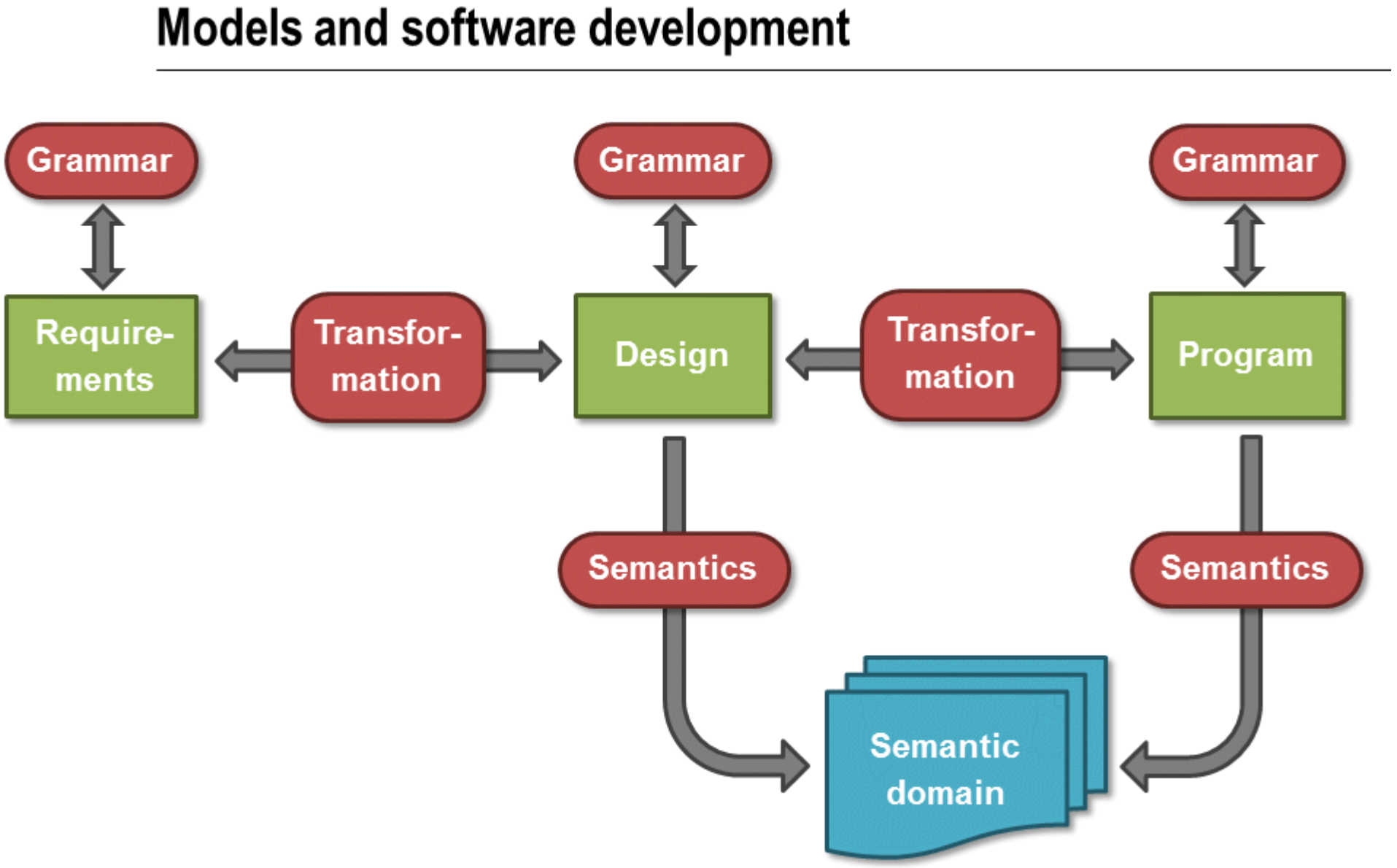
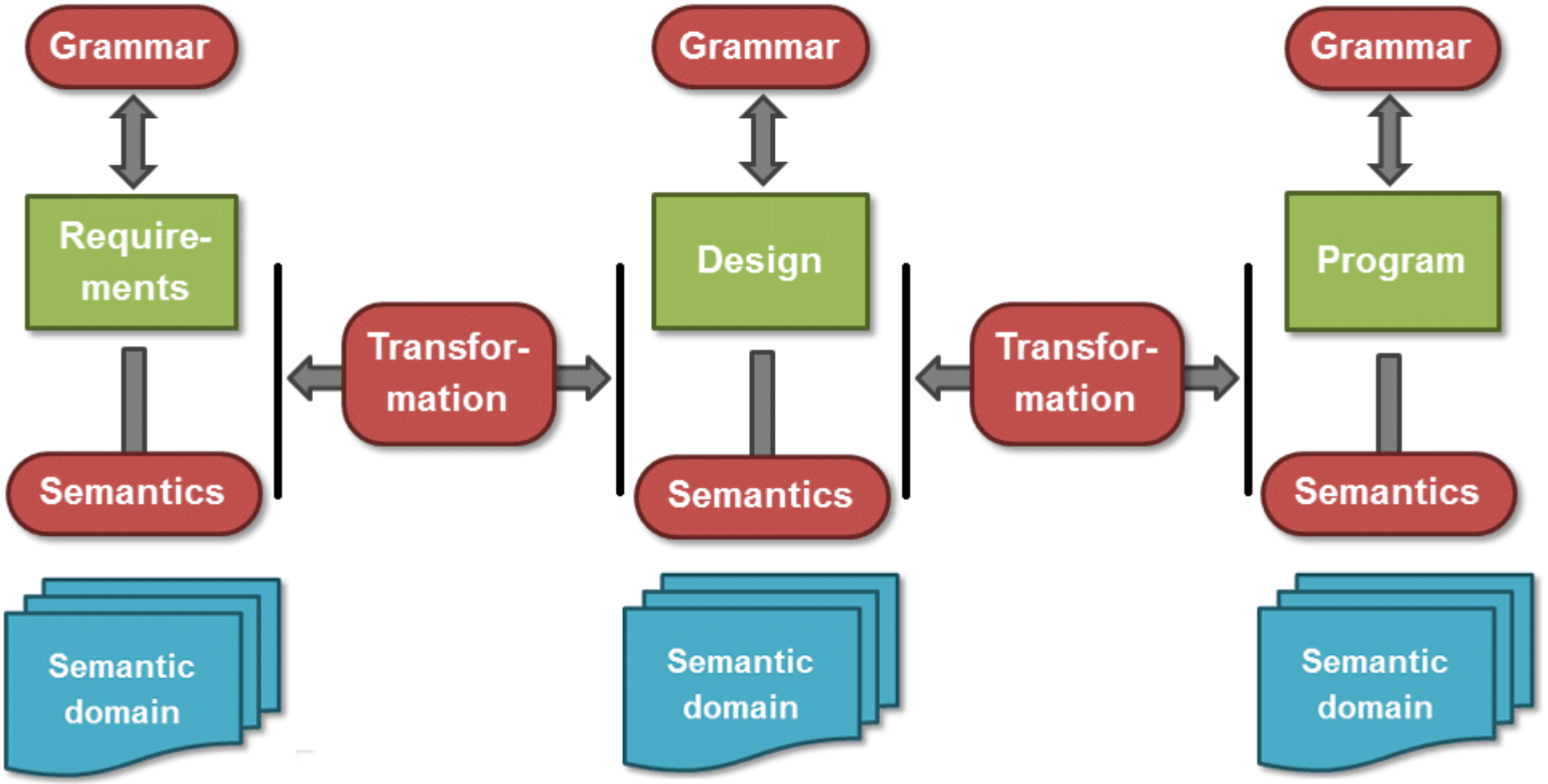
Modeling Language Transformations with USE

Martin Gogolla
University of Bremen, Germany
Database Systems Group

Overview slide from Arend Rensink



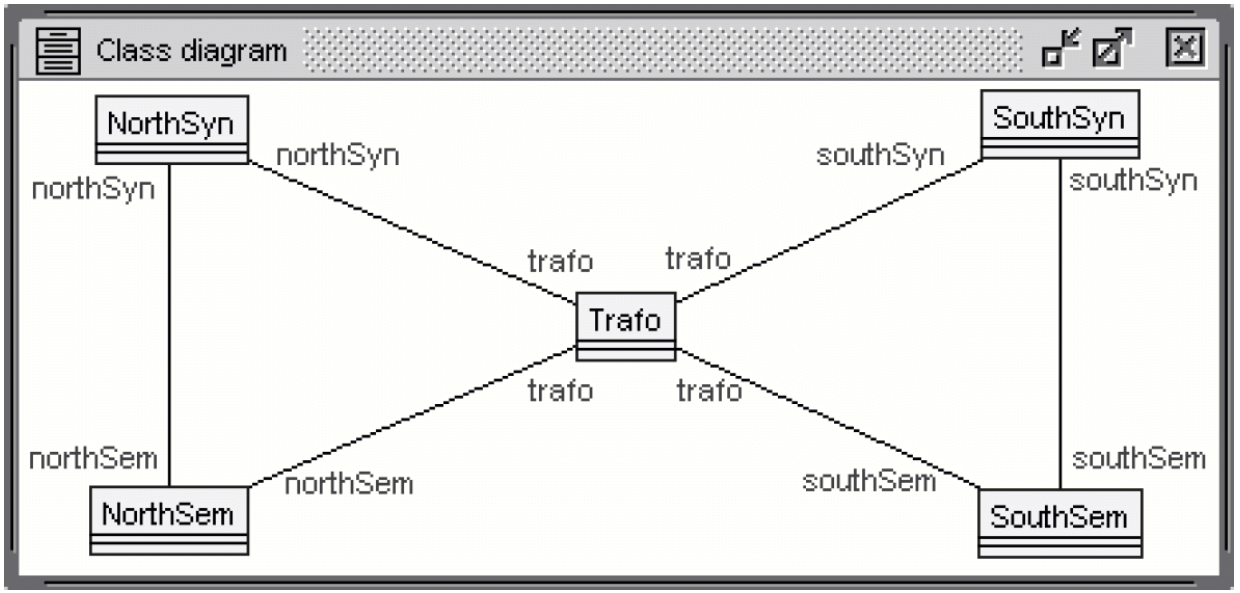
Models and software development



Tool USE (UML-based Specification Environment)

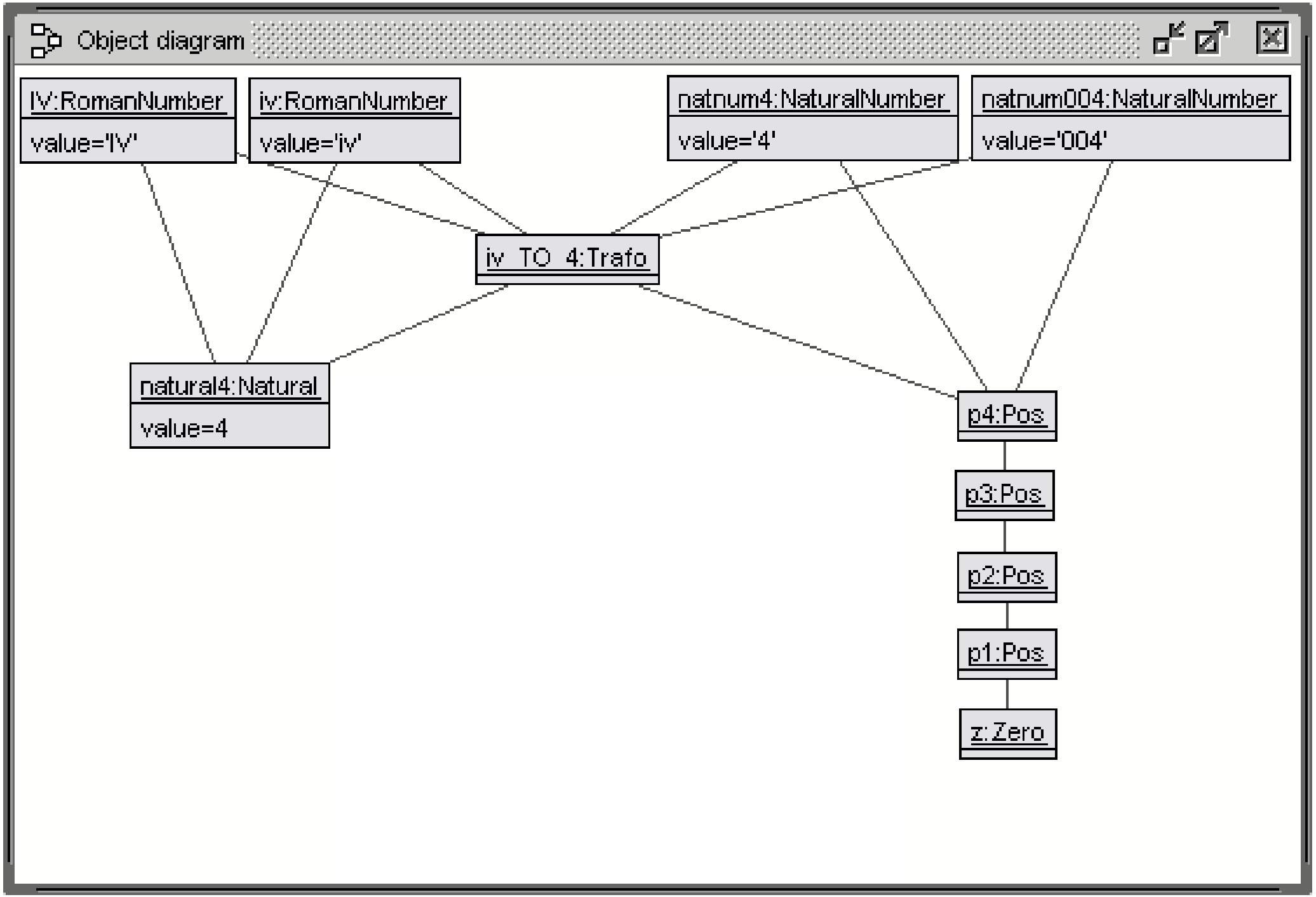
- Validation and verification tool for UML and OCL models
- Supports UML class, object, sequence, and statechart diagrams
- OCL support for (a) class invariants, (b) operation pre- and postconditions, (c) definition of query operations, (d) derivations for attributes and association ends, (e) state invariants, and (f) transition guards and transition postconditions
- Imperative language for implementing non-query operations on the model level: SOIL (Simple Ocl-like Imperative Language)
- Model validation by executing test scenarios
- Automatic generation of test scenarios in form of object diagrams through a model validator based on a translation of UML and OCL into relational logic (realized in Kodkod/Alloy)
- Checking of model properties like model consistency, model minimality or model state reachability

Transformation model SPECIFYING Model transformations

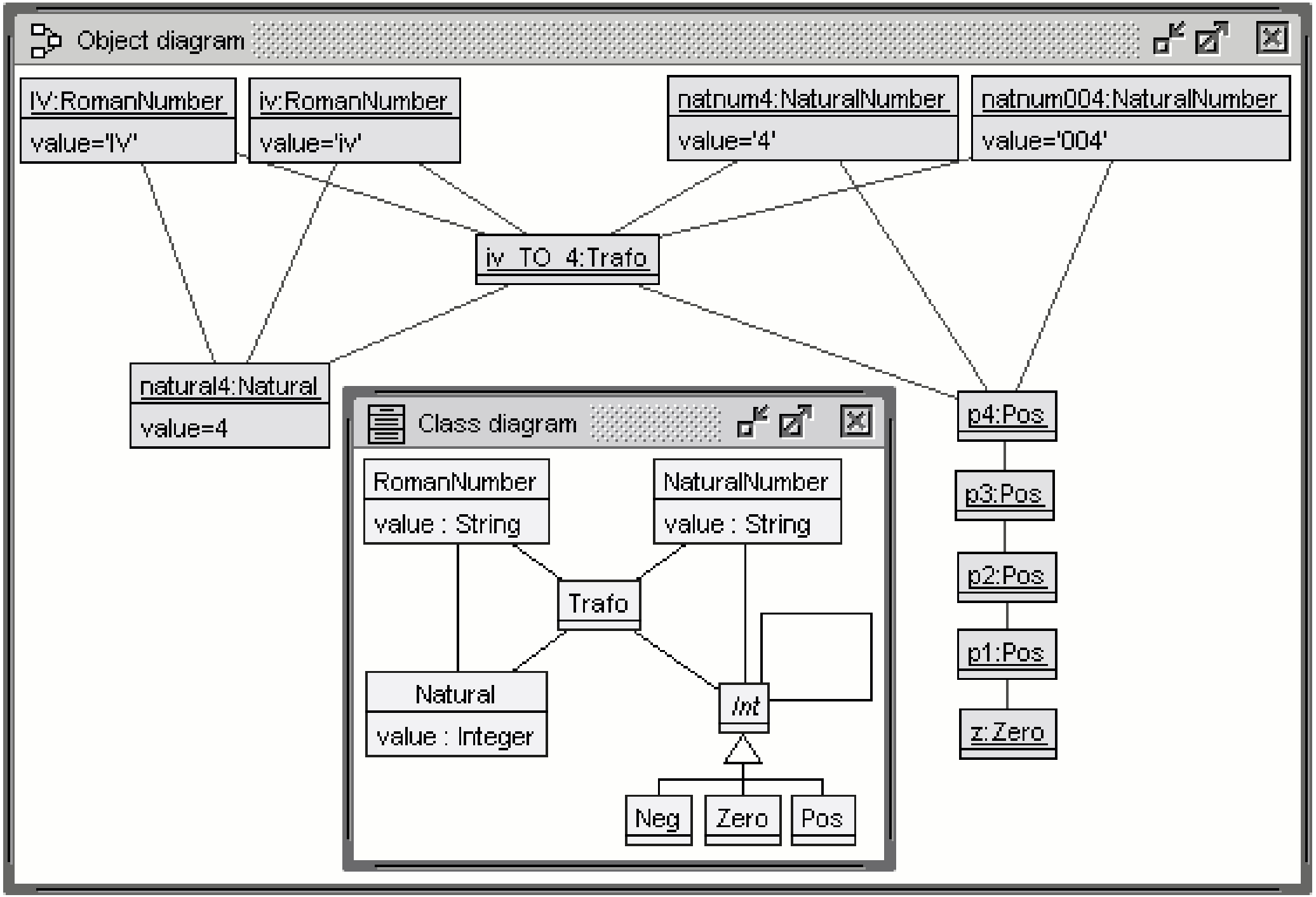


- Language described by metamodel (class model plus OCL constraints) divided into syntax and semantics (evaluation)
- Transformation needs two languages: North and South
- Direction-neutral transformation model Trafo
- Transformation properties (e.g., equivalence, embedding) stated as class invariants in Trafo
- Model transformation realized as imperative operations
- (a) Languages to be transformed, (b) transformation and (c) properties are formulated in a uniform way as metamodels

Example Transformation Model: Roman numbers



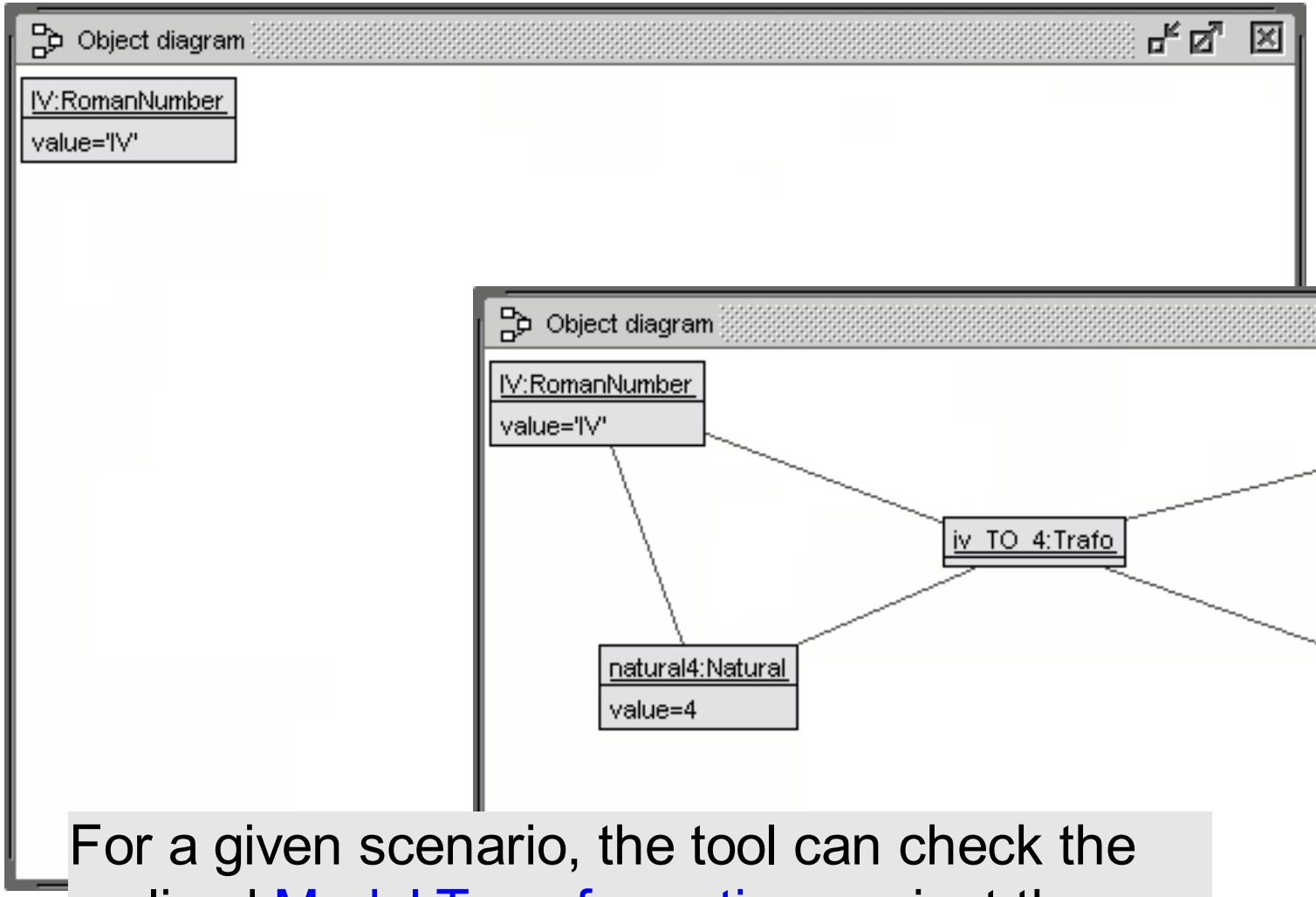
Example Transformation Model: Roman numbers



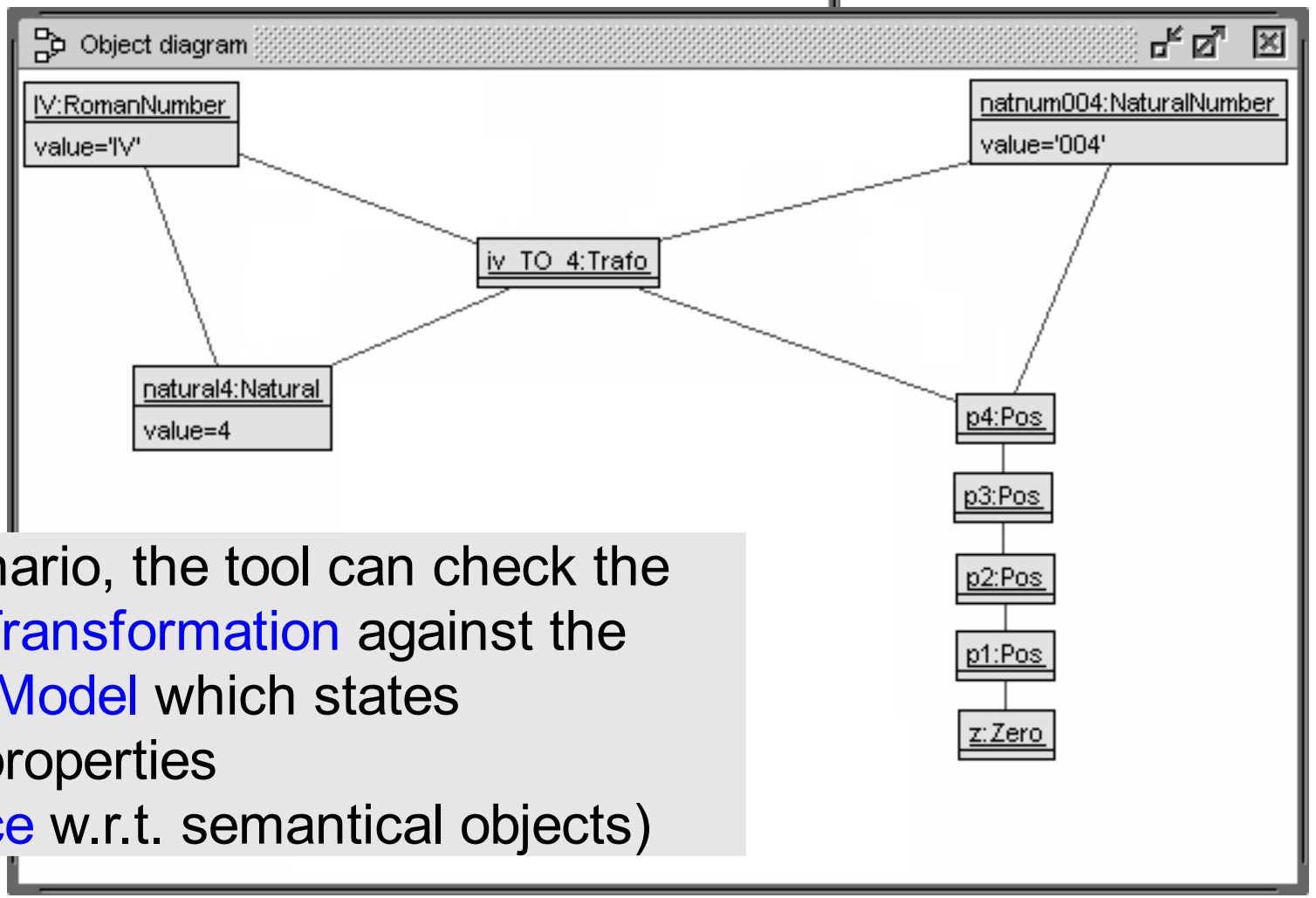
Employing SOIL for Implementing a Model Transformation

```
RomanNumber::RN2NN()
begin
-- IN: RomanNumber object self representing a correct Roman number
-- OUT: Natural, Trafo, NaturalNumber, Int objects with proper links
declare n:Natural, t:Trafo, nn:NaturalNumber, i:Int, j:Int, s:Integer;
n:=new Natural; n.value:=self.value();
insert (self,n) into RomanNumber_Natural;
t:=new Trafo;
insert (t,self) into Trafo_RomanNumber;
insert (t,n) into Trafo_Natural;
nn:=new NaturalNumber; nn.value:='00'.concat(n.value.toString());
insert (t,nn) into Trafo_NaturalNumber;
i:= new Pos;
insert (t,i) into Trafo_Int;
insert (nn,i) into NaturalNumber_Int;
for s in Sequence{1..n.value-1} do
  j:=new Pos; insert (j,i) into PredSucc; i:=j;
end;
j:=new Zero; insert (j,i) into PredSucc;
end
```

Employing SOIL for Implementing Transformation Operations




```
IV.RN2NN()
```

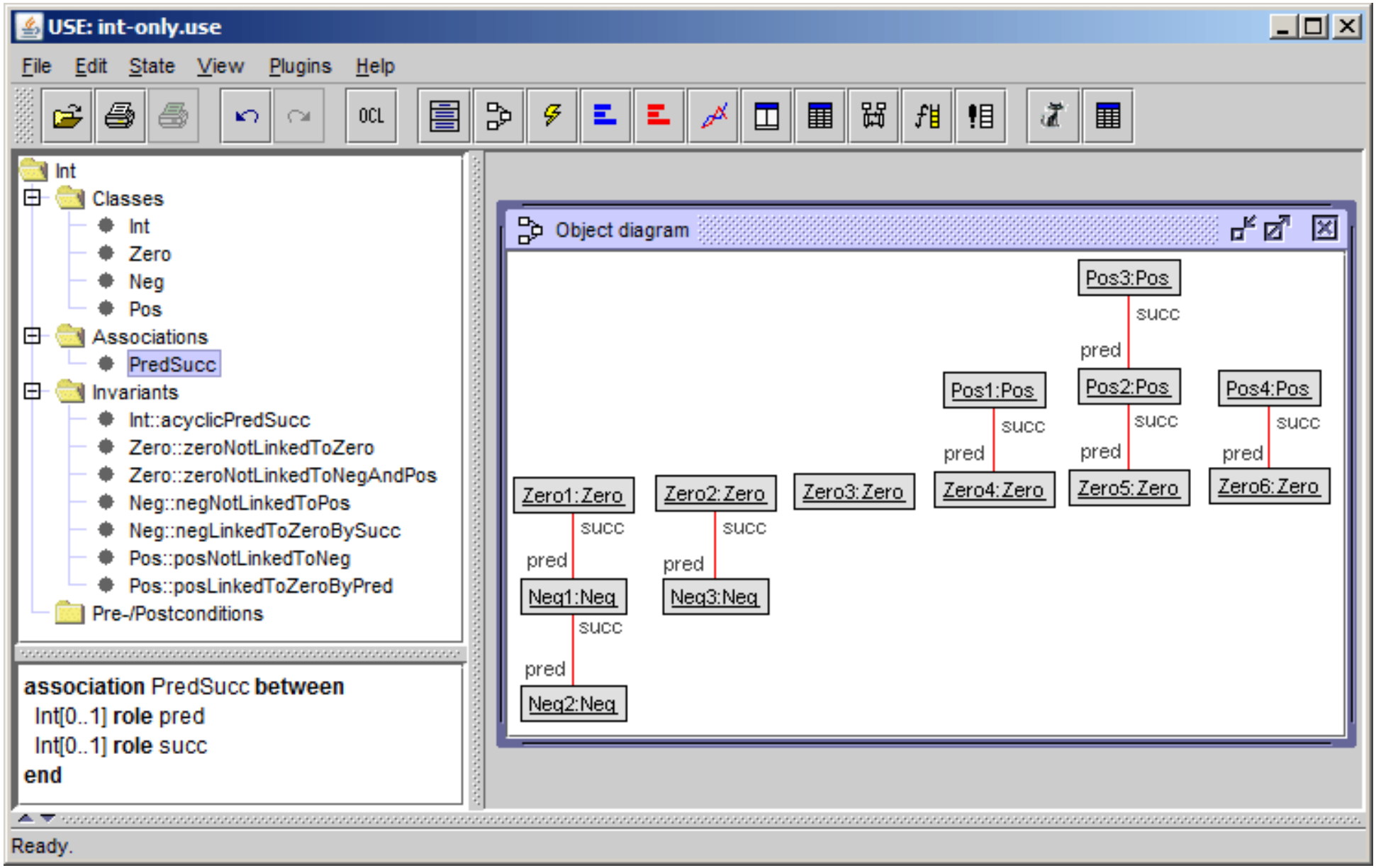


For a given scenario, the tool can check the realized Model Transformation against the Transformation Model which states transformation properties
(e.g., equivalence w.r.t. semantical objects)

Employing the Model Validator for Checking Model Properties

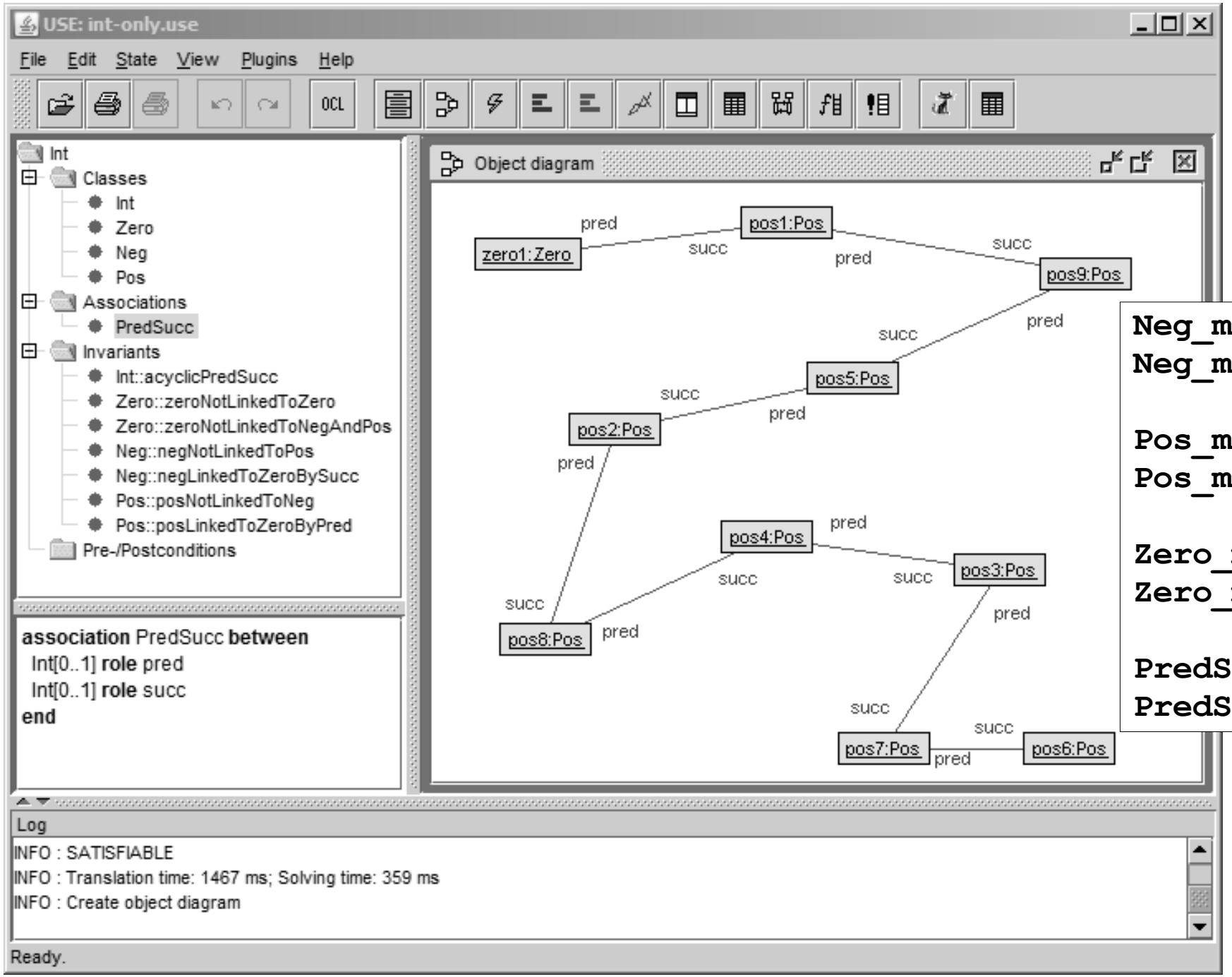
- Model validator is able to automatically construct test scenarios (object diagrams) from a finite search space which has to be configured by the developer
- Existence of object diagram can prove consistency of the transformation model (and other properties)
- Suspected logical consequence of a model may be added in negated form to the model
- If no object diagram is found, the suspected consequence can be considered to be a valid consequence of the model (this is true at least in the considered finite search space)

Employing the Model Validator for Checking Model Properties



Representation of integers in $pred^n(zero)$ or $succ^n(zero)$ normalform

Employing the Model Validator for Checking Consistency



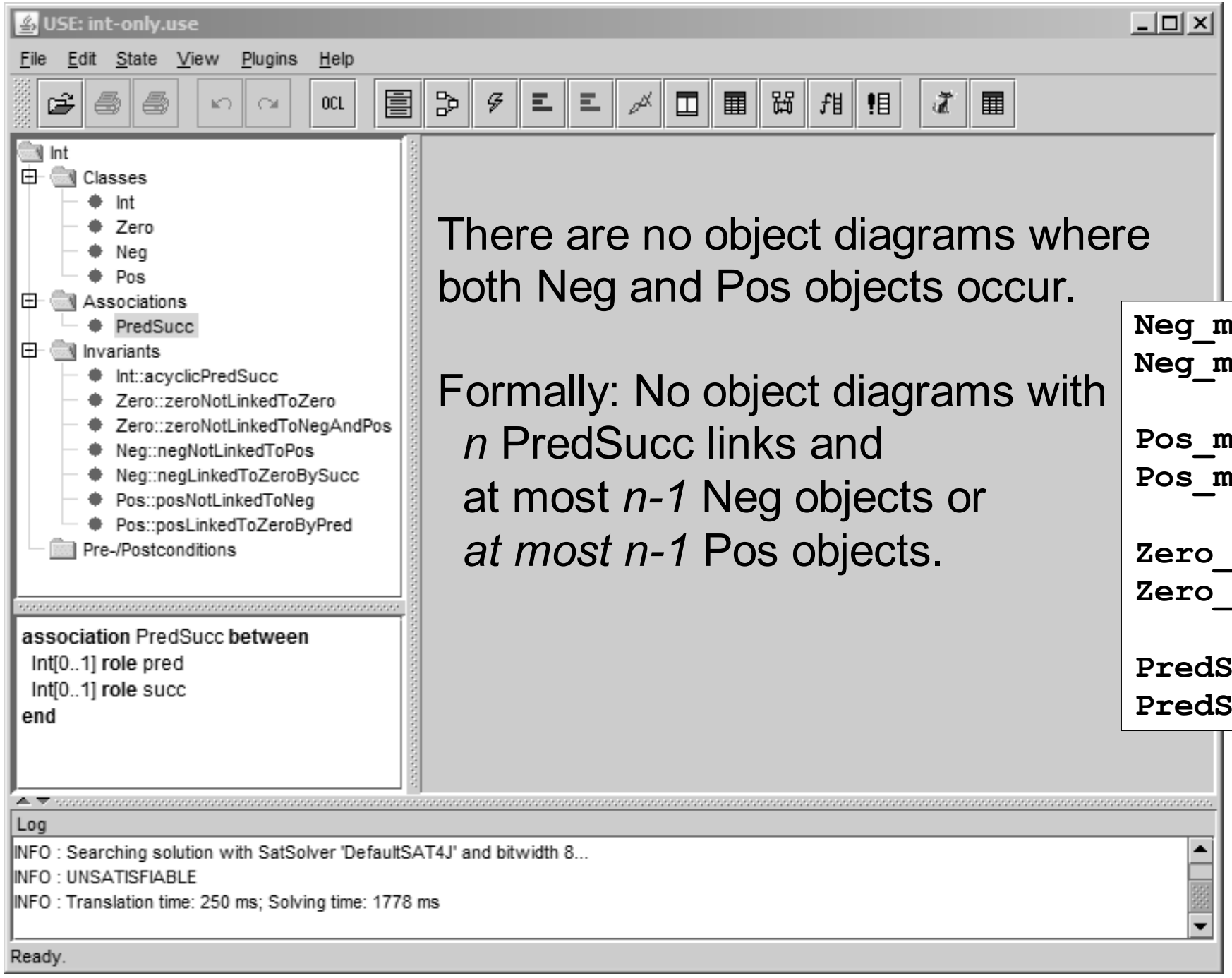

```
Neg_min = 0
Neg_max = 9

Pos_min = 0
Pos_max = 9

Zero_min = 1
Zero_max = 1

PredSucc_min = 9
PredSucc_max = 9
```

Employing the Model Validator for Checking Consistency



Terms North and South language are ok, but in a Banff context ...

Terms North and South language are ok, but in a Banff context ...

... Grizzly and Wolf language are more appropriate.

Thanks for your attention!

```
-- synob1.semantics<>synob2.semantics IMPLIES
--   synob1.trafo.syntax<>synob2.trafo.syntax

context rn1,rn2:RomanNumber inv embeddingOnSyntaxLevel:
    rn1.natural<>rn2.natural implies
    rn1.trafo.naturalNumber<>rn2.trafo.naturalNumber

Possible: transformation testing with scenarios
```