Experimenting with Multi-Level Models in a Two-Level Modeling Tool

martin gogolla
University of Bremen

Motivation

- talk presents two proposals for handling different metamodel levels in a uniform way

- in technical terms: represent different metamodel levels in ONE model, i.e. one class diagram including OCL constraints

- establish the connection between levels with
  + associations and generalizations
  + special OCL(?) operations

- in first approach, instanceOf relationship (usually between metamodel levels) becomes a simple association with precise meaning

- advantage: uniform employment of OCL
  - within each metamodel level,
  - for restricting the connection between the metamodel levels, and
  - for navigation between the metamodel levels

Structure of the talk

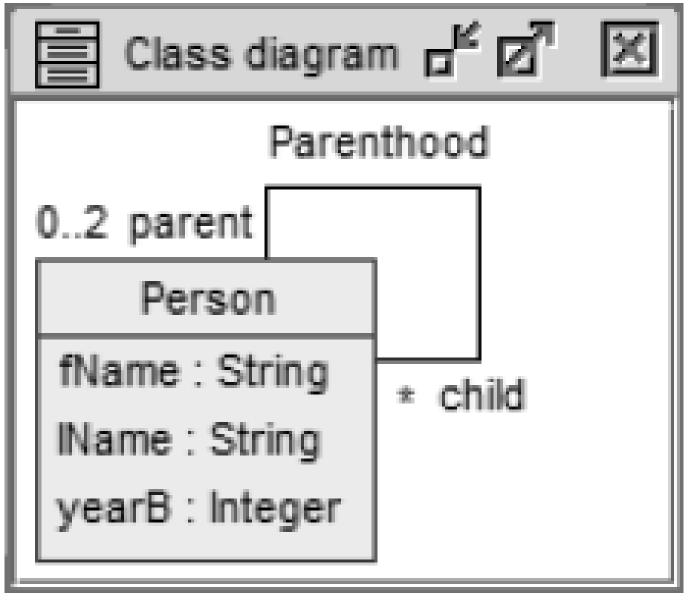
- Our context: USE (Uml-based Specification Environment)
- First approach: Metamodel level connection with associations and generalizations
- Second approach: Metamodel level connection with special OCL(?) operations
- A touch of related work
- Conclusion

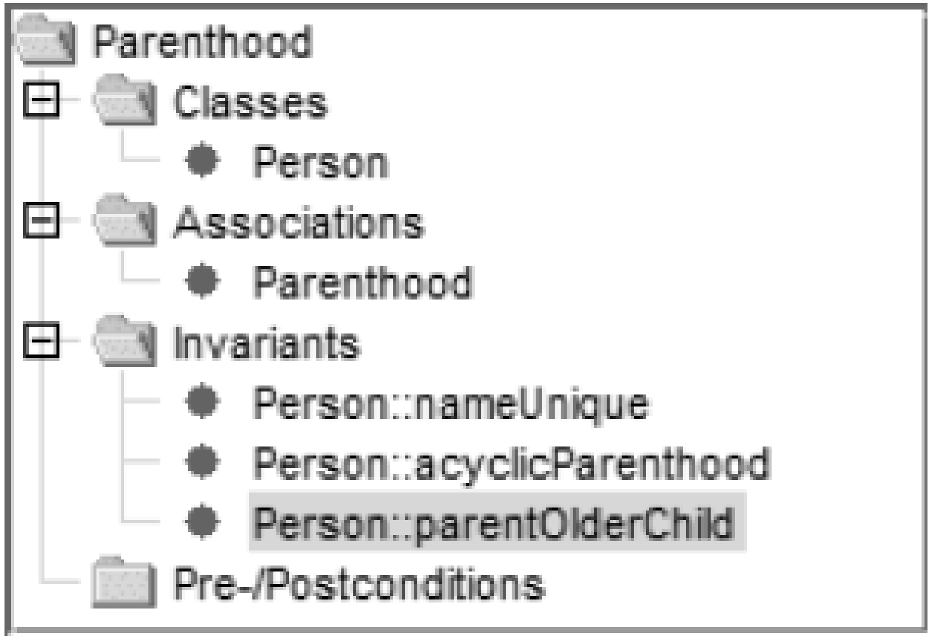
Structure of the talk

- **Our context: USE (Uml-based Specification Environment)**
- **First approach: Metamodel level connection with associations and generalizations**
- **Second approach: Metamodel level connection with special OCL(?) operations**
- **A touch of related work**
- **Conclusion**

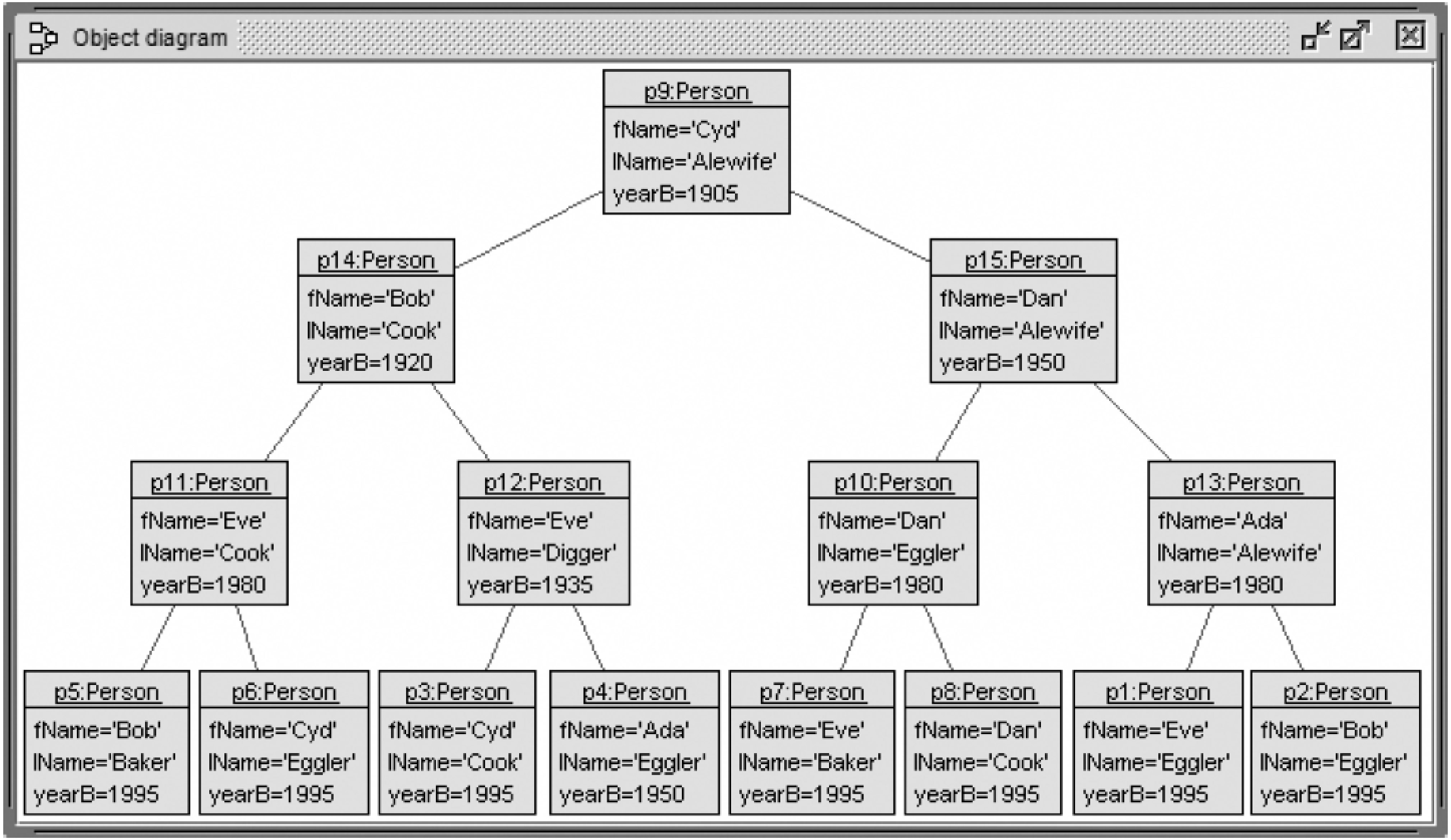
Tool USE (UML-based Specification Environment)

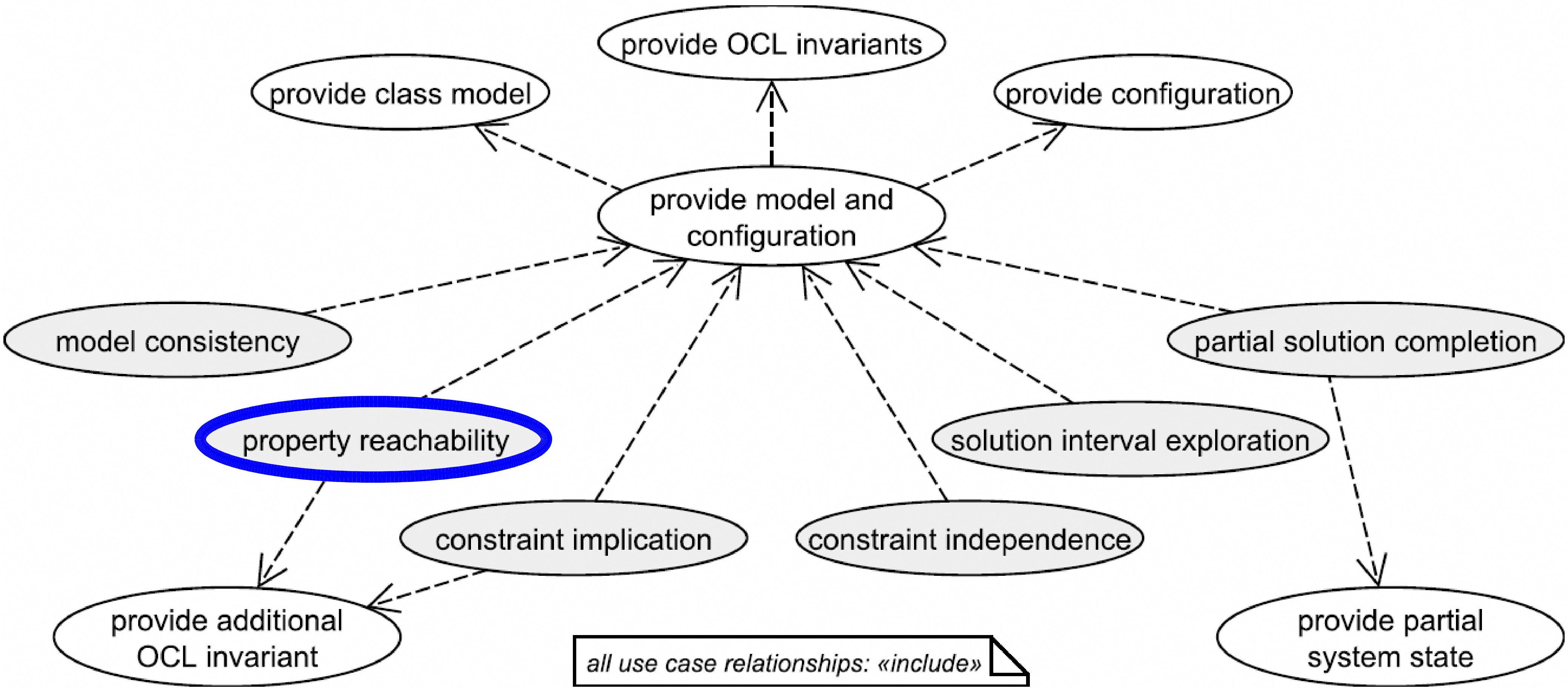
- google: "use ocl bremen" -> Sourceforge USE project page
- Validation and verification tool for UML and OCL models
- UML class, statechart, object, sequence, and communication diagrams
- OCL support for
  + class invariants and operation pre- and postconditions
  + query operations and ad-hoc queries
  + derivation rules for attributes and associations
  + state invariants, transition guards and transition postconditions
- Imperative action language for implementing non-query operations on the model level: SOIL (Simple Ocl-based Imperative Language)
- Model validation by executing test scenarios
- Automatic generation of object diagrams through a model validator based on a translation of UML and OCL into relational logic (realized in Kodkod/Alloy) starting from a class diagram and invariants
- Verification of model properties like model consistency, model minimality (invariant independence) or model state reachability






```
context p:Person inv balancedBinaryTree:
  (p.child->size=0 or p.child->size=2) and
  Person.allInstances->one(r | r.parent->size=0 and -- root
    Person.allInstances->excluding(r)->forAll(p | p.parent->size=1)) and
  p.child->forAll(c1,c2 | -- balance
    c1.child->closure(child)->size = c2.child->closure(child)->size)

Person_min = 15; Person_max = 15

Person_fName = Set{'Ada','Bob','Cyd','Dan','Eve'}
Person_lName = Set{'Alewife','Baker','Cook','Digger','Eggler'}
Person_yearB = Set{1905,1920,1935,1950,1965,1980,1995}

Parenthood_min = 0; Parenthood_max = *
```


Object diagram
p9:Person
fName='Cyd'
lName='Alewife'
yearB=1905
p14:Person
fName='Bob'
lName='Cook'
yearB=1920
p15:Person
fName='Dan'
lName='Alewife'
yearB=1950
p11:Person
fName='Eve'
lName='Cook'
yearB=1980
p12:Person
fName='Eve'
lName='Digger'
yearB=1935
p10:Person
fName='Dan'
lName='Eggler'
yearB=1980
p13:Person
fName='Ada'
lName='Alewife'
yearB=1980
p5:Person
fName='Bob'
lName='Baker'
yearB=1995
p6:Person
fName='Cyd'
lName='Eggler'
yearB=1995
p3:Person
fName='Cyd'
lName='Cook'
yearB=1995
p4:Person
fName='Ada'
lName='Eggler'
yearB=1950
p7:Person
fName='Eve'
lName='Baker'
yearB=1995
p8:Person
fName='Dan'
lName='Cook'
yearB=1995
p1:Person
fName='Eve'
lName='Eggler'
yearB=1995
p2:Person
fName='Bob'
lName='Eggler'
yearB=1995

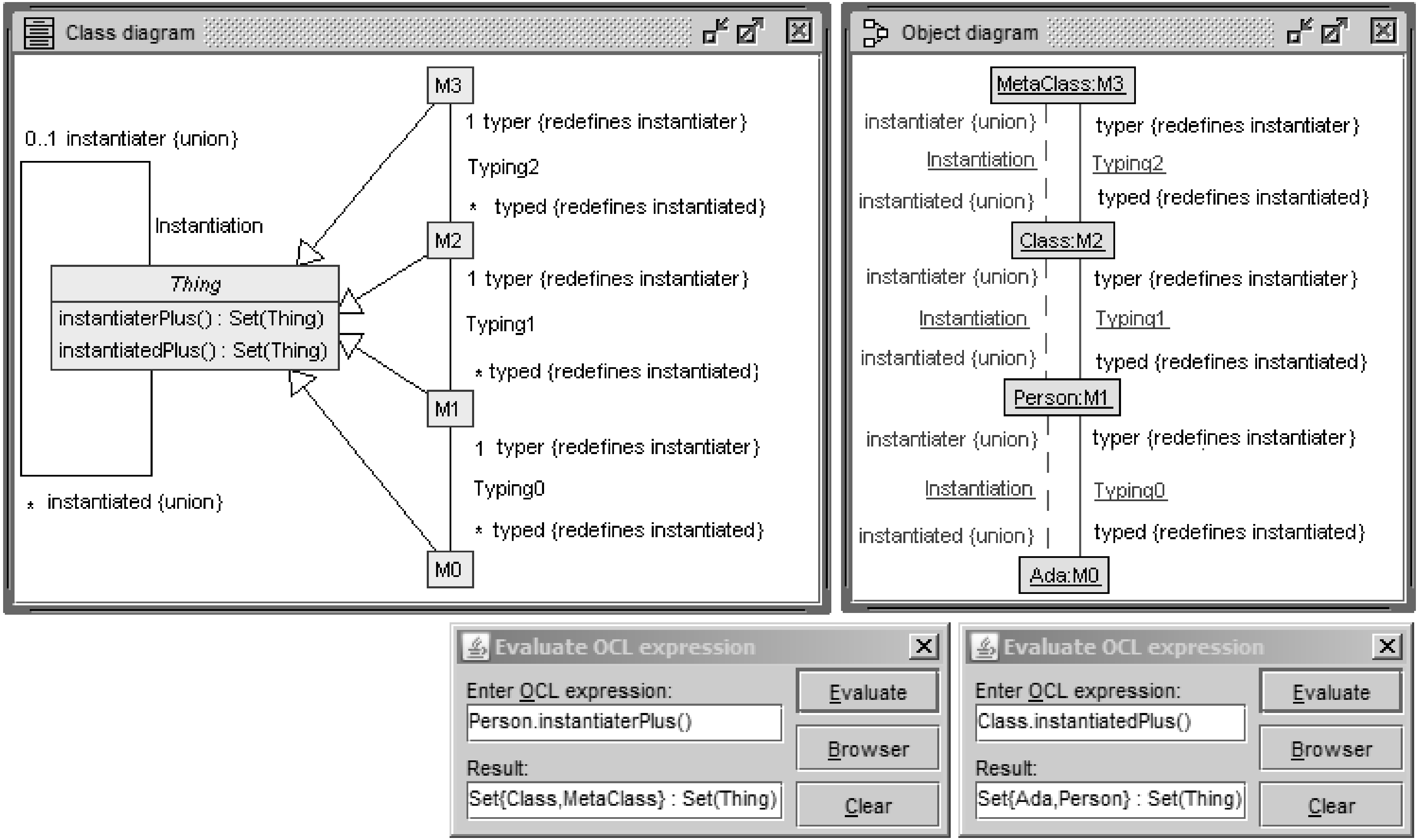

provide OCL invariants
provide class model
provide configuration
provide model and configuration
model consistency
partial solution completion
property reachability
solution interval exploration
constraint implication
constraint independence
provide additional OCL invariant
provide partial system state
all use case relationships: «include»

Structure of the talk

- Our context: USE (Uml-based Specification Environment)
- First approach: Metamodel level connection with associations and generalizations
- Second approach: Metamodel level connection with special OCL(?) operations
- A touch of related work
- Conclusion

Example 1

Ada is a Person, Person is a Class, Class is MetaClass

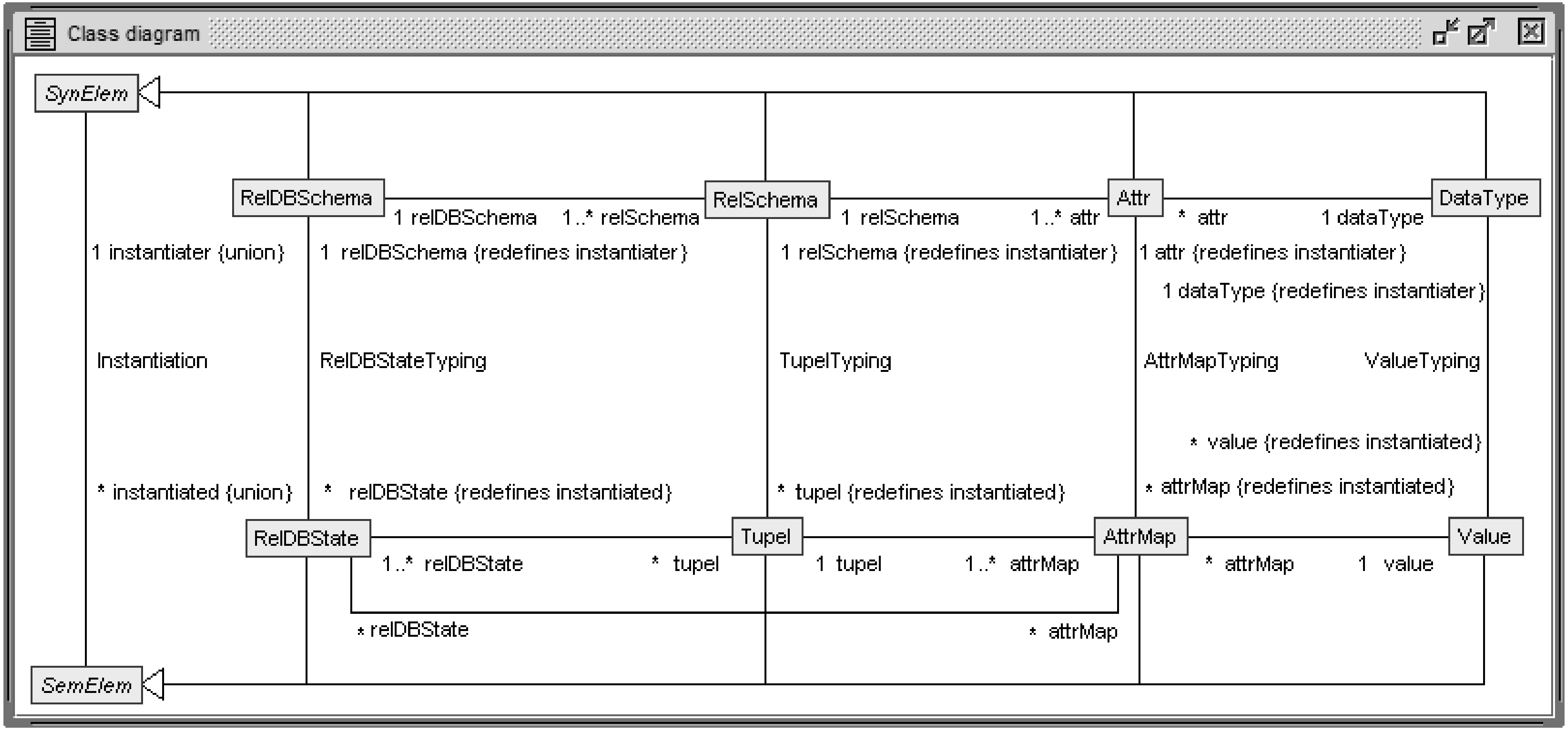

Class diagram
M3
1 typer {redefines instantiater}
Typing2
* typed {redefines instantiated}
M2
1 typer {redefines instantiater}
Typing1
* typed {redefines instantiated}
M1
1 typer {redefines instantiater}
Typing0
* typed {redefines instantiated}
M0
0..1 instantiater {union}
Instantiation
Thing
instantiaterPlus() : Set(Thing)
instantiatedPlus() : Set(Thing)
* instantiated {union}
Object diagram
MetaClass:M3
instantiater {union}
Instantiation
instantiated {union}
typer {redefines instantiater}
Typing2
typed {redefines instantiated}
Class:M2
instantiater {union}
Instantiation
instantiated {union}
typer {redefines instantiater}
Typing1
typed {redefines instantiated}
Person:M1
instantiater {union}
Instantiation
instantiated {union}
typer {redefines instantiater}
Typing0
typed {redefines instantiated}
Ada:M0
Evaluate OCL expression
Enter OCL expression:
Person.instantiaterPlus()
Evaluate
Browser
Result:
Set{Class,MetaClass} : Set(Thing)
Clear
Evaluate OCL expression
Enter OCL expression:
Class.instantiatedPlus()
Evaluate
Browser
Result:
Set{Ada,Person} : Set(Thing)
Clear

```
abstract class Thing
operations
  instantiatedPlus():Set(Thing)=
    self.instantiated->closure(t|t.instantiated)
  instantiaterPlus():Set(Thing)= ...

constraints
  inv acyclicInstantiation: self.instantiatedPlus()->excludes(self)
end
```

Example 2: Relational data model

Metamodel level 1 - Database schemata (Syntax)

Metamodel level 0 - Database states (Semantics)



Class invariants

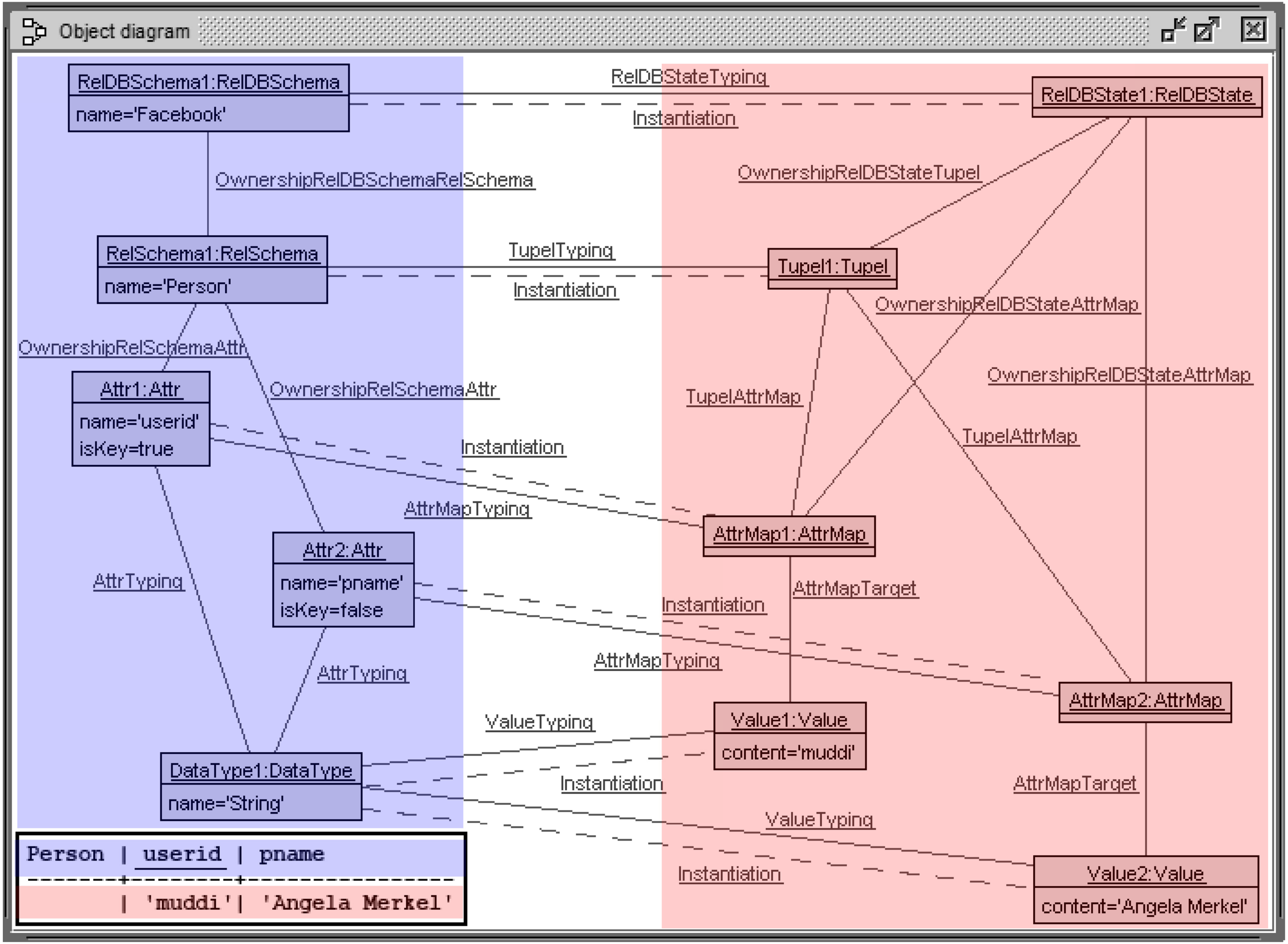
Invariant	Result
DataType::uniqueDataTypeNames	true
RelDBSchema::uniqueRelDBSchemaNames	true
RelDBSchema::uniqueRelSchemaNamesWithinRelDBSchema	true
RelSchema::relSchemaKeyNotEmpty	true
RelSchema::uniqueAttrNamesWithinRelSchema	true

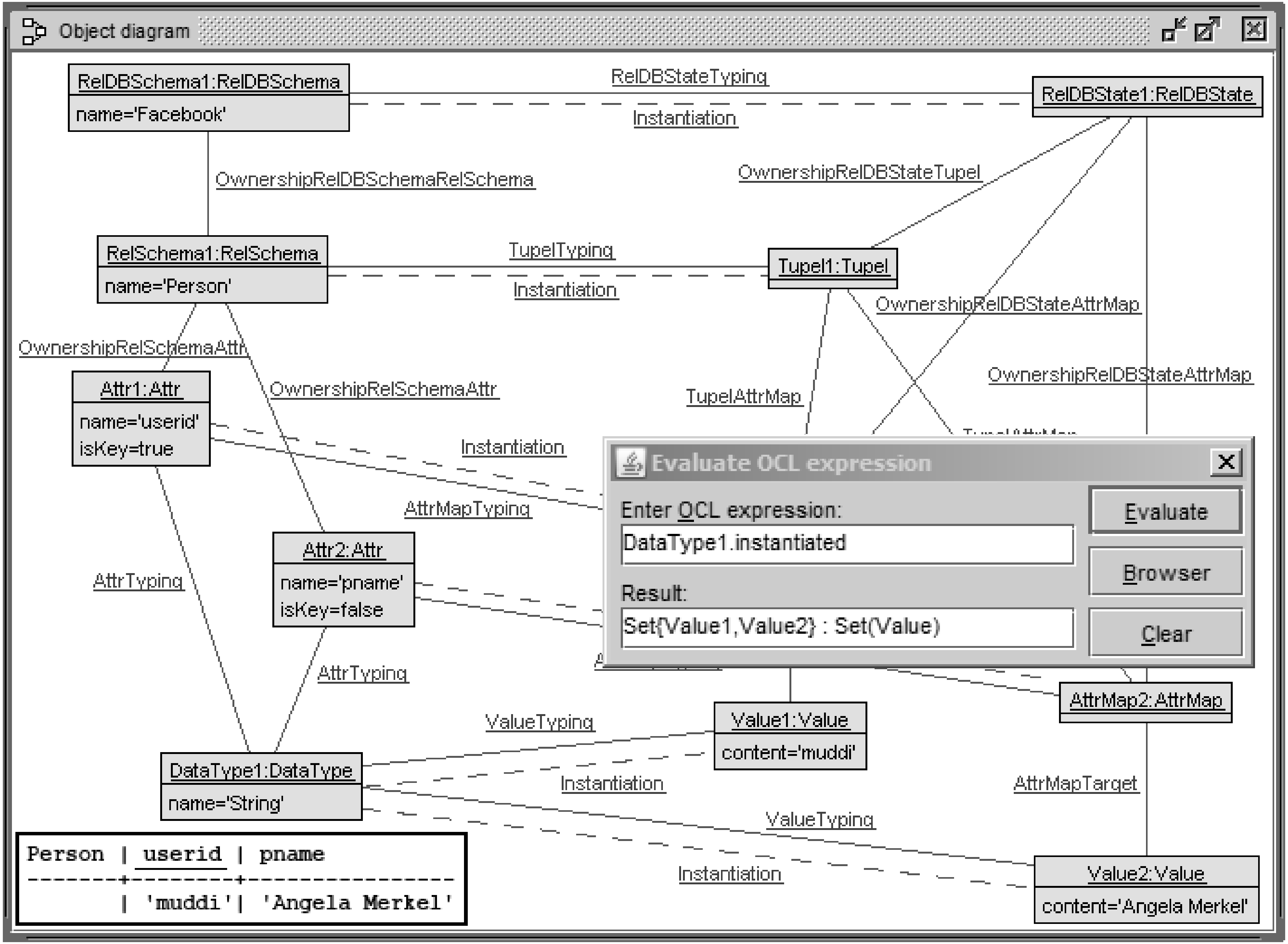
Constraints ok. (0ms) 100%

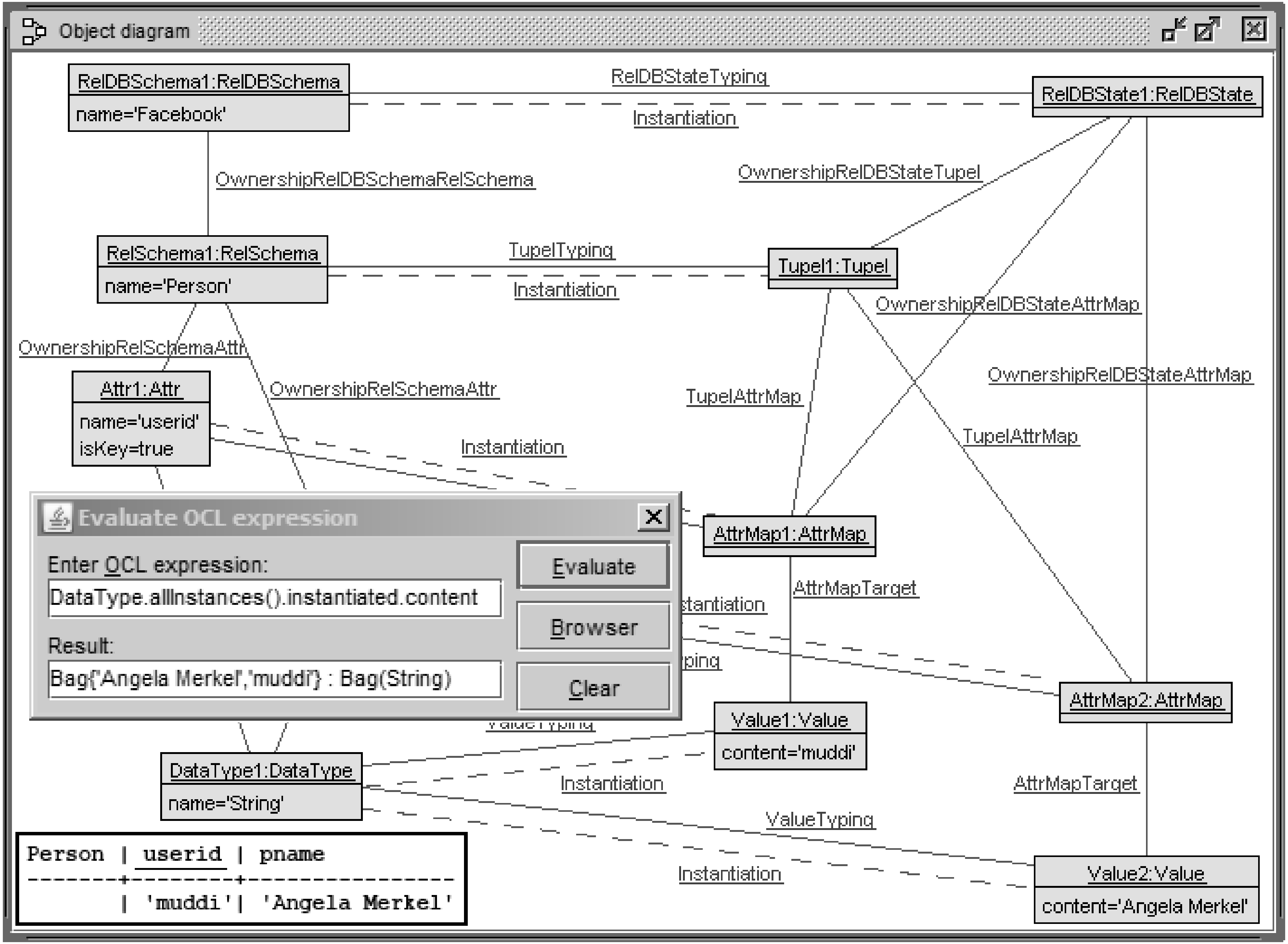
Class invariants

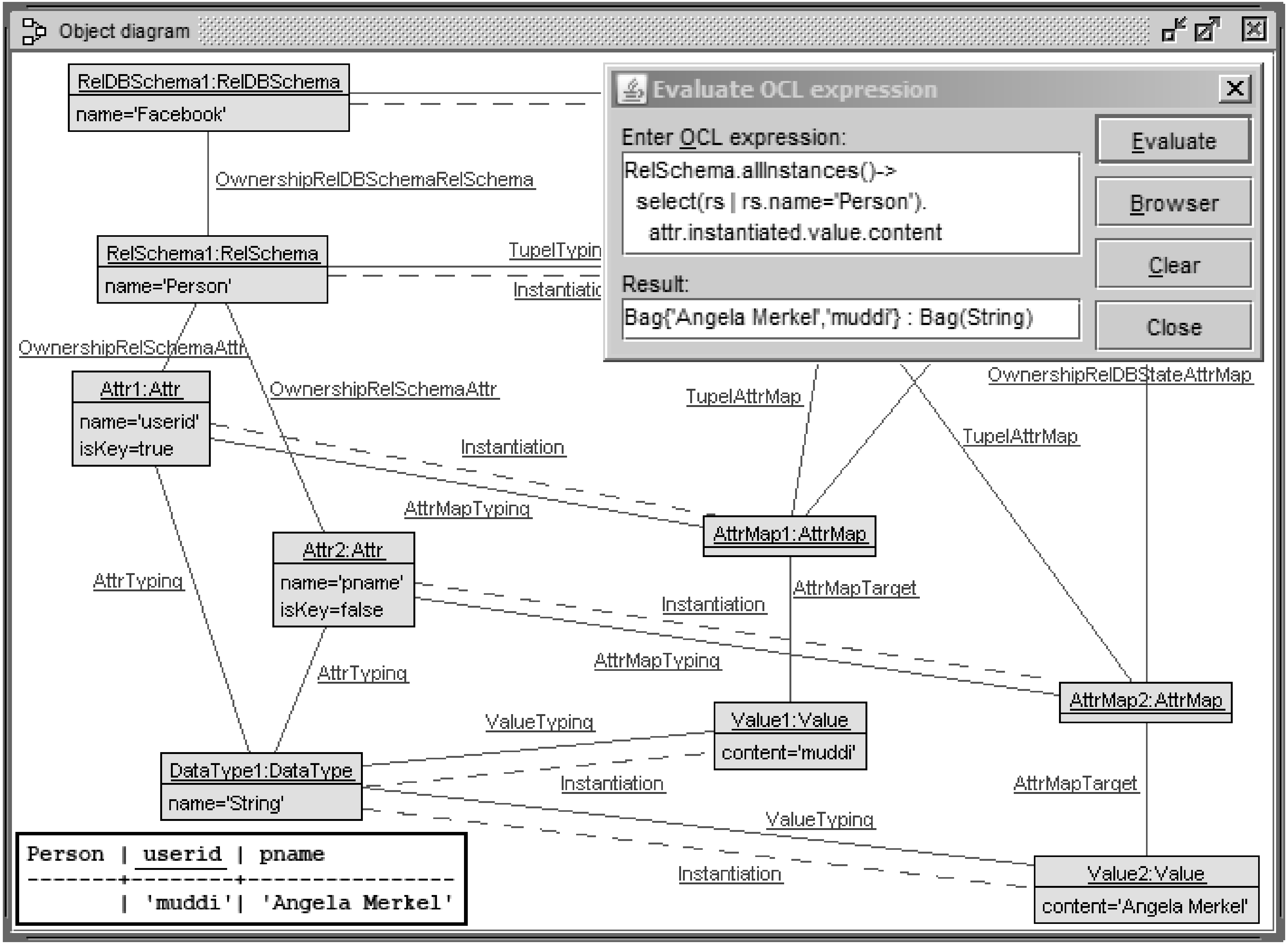
Invariant	Result
AttrMap::c_AttrMap_Attr_Tupel_RelSchema	true
AttrMap::c_AttrMap_Attr_Value_DataType	true
AttrMap::c_AttrMap_Tupel_RelDBState	true
AttrMap::tupelAttrMapIsFunction	true
Tupel::c_Tupel_RelSchema_AttrMap_Attr	true
Tupel::c_Tupel_RelSchema_RelDBState_RelDBSchema	true
Tupel::keyMapUnique	true
Value::differentContentOrDataType	true

Constraints ok. (0ms) 100%

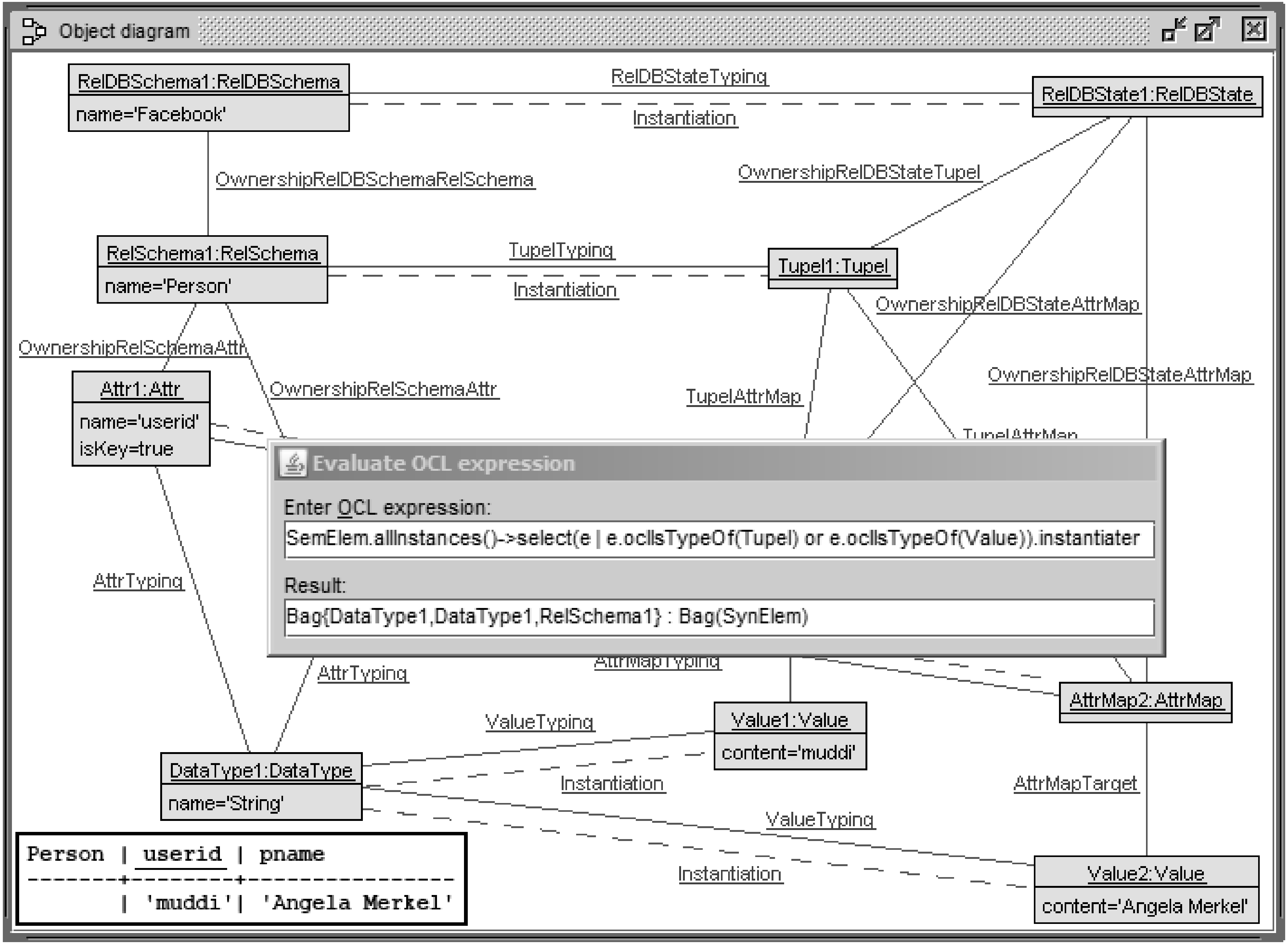

Object diagram
RelDBSchema1:RelDBSchema
name='Facebook'
RelDBStateTyping
Instantiation
RelDBState1:RelDBState
OwnershipRelDBSchemaRelSchema
OwnershipRelDBStateTupel
RelSchema1:RelSchema
name='Person'
TupelTyping
Instantiation
Tupel1:Tupel
OwnershipRelDBStateAttrMap
OwnershipRelSchemaAttr
Attr1:Attr
name='userid'
isKey=true
OwnershipRelSchemaAttr
TupelAttrMap
OwnershipRelDBStateAttrMap
TupelAttrMap
Instantiation
AttrMapTyping
AttrMap1:AttrMap
Attr2:Attr
name='pname'
isKey=false
AttrTyping
AttrMapTarget
Instantiation
AttrMapTyping
AttrTyping
AttrMap2:AttrMap
ValueTyping
Value1:Value
content='muddi'
DataType1:DataType
name='String'
Instantiation
AttrMapTarget
ValueTyping
Instantiation
Value2:Value
content='Angela Merkel'
Person | userid | pname
-------+--------+---------------
| 'muddi'| 'Angela Merkel'

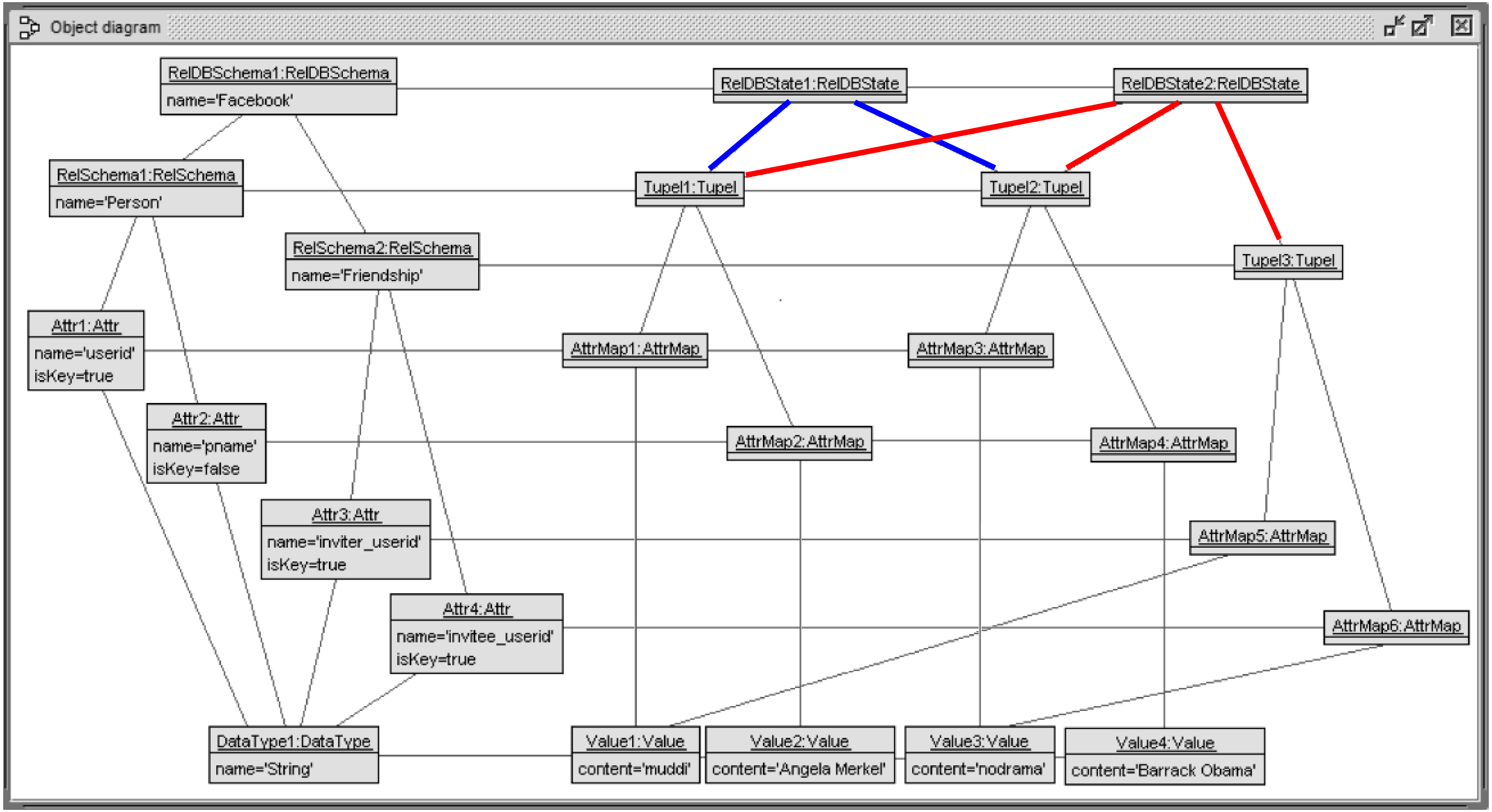

Object diagram
RelDBSchema1:RelDBSchema
name='Facebook'
RelDBStateTyping
Instantiation
RelDBState1:RelDBState
OwnershipRelDBSchemaRelSchema
OwnershipRelDBStateTupel
RelSchema1:RelSchema
name='Person'
TupelTyping
Instantiation
Tupel1:Tupel
OwnershipRelDBStateAttrMap
OwnershipRelSchemaAttr
Attr1:Attr
name='userid'
isKey=true
OwnershipRelSchemaAttr
TupelAttrMap
OwnershipRelDBStateAttrMap
Instantiation
AttrMapTyping
Evaluate OCL expression
Enter OCL expression:
DataType1.instantiated
Result:
Set{Value1,Value2} : Set(Value)
Evaluate
Browser
Clear
AttrTyping
Attr2:Attr
name='pname'
isKey=false
AttrTyping
ValueTyping
Value1:Value
content='muddi'
AttrMap2:AttrMap
DataType1:DataType
name='String'
Instantiation
AttrMapTarget
ValueTyping
Instantiation
Value2:Value
content='Angela Merkel'
Person | userid | pname
-------+--------+----------------
| 'muddi'| 'Angela Merkel'


Object diagram
RelDBSchema1:RelDBSchema
name='Facebook'
RelDBStateTyping
Instantiation
RelDBState1:RelDBState
OwnershipRelDBSchemaRelSchema
OwnershipRelDBStateTupel
RelSchema1:RelSchema
name='Person'
TupelTyping
Instantiation
Tupel1:Tupel
OwnershipRelDBStateAttrMap
OwnershipRelSchemaAttr
Attr1:Attr
name='userid'
isKey=true
OwnershipRelSchemaAttr
Instantiation
TupelAttrMap
OwnershipRelDBStateAttrMap
TupelAttrMap
Evaluate OCL expression
Enter OCL expression:
DataType.allInstances().instantiated.content
Result:
Bag{'Angela Merkel','muddi'} : Bag(String)
Evaluate
Browser
Clear
AttrMap1:AttrMap
AttrMapTarget
AttrMap2:AttrMap
Value1:Value
content='muddi'
DataType1:DataType
name='String'
Instantiation
AttrMapTarget
ValueTyping
Instantiation
Value2:Value
content='Angela Merkel'
Person | userid | pname
--------+--------+----------------
| 'muddi'| 'Angela Merkel'



Object diagram
RelDBSchema1:RelDBSchema
name='Facebook'
OwnershipRelDBSchemaRelSchema
RelSchema1:RelSchema
name='Person'
TupelTypin
Instantiatic
Evaluate OCL expression
Enter OCL expression:
RelSchema.allInstances()->
select(rs | rs.name='Person').
attr.instantiated.value.content
Result:
Bag{'Angela Merkel','muddi'} : Bag(String)
Evaluate
Browser
Clear
Close
OwnershipRelSchemaAttr
Attr1:Attr
name='userid'
isKey=true
OwnershipRelSchemaAttr
TupelAttrMap
OwnershipRelDBStateAttrMap
TupelAttrMap
Instantiation
AttrMapTyping
AttrMap1:AttrMap
Attr2:Attr
name='pname'
isKey=false
AttrTyping
AttrMapTarget
Instantiation
AttrMapTyping
AttrMap2:AttrMap
AttrTyping
ValueTyping
Value1:Value
content='muddi'
DataType1:DataType
name='String'
Instantiation
AttrMapTarget
ValueTyping
Instantiation
Value2:Value
content='Angela Merkel'
Person | userid | pname
| 'muddi'| 'Angela Merkel'


Object diagram
RelDBSchema1:RelDBSchema
name='Facebook'
RelDBStateTyping
Instantiation
RelDBState1:RelDBState
OwnershipRelDBSchemaRelSchema
OwnershipRelDBStateTupel
RelSchema1:RelSchema
name='Person'
TupelTyping
Instantiation
Tupel1:Tupel
OwnershipRelDBStateAttrMap
OwnershipRelSchemaAttr
Attr1:Attr
name='userid'
isKey=true
OwnershipRelSchemaAttr
TupelAttrMap
OwnershipRelDBStateAttrMap
Evaluate OCL expression
Enter OCL expression:
SemElem.allInstances()->select(e | e.oclIsTypeOf(Tupel) or e.oclIsTypeOf(Value)).instantiater
Result:
Bag{DataType1,DataType1,RelSchema1} : Bag(SynElem)
AttrTyping
AttrTyping
AttrMap2:AttrMap
ValueTyping
Value1:Value
content='muddi'
DataType1:DataType
name='String'
Instantiation
AttrMapTarget
ValueTyping
Instantiation
Value2:Value
content='Angela Merkel'
Person | userid | pname
| 'muddi'| 'Angela Merkel'

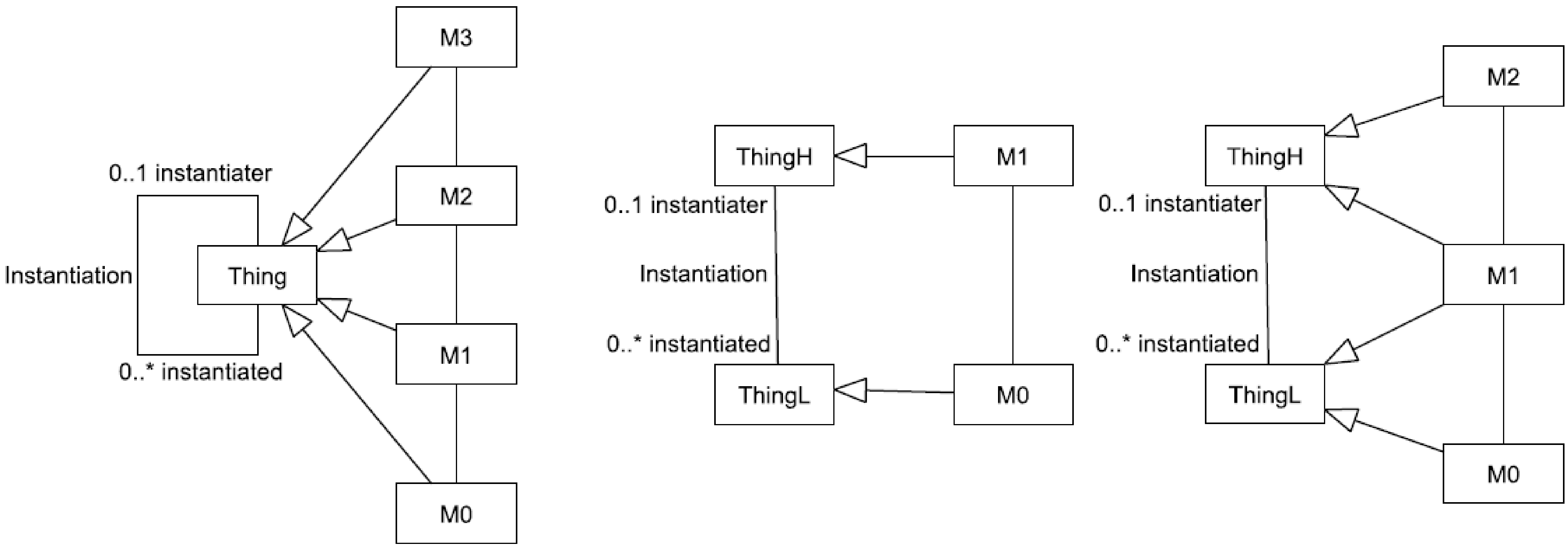



```
Person | userid     | pname
-------+------------+-----------------
       | 'muddi'    | 'Angela Merkel'
       | 'nodrama'  | 'Barrack Obama'
```

```
insert into Friendship
     values ('muddi','nodrama')
```

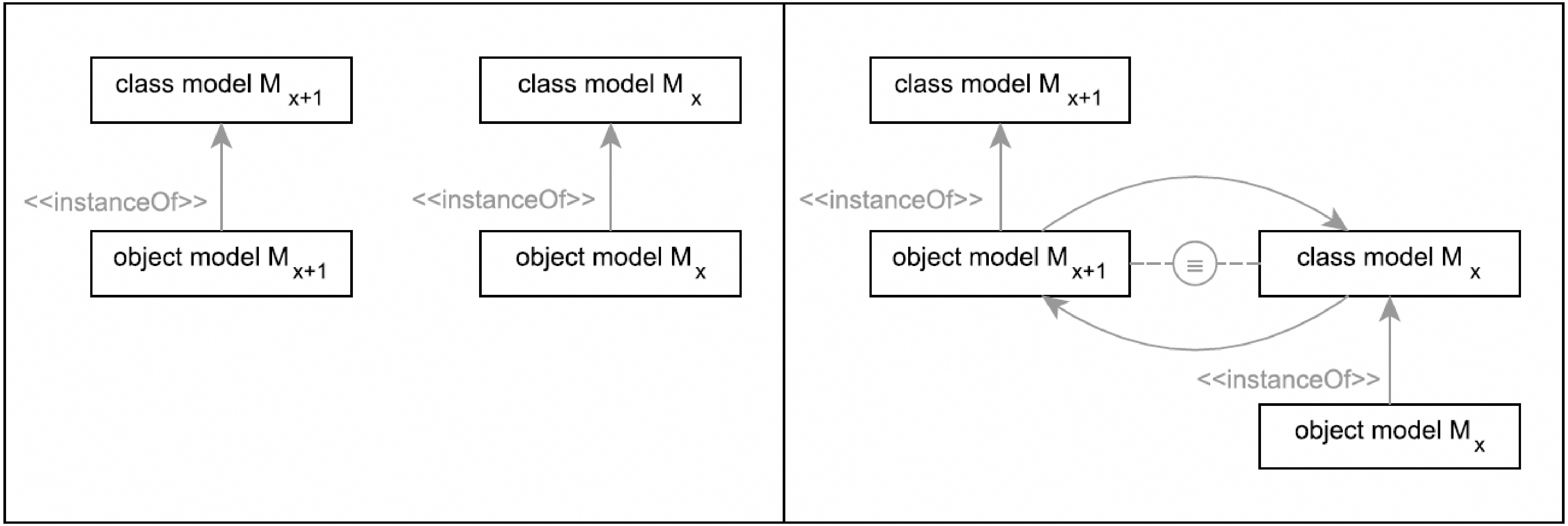
```
Person | userid     | pname
-------+------------+-----------------
       | 'muddi'    | 'Angela Merkel'
       | 'nodrama'  | 'Barrack Obama'

Friendship | inviter_userid | invitee_userid
-----------+----------------+---------------
           | 'muddi'        | 'nodrama'
```

Different metamodel structures

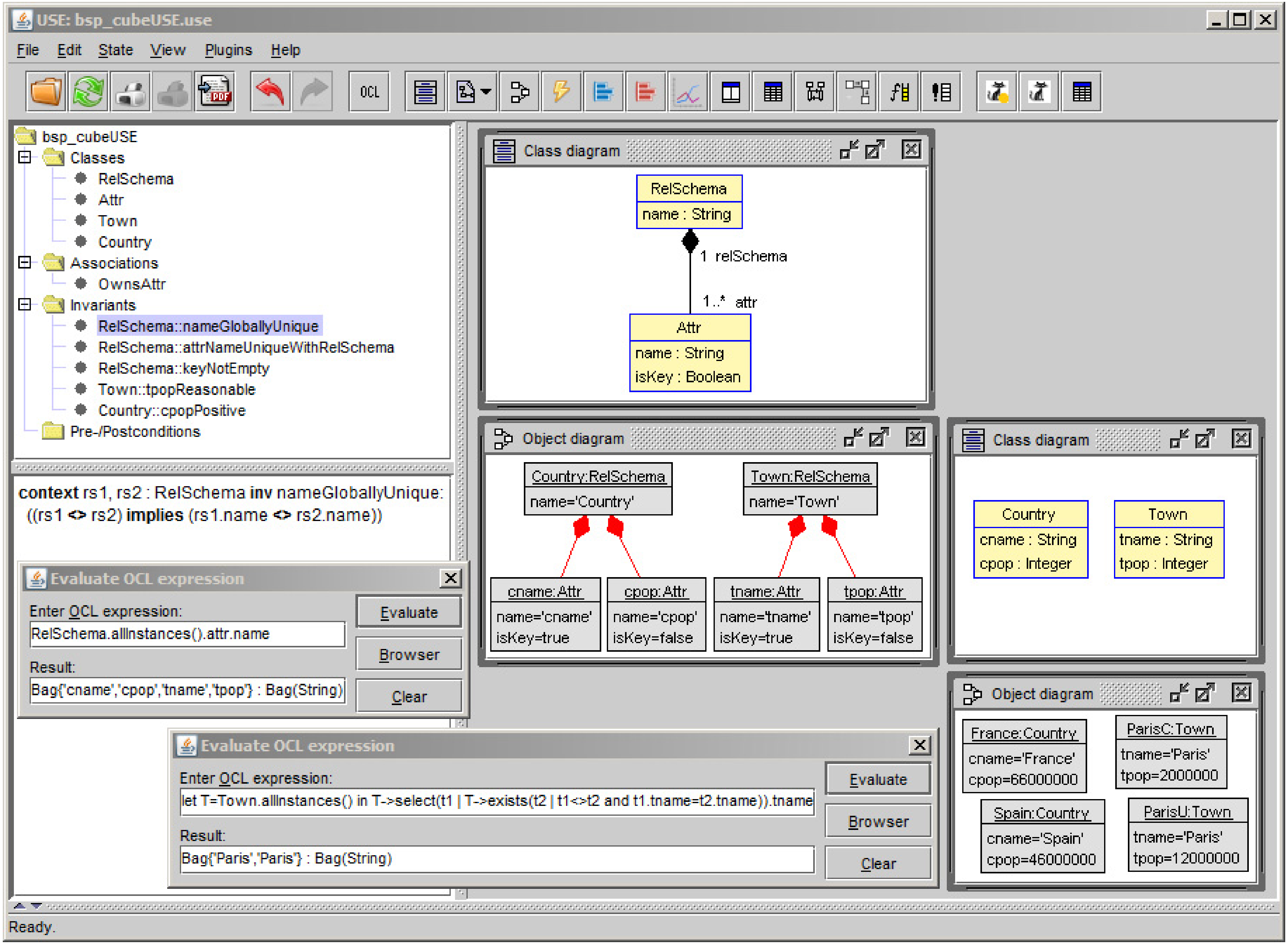


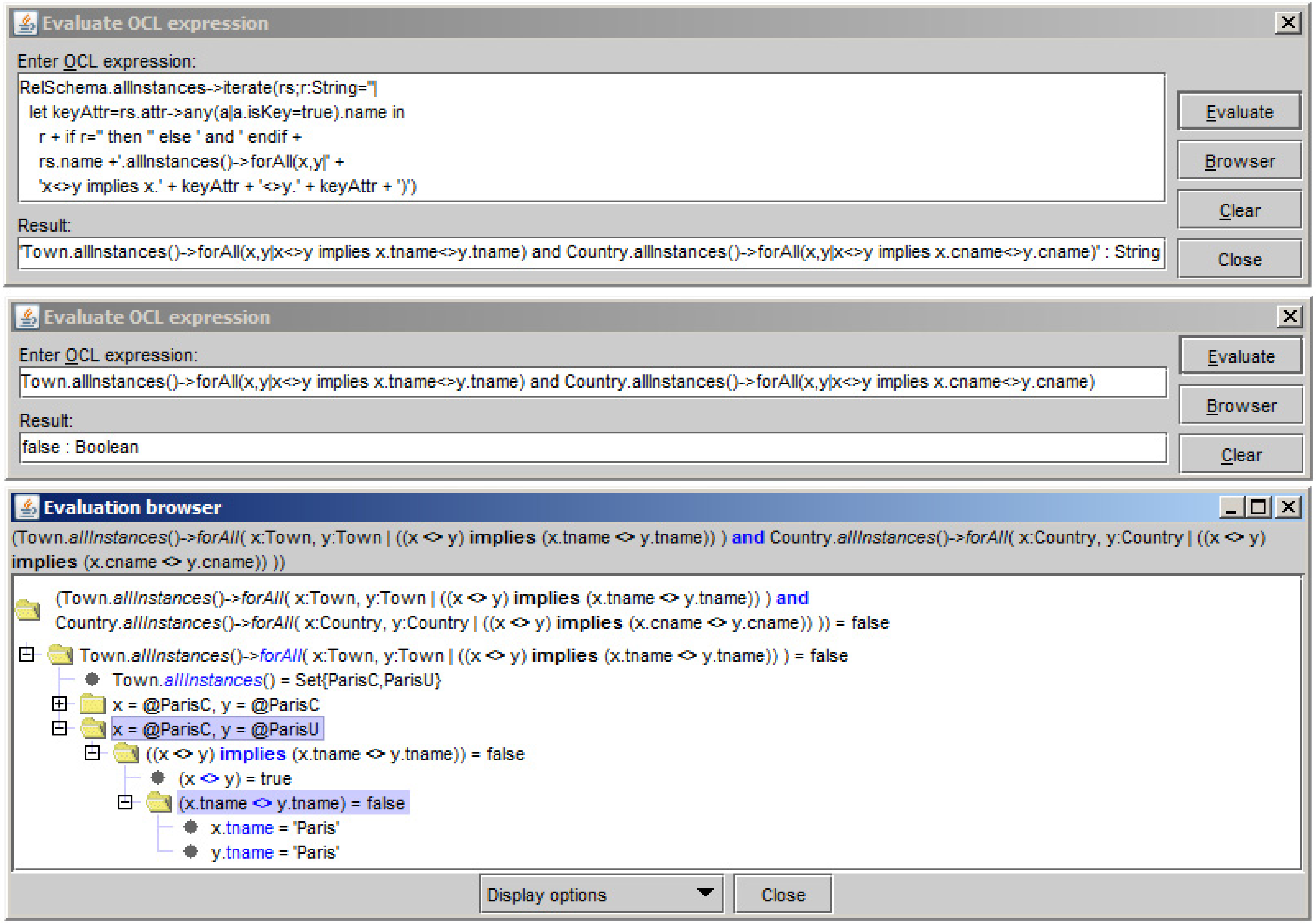
Structure of the talk

- **Our context: USE (Uml-based Specification Environment)**
- **First approach: Metamodel level connection with associations and generalizations**
- **Second approach: Metamodel level connection with special OCL(?) operations**
- **A touch of related work**
- **Conclusion**



class model M x+1
<<instanceOf>>
object model M x+1
class model M x
<<instanceOf>>
object model M x
class model M x+1
<<instanceOf>>
object model M x+1
≡
class model M x
<<instanceOf>>
object model M x


USE: bsp_cubeUSE.use
File Edit State View Plugins Help
bsp_cubeUSE
Classes
RelSchema
Attr
Town
Country
Associations
OwnsAttr
Invariants
RelSchema::nameGloballyUnique
RelSchema::attrNameUniqueWithRelSchema
RelSchema::keyNotEmpty
Town::tpopReasonable
Country::cpopPositive
Pre-/Postconditions
context rs1, rs2 : RelSchema inv nameGloballyUnique:
((rs1 <> rs2) implies (rs1.name <> rs2.name))
Evaluate OCL expression
Enter OCL expression:
RelSchema.allInstances().attr.name
Result:
Bag{'cname','cpop','tname','tpop'} : Bag(String)
Evaluate
Browser
Clear
Class diagram
RelSchema
name : String
1 relSchema
1..* attr
Attr
name : String
isKey : Boolean
Object diagram
Country:RelSchema
name='Country'
Town:RelSchema
name='Town'
cname:Attr
name='cname'
isKey=true
cpop:Attr
name='cpop'
isKey=false
tname:Attr
name='tname'
isKey=true
tpop:Attr
name='tpop'
isKey=false
Class diagram
Country
cname : String
cpop : Integer
Town
tname : String
tpop : Integer
Object diagram
France:Country
cname='France'
cpop=66000000
ParisC:Town
tname='Paris'
tpop=2000000
Spain:Country
cname='Spain'
cpop=46000000
ParisU:Town
tname='Paris'
tpop=12000000
Evaluate OCL expression
Enter OCL expression:
let T=Town.allInstances() in T->select(t1 | T->exists(t2 | t1<>t2 and t1.tname=t2.tname)).tname
Result:
Bag{'Paris','Paris'} : Bag(String)
Evaluate
Browser
Clear
Ready.


Evaluate OCL expression
Enter OCL expression:
RelSchema.allInstances->iterate(rs;r:String=''|
let keyAttr=rs.attr->any(a|a.isKey=true).name in
r + if r='' then '' else ' and ' endif +
rs.name +'.allInstances()->forAll(x,y|' +
'x<>y implies x.' + keyAttr + '<>y.' + keyAttr + ')')
Evaluate
Browser
Clear
Close
Result:
'Town.allInstances()->forAll(x,y|x<>y implies x.tname<>y.tname) and Country.allInstances()->forAll(x,y|x<>y implies x.cname<>y.cname)' : String
Evaluate OCL expression
Enter OCL expression:
Town.allInstances()->forAll(x,y|x<>y implies x.tname<>y.tname) and Country.allInstances()->forAll(x,y|x<>y implies x.cname<>y.cname)
Evaluate
Browser
Clear
Result:
false : Boolean
Evaluation browser
(Town.allInstances()->forAll(x:Town, y:Town | ((x <> y) implies (x.tname <> y.tname))) and Country.allInstances()->forAll(x:Country, y:Country | ((x <> y) implies (x.cname <> y.cname))))
(Town.allInstances()->forAll(x:Town, y:Town | ((x <> y) implies (x.tname <> y.tname))) and Country.allInstances()->forAll(x:Country, y:Country | ((x <> y) implies (x.cname <> y.cname)))) = false
Town.allInstances()->forAll(x:Town, y:Town | ((x <> y) implies (x.tname <> y.tname))) = false
Town.allInstances() = Set{ParisC,ParisU}
x = @ParisC, y = @ParisC
x = @ParisC, y = @ParisU
((x <> y) implies (x.tname <> y.tname)) = false
(x <> y) = true
(x.tname <> y.tname) = false
x.tname = 'Paris'
y.tname = 'Paris'
Display options
Close

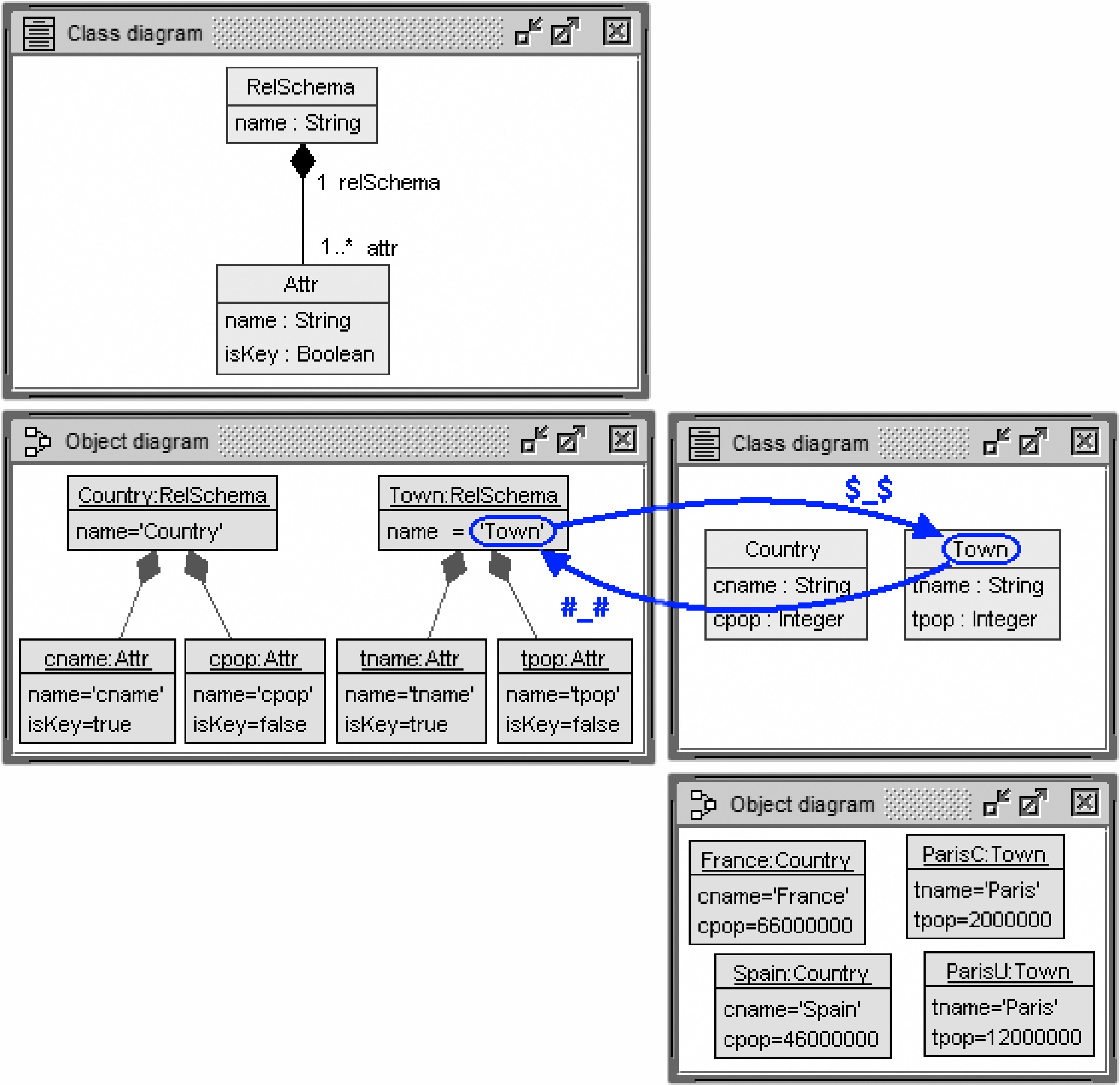
```
parameter[rs:RelSchema]
let relSchemaClass = $rs.name$ in
let keyAttr = $rs.attr->any(a|a.isKey=true).name$ in
context relSchemaClass inv keyAttrUnique:
  relSchemaClass.allInstances->forAll(x,y |
    x<>y implies x.keyAttr<>y.keyAttr)
```

```
context Town inv keyAttrUnique:
  Town.allInstances->forAll(x,y |
    x<>y implies x.name<>y.name)
```

```
new OCL features:
- OCL clauses with parameters that are variables for model elements
- special expressions for model elements (e.g., for class or attribute)
- an operation accessing a model element through its String-valued name
  $_$ : String -> ModelElement
- an operation returning the String-valued name of a model element
  #_# : ModelElement -> String
```

```
parameter[rs:RelSchema]
let relSchemaClass = $rs.name$ in
let keyAttr = $rs.attr->any(a|a.isKey=true).name$ in
context x,y:relSchemaClass inv 'keyAttrUniqueIn' + #rs#:
  x<>y implies x.keyAttr<>y.keyAttr
```

```
context x,y:Town inv keyAttrUniqueInTown:
  x<>y implies x.name<>y.name
```



Class diagram
RelSchema
name : String
1 relSchema
1..* attr
Attr
name : String
isKey : Boolean
Object diagram
Country:RelSchema
name='Country'
Town:RelSchema
name = 'Town'
cname:Attr
name='cname'
isKey=true
cpop:Attr
name='cpop'
isKey=false
tname:Attr
name='tname'
isKey=true
tpop:Attr
name='tpop'
isKey=false
Class diagram
$_$
#_#
Country
cname : String
cpop : Integer
Town
tname : String
tpop : Integer
Object diagram
France:Country
cname='France'
cpop=66000000
ParisC:Town
tname='Paris'
tpop=2000000
Spain:Country
cname='Spain'
cpop=46000000
ParisU:Town
tname='Paris'
tpop=12000000

Structure of the talk

- Our context: USE (Uml-based Specification Environment)
- First approach: Metamodel level connection with associations and generalizations
- Second approach: Metamodel level connection with special OCL(?) operations
- A touch of related work
- Conclusion

A touch of related work

- Guerra / de Lara (MULTI WS 2014)

  Towards Automating the Analysis of Integrity Constraints in Multi-Level Models

- Igamberdiev / Grossmann / Stumptner (MULTI WS 2014)

  An Implementation of Multi-Level Modelling in F-logic

- Clark / Gonzalez-Perez / Henderson-Sellers (MULTI WS 2014)

  A Foundation for Multi-Level Modelling

- Atkinson / Gerbig / Kühne (OCL WS 2015)

  Opportunities and Challenges for Deep Constraint Languages

- Atkinson / Gerbig / Kühne (MODELS 2015)

  A Unifying Approach to Connections for Multi-Level Modeling Foundations

... [my apologies to the many good works that i did not mention]

Structure of the talk

- Our context: USE (Uml-based Specification Environment)
- First approach: Metamodel level connection with associations and generalizations
- Second approach: Metamodel level connection with special OCL(?) operations
- A touch of related work
- Conclusion

Summary

- presented approaches for incorporating different metamodel levels into a single model

- employed
  + associations, generalizations and OCL for restricting the connection between metamodel levels
  + special OCL(?) operations

Future work

- discover connections to and formalize notions like clabject, potency, powertype

- build more case studies in order to obtain more insights into advantages and drawbacks

- extend our tool USE to cope with (at least) three modeling levels
  - class diagram
  - object diagram = class diagram
  -                  object diagram

...

Thanks for your attention!

```
context t1,t2:Tupel inv keyMapUnique:
t1<>t2 and t1.relSchema=t2.relSchema
    implies
    t1.relDBState->intersection(t2.relDBState)->forAll(s |
      t1.relSchema.key()->exists(ka |
        t1.applyAttr(s,ka)<>t2.applyAttr(s,ka)))
```