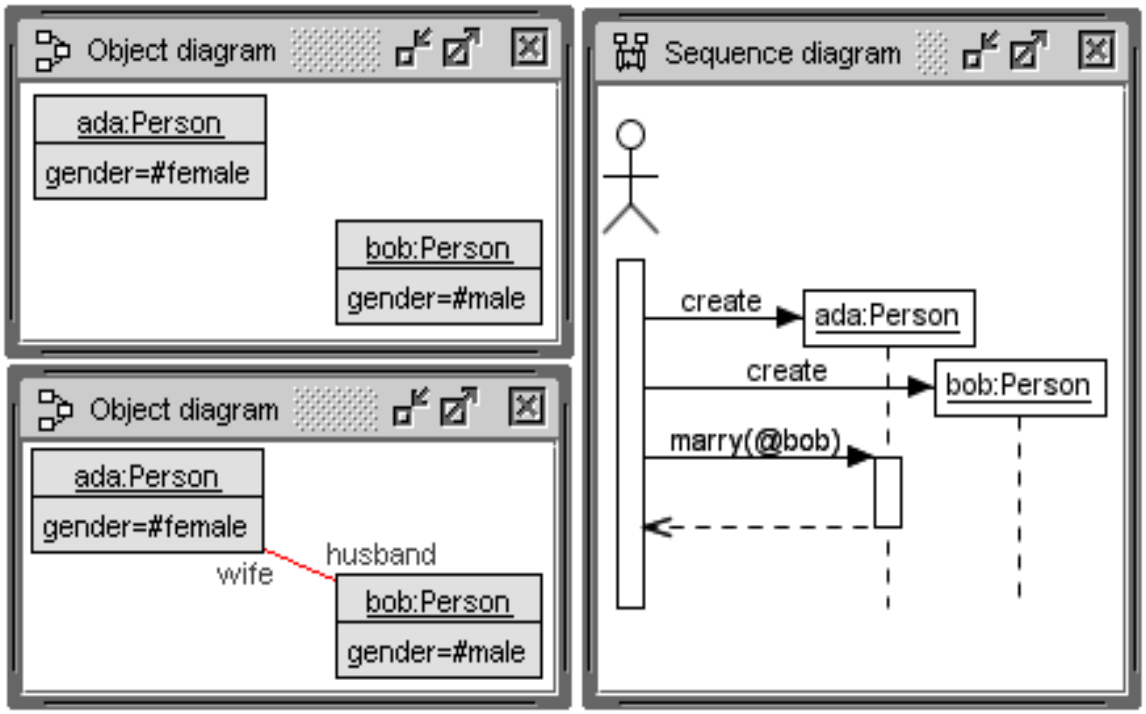


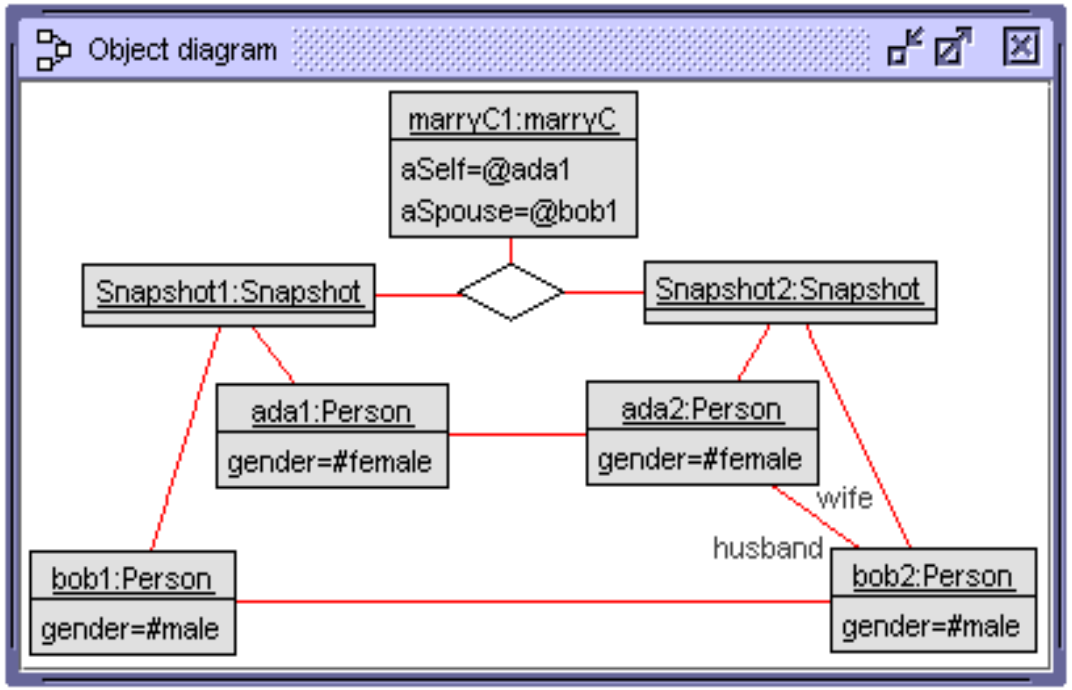

```
create table D(
  H HF primary key)
create table KH(
  EH HF,
  JH HF,
  A CF,
  primary key(EH,JH))
```

Outline

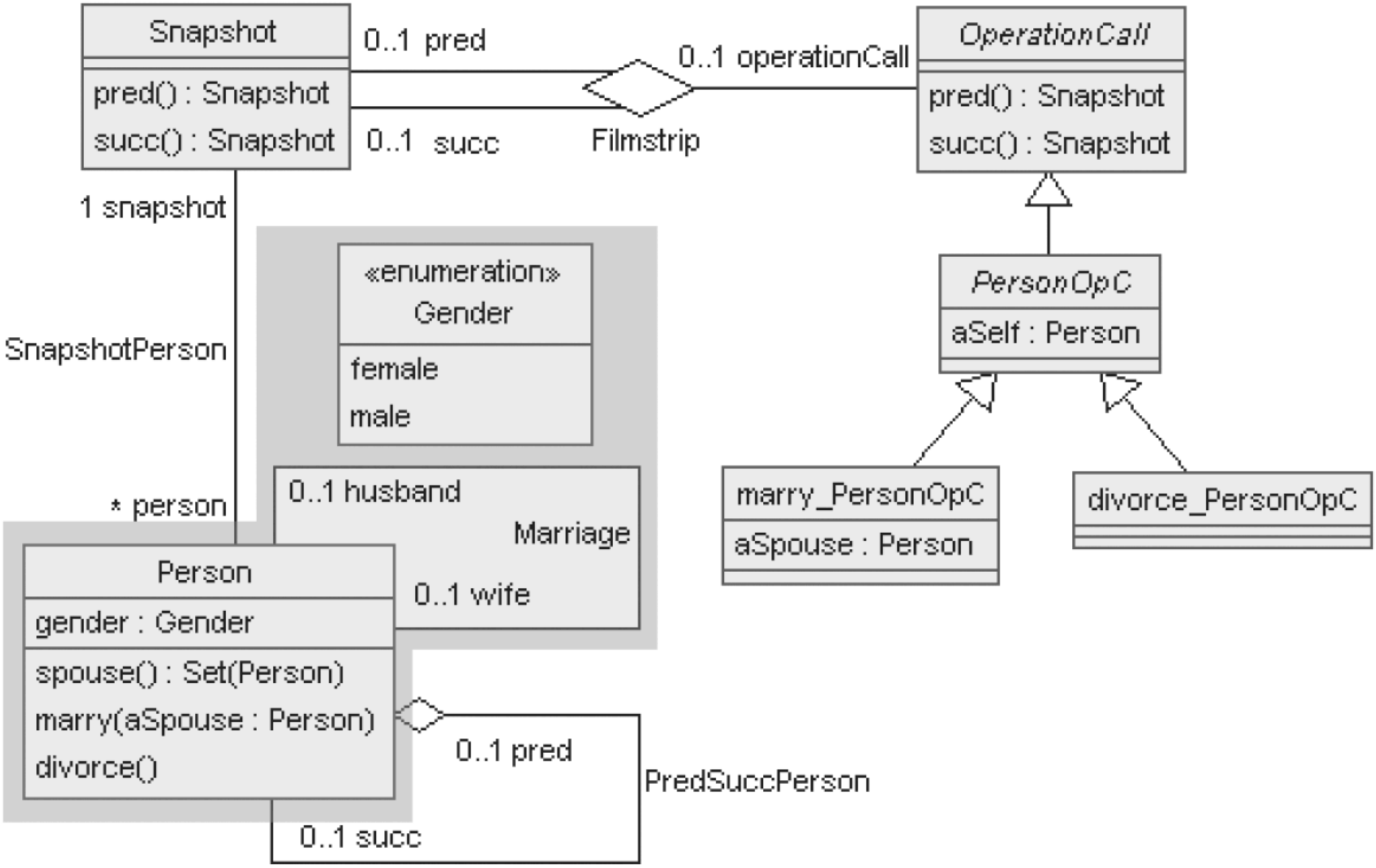
- View on modeling
- USE features
- Classifying terms
- Transformation models
- Filmstrip models

Filmstripping: Transforming pre- and postconditions into invariants





Filmstripping: Class model transformation USE plugin



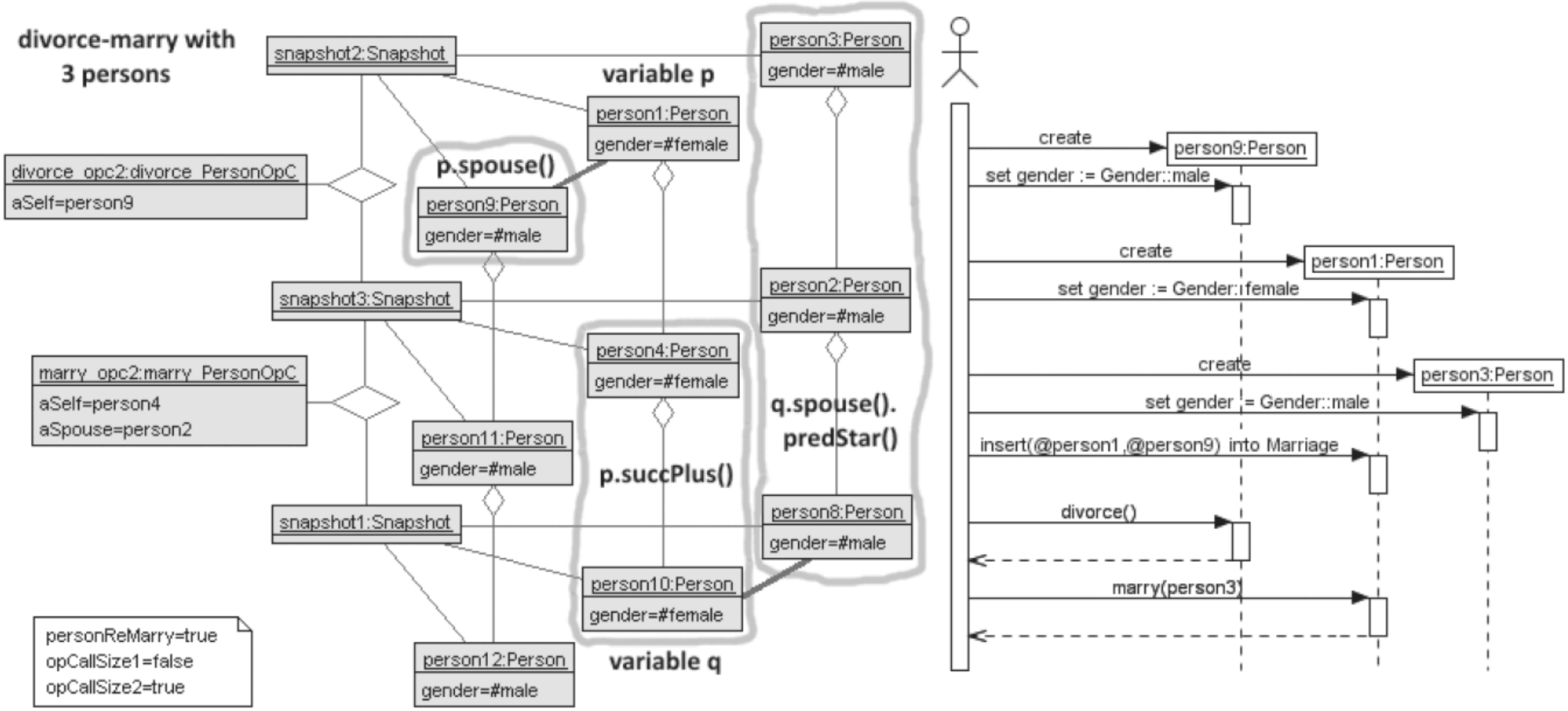
Filmstripping: Configuration and classifying terms

	ThreeSeqDias (TSD)
Snapshot min..max	2..3
Filmstrip min..max	1..2
marry_PersonOpC min..max	0..2
divorce_PersonOpC min..max	0..2
Person min..max	4..12
SnapshotPerson min..max	4..12
PredSuccPerson min..max	2..8
Marriage min..max	0..*

```
TSD1 Person.allInstances->exists(p | p.spouse()->size=1 and
       p.succPlus()->exists(q | q.spouse()->size=1 and
         q.spouse().predStar()->excludesAll(p.spouse())))
TSD2 OperationCall.allInstances->size=1
TSD3 OperationCall.allInstances->size=2
```

TSD1: "Person p is married and later married again, but to a different person."

Filmstrip object diagram VS Application sequence diagram



Summary

- Software development concentrating on models as first class citizens
- Successfully applied approach in larger German project: XGenerator @ Deutschland online
- UML (Class and Statechart diagrams) in combination with OCL (not mentioned yet the *darker side of the force*, namely class diagram features like subsets, redefines, union, association inheritance and association class inheritance)
- SOIL (Simple Ocl-like Imperative Language) for operation realization together model unit tests (Object and Sequence diagrams)
- Employ model validator for searching object diagram spaces
- Validation and verification
- Aim of modeling: Obtain trustworthy models

Perspective

- Support for model transformation: From descriptive transformations to prescriptive transformations; from platform-indepedent ones to platform-dependent ones; testing of transformations wrt contracts
- Technical extensions
  - statecharts features (adding change events and nested states)
  - synchronisation of seq. and com. diagram features
  - multilevel classifying terms
  - coverage of non-set collections in model validator
- Better support for filmstrip models and refinement models
- OCL extensions
  - Temporal logic for dynamic properties!
  - Deontic logic for obligations and permissions?
- Monitoring running applications (JVM, CRL, ...) with models

Thanks for your attention!