Initiating a Benchmark for UML and OCL Analysis Tools

Martin Gogolla[(A)], Fabian Büttner[(B)⋆], Jordi Cabot[(B)]

University of Bremen, Germany[(A)],
AtlanMod, École des Mines de Nantes - INRIA, LINA, France[(B)]

Abstract. The Object Constraint Language (OCL) is becoming more and more popular for model-based engineering, in particular for the development of models and model transformations. OCL is supported by a variety of analysis tools having different scopes, aims and technological corner stones. The spectrum ranges from treating issues concerning formal proof techniques to testing approaches, from validation to verification, and from logic programming and rewriting to SAT-based technologies. This paper is a first step towards a well-founded benchmark for assessing validation and verification techniques on UML and OCL models. The paper puts forward a set of UML and OCL models together with particular questions for these models roughly characterized by the notions consistency, independence, consequences, and reachability. The paper sketches how these questions are handled by two OCL tools, USE and EMFtoCSP. The claim of the paper is not to present a complete benchmark right now. The paper is intended to initiate the development of further UML and OCL models and accompanying questions within the UML and OCL community. The OCL community is invited to check the presented UML and OCL models with their approaches and tools and to contribute further models and questions which emphasize the possibilities offered by their own tools.

1 Introduction

Model-driven engineering (MDE) as a paradigm for software development is gaining more and more importance. Models and model transformations are central notions in modeling languages like UML, SysML, or EMF and transformation languages like QVT or ATL. In these approaches, the Object Constraint Language (OCL) can be employed for expressing constraints and operations, thus OCL plays a central role in MDE. A variety of OCL tools is currently available, but it is an open issue how to compare these tools and how to support developers in choosing the OCL tool appropriate for their project. This paper puts forward a set of UML and OCL models together with particular questions for these models. This set of models is intended to be a first version of an OCL analysis tool benchmark to be developed within the OCL and UML community.

⋆ This research was partially funded by the Nouvelles Eq́uipes program of the Pays de la Loire region (France).

The current benchmark consists of four UML and OCL models: CivilStatus (CS), WritesReviews (WR), DisjointSubclasses (DS), and ObjectsAsIntegers (OAI). These models employ and emphasize different UML and OCL language features and pose different computational challenges for the analysis tools and their underlying technologies like provers, solvers, or finders: Plain invariants and enumerations in CS, association multiplicities in WR, classifier generalization in DS, and recursive operation definitions with inherited association ends and constraints in OAI. The accompanying questions can be roughly characterized by the partly overlapping notions consistency, independence, consequences, and reachability: under the label 'consistency' we discuss whether there exist object diagrams for the model at all, 'independence' concentrates on whether the invariants are non-redundant, 'consequences' studies how to formally deduce new properties from the explicitly stated ones, and 'reachability' focuses on how to characterize all object diagrams of a model and how to construct an object diagram with stated properties. The benchmark does not expect that all questions can be fully answered by a considered tool, but it expects that it is discussed to what extent and in which direction an approach or tool can help to answer the question.

The structure of the rest of this paper is as follows. The next section gives a short introduction to OCL. Section 3 introduces the first version of our benchmark. Four example models with accompanying questions are introduced. As a proof of concept for the applicability of the models, Sect. 4 and Sect. 5 show how these models and questions are handled by two concrete tools and how the models must be fine-tuned to become processable by the respective tool, if needed. These two tools have been selected to illustrate how the models can be used to evaluate tools. Section 6 puts forward a list of topics that could be addressed in future work. Section 7 discusses related work and some (not all) approaches suitable to be subject to an OCL analysis tool benchmark. The paper is finished in Sect. 8 with concluding remarks. Furthermore, the paper is extended by an additional document [14] in which all models are detailed in the formats `.use` and `.ecore` and all details of the benchmark examples for the tools USE and EMFtoCSP are made available.

2 OCL in 5 Minutes

The Object Constrains Language (OCL) is a textual, descriptive expression language. OCL is side effect free and is mainly used for phrasing constraints and queries in object-oriented models. Most OCL expressions rely on a class model which is expressed in a graphical modeling language like UML, MOF or EMF. The central concepts in OCL are objects, object navigation, collections, collection operations and boolean-valued expressions, i.e., formulas. Let us consider these concepts in connection with the object diagram in Fig. 1 which belongs to the class diagram in Fig. 3. This class diagram captures part of the submission and reviewing process of conference papers. A more detailed description of the class diagram and the corresponding constraints is given later in Section 2.2.

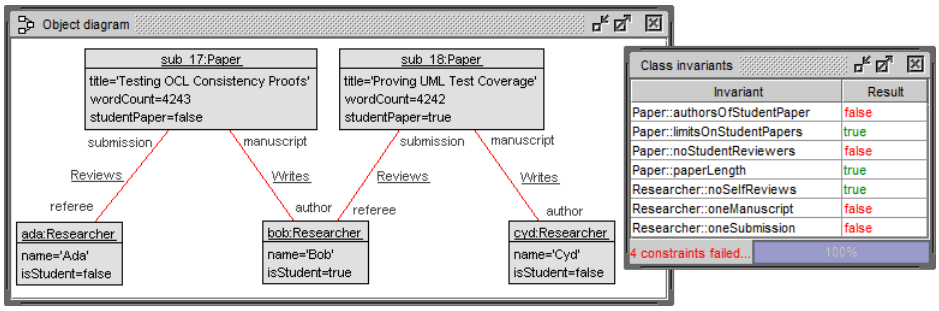


Fig. 1. Object Diagram for WR

The class diagram defines classes with attributes (and operations, not used in this example) and associations with roles and multiplicities which restrict the number of possible connected objects.

Objects: An OCL expression will often begin with an object literal or an object variable. For the system state represented in the object diagram, one can use the objects `ada,bob,cyd` of type `Researcher` and `sub_17,sub_18` of type `Paper`. Furthermore variables like `p:Paper` and `r:Researcher` can be employed.

Object navigation: Object navigation is realized by using role names from associations (or object-valued attributes, not occurring in this example) which are applied to objects or object collections. In the example, the following navigation expressions can be stated. The first line shows the OCL expression and the second line the evaluation result and the type of the expression and the result.

```
bob.manuscript
sub_17 : Paper

bob.manuscript.referee
Set{ada} : Set(Researcher)

cyd.manuscript.referee.manuscript.referee
Bag{ada} : Bag(Researcher)

sub_17.author->union(sub_17.referee)
Set{ada,bob} : Set(Researcher)
```

Collections: Collections can be employed in OCL to merge different elements into a single structure containing the elements. There are four collection kinds: sets, bags, sequences and ordered sets. Sets and ordered sets can contain an elements at most once, whereas bags and sequences may contain an element more than once. In sets and bags the element order is insignificant, whereas sequences and ordered sets are sensitive to the element order. For

a given class, the operation allInstances yields the set of current objects in the class.

```
Paper.allInstances
Set{sub_17,sub_18} : Set(Paper)

let P=Paper.allInstances in P.referee->union(P.author)
Bag{ada,bob,bob,cyd} : Bag(Researcher)

Paper.allInstances->sortedBy(p|p.wordCount)
Sequence{sub_18,sub_17} : Sequence(Paper)

Sequence{bob,ada,bob,cyd,ada}->asOrderedSet
OrderedSet{bob,ada,cyd} : OrderedSet(Researcher)
```

Collection operations: There is a number of collection operations which contribute essentially to the expressibility of OCL and which are applied with the arrow operator. Among further operations, collections can be tested on emptiness (`isEmpty`, `notEmpty`), the number of elements can be determined (`size`), the elements can be filtered (`select`, `reject`), elements can be mapped to a different item (`collect`) or can be sorted(`sortedBy`), set-theoretic operations may be employed (`union`, `intersection`), and collections can be converted into other collection kinds (`asSet`, `asBag`, `asSequence`, `asOrderdSet`). Above, we have already used the collection operations `union`, `sortedBy`, and `asOrderedSet`.

```
Paper.allInstances->isEmpty
false : Boolean

Researcher.allInstances->size
3 : Integer

Researcher.allInstances->select(r | not r.isStudent)
Set{ada,cyd} : Set(Researcher)

Paper.allInstances->reject(p | p.studentPaper)
Set{sub_17} : Set(Paper)

Paper.allInstances->collect(p | p.author.name)
Bag{'Bob','cyd'} : Bag(String)
```

Boolean-valued expressions: Because OCL is a constraint language, boolean expressions which formalize model properties play a central role. Apart from typical boolean connectives (`and`, `or`, `not`, `=`, `implies`, `xor`), universal and existential quantification are available (`forAll`, `exists`).

```
Researcher.allInstances->forAll(r,s | r<>s implies r.name<>s.name)
true : Boolean

Paper.allInstances->exists(p | p.studentPaper and p.wordCount>4242)
false : Boolean
```

Boolean expressions are frequently used to describe class invariants and operation pre- and postconditions.

3 Benchmark for UML and OCL Models (V-2013-04-05)

This section introduces the current benchmark models. We believe these four models offer a representative set of challenges and modeling language features.

3.1 CivilStatus (CS)

The simple class model in Fig. 2 with one class, one association, one operation defined with OCL, and two enumerations describes the civil status of persons. The six invariants require that (1) all attributes take defined values only, (2) the `name` attribute values follow a particular format, (3) the `name` attribute is unique among all persons, (4) a female person does not possess a wife, (5) a male person does not possess a husband,[1] and (6) a person has a spouse, if and only if the civil status attribute holds the value married.

Questions: (Questions are given names in order to reference them)

ConsistentInvariants: Is the model consistent? Is there at least one object diagram satisfying the UML class model and the explicit OCL invariants?

Independence: Are the invariants independent? Is there an invariant which is a consequence of the conditions imposed by the UML class model and the other invariants?

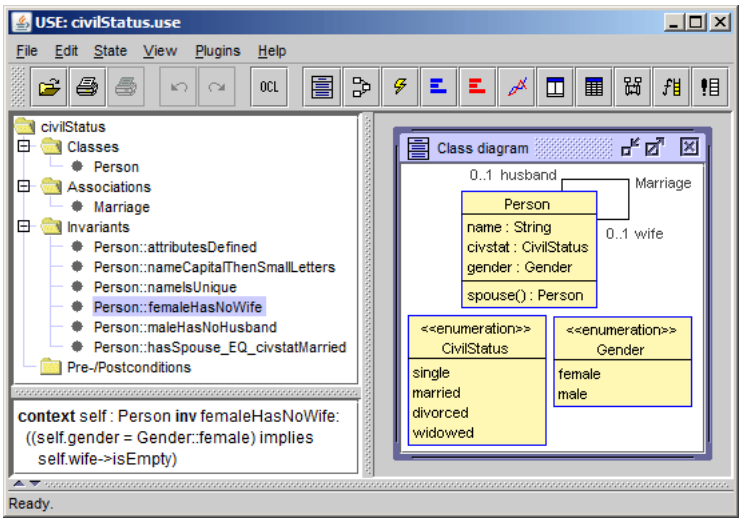
Consequences: Is it possible to show that a stated new property is a consequence of the given model? As a concrete question in terms of the model, one may ask: Is the model bigamy-free? Is it possible to have a person possessing both a wife and a husband?

LargeState: Is it possible to automatically build valid object diagrams in a parameterized way with a medium-sized number of objects, e.g. 10 to 30 objects and appropriate links, where all attributes take meaningful values and all links are established in a meaningful way? For example, a female person named `Ada` could be married in role `wife` to a male person named `Bob` occupying the `husband` role. These larger object diagrams are intended to explain the used model elements (like classes, attributes and associations) and the constraints upon them by non-trivial, meaningful examples to domain experts not necessarily familiar with formal modeling techniques.

3.2 WritesReviews (WR)

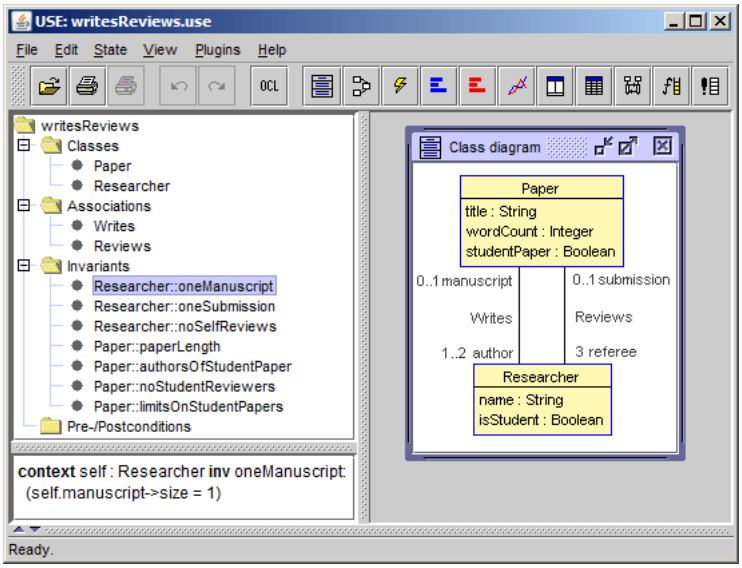
The class model in Fig. 3 has the classes `Paper` and `Researcher` and two associations in between. The first two invariants (1) `oneManuscript` and

[1] We are aware of the fact that we are only dealing with 'traditional marriages' with traditional roles, and not with more modern concepts like 'common law marriages'.




```
context Person
  inv attributesDefined: name<>null and civstat<>null and
      gender<>null
  inv nameCapitalThenSmallLetters:
      let small:Set(String)=
        Set{'a','b','c','d','e','f','g','h','i','j','k','l','m',
            'n','o','p','q','r','s','t','u','v','w','x','y','z'} in
      let capital:Set(String)=
        Set{'A','B','C','D','E','F','G','H','I','J','K','L','M',
            'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'} in
      capital->includes(name.substring(1,1)) and
      Set{2..name.size}->forAll(i |
        small->includes(name.substring(i,i)))
  inv nameIsUnique: Person.allInstances->forAll(self2|
      self<>self2 implies self.name<>self2.name)
  inv femaleHasNoWife: gender=#female implies wife->isEmpty
  inv maleHasNoHusband: gender=#male implies husband->isEmpty
  inv hasSpouse_EQ_civstatMarried: (spouse()<>null)=(civstat=#married)
```

Fig. 2. Class Diagram and Invariants for CS




```
context Researcher inv oneManuscript:
  self.manuscript->size=1
context Researcher inv oneSubmission:
  self.submission->size=1
context Researcher inv noSelfReviews:
  self.submission->excludes(self.manuscript)
context Paper inv paperLength:
  self.wordCount < 10000
context Paper inv authorsOfStudentPaper:
  self.studentPaper=self.author->exists(x | x.isStudent)
context Paper inv noStudentReviewers:
  self.referee->forAll(r | not r.isStudent)
context Paper inv limitsOnStudentPapers:
  Paper.allInstances->exists(p | p.studentPaper) and
  Paper.allInstances->select(p | p.studentPaper)->size < 5
```

Fig. 3. Class Diagram and Invariants for WR

(2) `oneSubmission` basically sharpen the `0..1` multiplicities to `1..1` multiplicities. In order to discuss alternative models, these two invariants will later be switched off for the construction of object diagrams. The next five invariants require that (3) a paper cannot be refereed by one of its authors, (4) the paper must obey a given length by restricting the attribute `wordCount`, (5) one of the authors of a `studentPaper` must be a student, (6) students are not allowed to

review papers, and (7) there must be at least one student paper, but no more than 4 student papers are allowed (assumed that there are `Paper` objects at all).

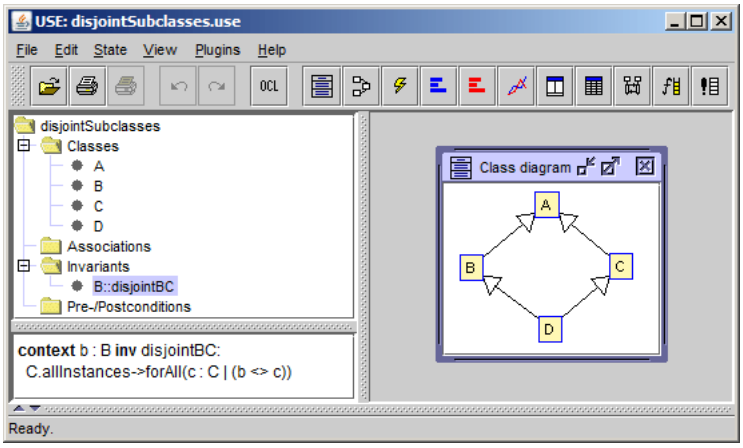
Questions:

InstantiateNonemptyClass: Can the model be instantiated with non-empty populations for all classes?

InstantiateNonemptyAssoc: Can the model be instantiated with non-empty populations for all classes and all associations?

InstantiateInvariantIgnore: Can the model be instantiated if the invariants oneManuscript and oneSubmission are ignored?

3.3 DisjointSubclasses (DS)




```
context b:B inv disjointBC: C.allInstances->forAll(c|b<>c)
```

Fig. 4. Class Diagram and Invariants for WR

The class model in Fig. 4 shows an example for multiple inheritance. Class `D` inherits from class `B` and class `C`. Class `B` and class `C` are required to be disjoint by the stated invariant.

Questions:

InstantiateDisjointInheritance: Can all classes be populated? Is it possible to build objects for class `D`?

InstantiateMultipleInheritance: Can class `D` be populated if the constraint `disjointBC` is ignored?

A light extension of this benchmark model might add attributes `a, b, c`, and `d` to all classes having the type Integer. A hypothetical example constraint for class `D` might then require `self.d=2*self.a`. It would be interesting to see whether a tool syntactically allows to reference the attribute `a` from class `D`.

3.4 ObjectsAsIntegers (OAI)

The class model in Fig. 5 introduces one abstract superclass `Int` and three concrete subclasses `Neg`, `Zero`, and `Pos`. Objects of class `Zero` are intended to represent the integer `0`, objects of class `Neg` are intended to represent a negative integer in the shape of a normal form `(...((0-1)-1)...)-1`, and objects of class `Pos` are intended to represent a positive integer in the shape of a normal form `(...((0+1)+1)...)+1`. The abstract class `Int` possesses one association which is inherited to the subclasses. The recursively defined operations `predPlus()` and `succPlus()` in class `Int` compute the (non-reflexive) transitive closure of the association ends `pred` and `succ`. The operations `predPlusOnSet(...)` and `succPlusOnSet(...)` are internal helper operations not intended to be called from outside the class. The invariants require that (1) the `PredSucc` links are acyclic, (2) a `Zero` object is not linked to another `Zero` object, (3) a `Zero` object is not linked to both a `Neg` and a `Pos` object, (4) a `Neg` object is not linked to a `Pos` object, and (5) a `Neg` object is linked to a `Zero` object by employing the `succ` association end. `Pos` objects are restricted analogously to `Neg` objects.
The upper object diagram in Fig. 6 shows a valid system state for OAI, the lower one an invalid system state with some invariants violated. For example, invariant zeroNotLinkedToNegAndPos is violated in the left connected component of the lower object diagram. The upper object diagram displays the object representation of the integer sequence $-2, -1, 0, +1, +2, +1$. Every connected component of the object diagram corresponds to an integer. The lower object diagram has two connected components, where both components taken in isolation already violate the model invariants, but obey the class diagram multiplicities.

Questions:

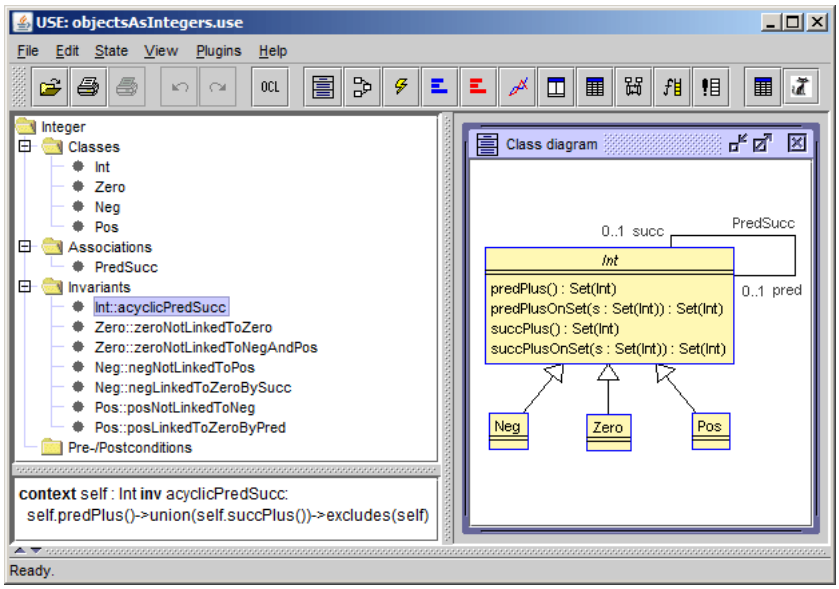
ObjectRepresentsInteger: Is it true that any connected component of a valid object diagram for the model either corresponds to the term *zero* or to a term of the form $succ^n(zero)$ with $n > 0$ or to a term of the form $pred^n(zero)$?

IntegerRepresentsObject: Is it true that any term of the form *zero* or of the form $succ^n(zero)$ or of the form $pred^n(zero)$ corresponds to a valid object diagram for the model?

A slight extension of the current benchmark might ask a tool to find a minimal constraint subset (or all constraint subsets) such that the same invariants are implied as above.

4 Handling the Benchmark in USE

USE is a tool that allows modelers to check and test UML and OCL models. It allows model validation and verification based of enumeration and SAT-based techniques. USE allows the developer to construct object diagrams with a specialized language called ASSL (A Snapshot Sequence Language). All details can be traced from the provided additional material [14].




```
context Int
  inv acyclicPredSucc:
    predPlus()->union(succPlus())->excludes(self)
context Zero
  inv zeroNotLinkedToZero:
    not predPlus()->union(succPlus())->exists(i|
      i.oclIsTypeOf(Zero))
  inv zeroNotLinkedToNegAndPos:
    not predPlus()->union(succPlus())->exists(n,p|
      n.oclIsTypeOf(Neg) and p.oclIsTypeOf(Pos))
context Neg
  inv negNotLinkedToPos:
    not predPlus()->union(succPlus())->exists(p|
      p.oclIsTypeOf(Pos))
  inv negLinkedToZeroBySucc:
    succPlus()->exists(z|z.oclIsTypeOf(Zero))
context Pos
  inv posNotLinkedToNeg:
    not predPlus()->union(succPlus())->exists(n|
      n.oclIsTypeOf(Neg))
  inv posLinkedToZeroByPred:
    predPlus()->exists(z|z.oclIsTypeOf(Zero))
```

Fig. 5. Class Diagram and Invariants for WR

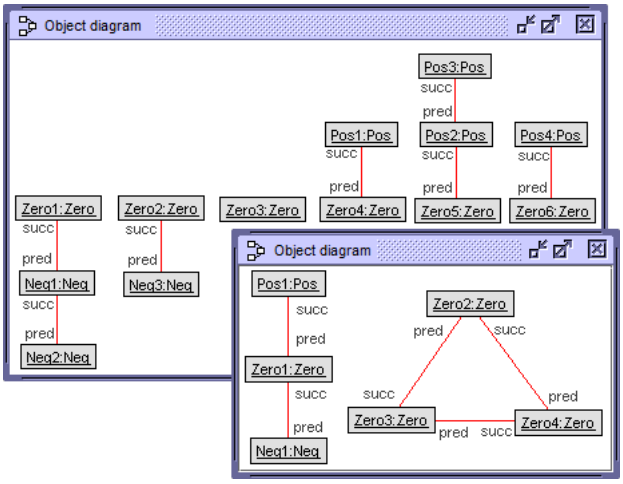


Fig. 6. Example Object Diagrams for OAI

4.1 CivilStatus (CS)

For the CS example, there are three procedures which aim to construct object diagrams: (1) `generateWorld(numFemale:Integer, numMale:Integer, numMarriage:Integer)` can build object diagrams satisfying all constraints and object diagrams violating particular constraints, (2) `largerWorld(numFemale:Integer, numMale:Integer, numMarriage:Integer)` can build larger object diagrams (up to 26 female persons and 26 male persons with at most 26 marriages) satisfying all constraints, and (3) `attemptBigamy()` tries to construct an object diagram including bigamy.

ConsistentInvariants: The consistency of the invariants in combination with the class diagram model inherent constraints is shown by a calling `generateWorld(1,1,1)` with all invariants activated.

Independence: The independence of the six invariants is shown by six calls to `generateWorld(1,1,1)` where before each single call exactly one invariant is negated and the other invariants are activated.

Consequences: The fact that the model is bigamy-free is demonstrated by a call to `attemptBigamy()`. In that procedure a large number of possible object diagrams with three persons and all possible assignments of roles and attribute values is considered and checked. No object diagram showing bigamy is found.

LargeState: A larger object diagram is constructed by the call `largerWorld(5,7,4)` which constructs a system state with five female persons, seven male persons, and four marriages.

4.2 WritesReviews (WR)

For the WR example, one ASSL procedure is provided: `generateWorld(numPap:Integer, numRes:Integer, fillAttr:Boolean)`. The parameters determine the number of papers, the number of researchers, and whether the object attributes should be filled with actual values.

InstantiateNonemptyClass,InstantiateNonemptyAssoc: A call to `generateWorld(4,4,false)` yields the answer that no valid object diagram can be constructed. The attribute values are not taken into account. This shows that the multiplicities cannot be satisfied in the considered search space.

InstantiateInvariantIgnore: If the two invariants `oneManuscript` and `oneSubmission` are deactivated, a valid object diagram can be constructed by calling `generateWorld(1,4,true)`. The attributes take meaningful values in the constructed object diagram.

4.3 DisjointSubclasses (DS)

For the model DS the ASSL procedure `generateWorld(noA:Integer, noB:Integer, noC:Integer, noD:Integer)` is employed.

InstantiateDisjointInheritance: A call to `generateWorld(1,1,1,1)` with invariant `disjointBC` activated does not yield a valid object diagram.

InstantiateMultipleInheritance: Calling `generateWorld(1,1,1,1)` with invariant `disjointBC` deactivated does return a valid object diagram, naturally with all objects of class `D` being also objects in class `B` and class `C`.

4.4 ObjectsAsIntegers (OAI)

For the model OAI, the ASSL procedure `generateWorld(intNum:Integer, predSuccNum:Integer)` constructs an object diagram with `intNum` objects for class `Int` and `predSuccNum` links between these `Int` objects. The constructed object diagram does not necessarily obey the invariants, but the results can be looked at being test cases for human inspection.

ObjectRepresentsInteger: We have generated various test cases with the above ASSL procedure and found no counter examples for the stated question resp. claim. However, we do not have solid formal arguments that the claim is valid.

IntegerRepresentsObject: One can formulate an ASSL procedure `generateInt(i:Integer)` that constructs the appropriate object diagram of class `Int`: Exactly one `Zero` object will be created; if $i < 0$, the respective number of `Neg` objects will be created and linked to the single `Zero` object in a correct way; if $i > 0$, the procedure will create `Pos` objects, analogously.

5 Handling the Benchmark in EMFtoCSP

Consistency checking and model instantiation are performed transparently in EMFtoCSP by internally creating a constraint satisfaction problem (CSP) that is satisfiable iff the model plus the OCL constraints satisfy the given correctness property. The user has to specify ranges for the class and association extents and for the attribute domains. All details are provided in the additional material [14].

5.1 CivilStatus (CS)

For running the CivilStatus checks, the range `0..5` was used for the Person class and the range `0..25` for the Marriage association. The string length of the name attribute was set to `0..10` (EMFtoCSP supports the String datatype and its operations [6]). We omitted the invariants `attributesDefined` and `nameCapitalThenSmallLetters`. The first holds implicitly because of the search space configuration, the second currently cannot be parsed by the EMFtoCSP front-end.

ConsistentInvariants: The consistency of the invariants in combination with the class diagram inherent constraints is shown by running EMFtoCSP with the described search bounds, selecting 'weak satisfiability' as the verification property, yielding a valid object diagram as proof.

Independence: The independence of the four considered invariants is shown by verifying four modified versions of `CivilStatus`, where one of the invariants is negated in each run. Using the described search bounds, each run yields an instance that is valid w.r.t. the modified version.

Consequences: The fact that the model is bigamy-free is demonstrated by amending `CivilStatus` with a constraint `notIsBigamyFree` that requires an instance with bigamy and then showing the unsatisfiability of that model using the described search bounds.

LargeState: Adding an invariant `niceInstance` to `CivilStatus` restricts the names to a meaningful set and the gender to be consistent with the name (e.g., `name = 'Ada' implies gender = 1`). We set the search bounds to 7 persons and 3 marriages and EMFtoCSP yields a valid instance.

5.2 WritesReviews (WR)

For running the WritesReviews checks, `0..5` was used for both classes and `0..25` for both associations, the string lengths were set to `0..10` and the range of wordCount to `0..10000`.

InstantiateNonemptyClass: Checking 'weak consistency' shows that the model and the constraints are unsatisfiable within the above search bounds.

InstantiateNonemptyAssoc: Checking 'strong consistency' shows that the model and the constraints are unsatisfiable within the above search bounds.

InstantiateInvariantIgnore: Checking 'weak consistency' on a modified version of WritesReviews, in which both `oneManuscript` and `oneSubmission` are commented out, yields a satisfying instance.

5.3 DisjointSubclasses (DS)

The front-end of EMFtoCSP does currently not support multiple inheritance, although the UML/OCL constraint library that is used in the background provides all necessary predicates.

5.4 ObjectsAsIntegers (OAI)

EMFtoCSP does currently not solve models with recursive operations.

6 Discussion

The benchmark as presented in this paper is a first step in the definition of a complete set of UML and OCL models that the modeling community could accept as valid. More importantly, the community could start to compare and to improve current MDE approaches and tools, similar to what other communities in Software Engineering are already doing.

The models that we have discussed give a taste of the difficulties that anybody working on a new OCL analysis technique should consider. Nevertheless, our long term goal is the complete specification of a full benchmark model suite covering all known challenging verification and validation scenarios. The need for such a benchmark was one of the outcomes of the last OCL Workshop. However, the notion 'challenging scenario' is not universal and debatable, in the sense that depending on the formalism used by a given tool a scenario may be easy or extremely demanding. With proposing this benchmark and its hopefully coming evolution, we want developers to evaluate the existing approaches, realize which are the strengths and drawbacks of each one, and choose a tool or an approach according to their specific needs. Speaking generally, for an OCL analysis tool benchmark there are challenges in two dimensions: (a) challenges related to the complexity of OCL (i.e., the complete and accurate handling of OCL) and (b) challenges related to the computational complexity of the underlying problem. Both should be treated in the benchmark.

Based on our own experience we believe that at least the following scenarios should be covered by models in the benchmark:

1. Mostly local constraints: models with many constraints but where all constraints are local, i.e., they only involve a single class or a cluster of closely related classes.
2. Mostly global constraints: models with many constraints but where all constraints are global, i.e., they usually involve a large percentage of the classes in the model, e.g. a constraint forcing all classes in the model to have the same number of instances.
3. Models with tractable constraints, i.e., constraints that can be solved 'trivially' by simple propagation steps.

4. Models with hard, non-tractable constraints, e.g., representations of NP-hard problems.
5. Highly symmetric problems, i.e., that require symmetry breaking to efficiently detect unsatisfiability.
6. Intensive use of Integer arithmetic allowing large ranges for integer values and employing heavily arithmetic and operation like inequality.
7. Intensive use of Real arithmetic.
8. Intensive use of String values and operations on strings. So far, String attributes are mostly ignored [6] or simply regarded as integers which prohibits the verification of OCL expressions including String operations other than equality and inequality.
9. Many redundant constraints: is the approach able to detect the redundancies and benefit from them to speed up the evaluation?
10. Sparse models: instances with comparably few links offer optimization opportunities that could be exploited by tools.
11. Support for recursive operations, e.g. in form of fixpoint detection or static unfolding.
12. Intensive use of the 'full' semantic of OCL (like the undefined value or collection semantics); this poses a challenge for the lifting to two-valued logics.
13. Problems that have large instances (with many objects).

Alternative models for each of these scenarios should be part of the benchmark to cover different goals in the evaluation. For instance, when evaluating the correctness of the results provided by a given tool we should execute satisfiable and unsatisfiable versions of each model and when evaluating its performance and scalability we should feed the tool increasingly larger versions of the same model.

We hope that as soon as these benchmarks become available the interested community (from developers of tools to tool users) will start applying them on a variety of tools and approaches, which will allow us to clarify and better understand the differences among the plethora of approaches and tools for OCL solving that are now available. However, we have to keep in mind that as discussed in the previous section, the results of the benchmark have to be interpreted with care. A bad score of a tool for a given model can be attributed to different reasons, from a simple syntax problem (maybe the tool does not support one of the OCL operations used in an expression even if this operation is not a key part of the benchmark) to a limitation of the tool or a limitation of the underlying tool formalism. This difference is important too. In a further step, we want to able not only to compare the tools themselves but to use the benchmark to study the limits of frequently used provers, solvers or finders when applied to the OCL realm.

7 Potential Tools to be Considered and Related Work

We have conducted the benchmark with the tools USE [15] and UMLtoCSP resp. EMFtoCSP [7, 16]. Other validation and verification tools for OCL which

are possible candidates to be examined under the benchmark are UML2Alloy [2], the planned USE extension arising from [19], the ITP/OCL tool [12], mOdCL [21] and MOMENT-OCL[4]. Furthermore, the OCL tools OCLE, ROCLET, and OCTOPUS would be benchmark candidates, but the projects seem to be inactive since years (http://lci.cs.ubbcluj.ro/ocle/, http://www.roclet.org [dead link], http://octopus.sourceforge.net/).

There is variety of other validation approaches for OCL based on SMT [13, 25] or SAT [23]. Description logics has been used as a basis for OCL expressions and constraints [20, 8] and for querying UML class diagram models [9].

On the prover side HOL-OCL combines Isabelle with UML and OCL [5], the Key project attempted theorem proving in connection with the commercial UML tool Together [3], and encoding of OCL into PVS was studied in [18]. We would expect that completeness problems as appearing in OAI ('Does the set of all object diagrams of the model correspond to the integers?') could be handled more adequately in proof-oriented approaches. An application of a combination of proof and test techniques in connection with OCL was described in a case study [22]. Elements from that work might be considered for future versions of the benchmark.

Testing approaches aiming at tool support were put forward in [10, 1]. The benchmark might also be applicable for code generation as in Dresden OCL [17] or MDT/OCL [24]. Last, the feature model for model development environments [11] could be connected to the benchmark.

8 Conclusion

The approach proposed here is only first step towards a more complete benchmark. We concentrated on four models with eleven questions and claims. We already have a sufficient coverage of OCL and questions, but more models and items are needed. We would be happy if other groups would contribute. We think more elaboration on complexity questions should be done in order to answer, for example, questions attacking the extent to which a tool can produce and deal with larger states or assert properties in larger states. A classification of questions and items suitable for proof techniques or for test techniques seems to be needed as well.

Acknowledgments

The comments of the referees have helped to improve the paper. Many thanks to the other developers of USE and EMFtoCSP. Without their work this contribution would not have been possible.

References

1. Bernhard K. Aichernig and Percy Antonio Pari Salas. Test Case Generation by OCL Mutation and Constraint Solving. In *QSIC*, pages 64–71. IEEE Computer Society, 2005.
2. Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On Challenges of Model Transformation from UML to Alloy. *Software and System Modeling*, 9(1):69–86, 2010.
3. Bernhard Beckert, Martin Giese, Reiner Hähnle, Vladimir Klebanov, Philipp Rümmer, Steffen Schlager, and Peter H. Schmitt. The KeY system 1.0 (Deduction Component). In Frank Pfenning, editor, *CADE*, LNCS 4603, pages 379–384. Springer, 2007.
4. Artur Boronat and José Meseguer. Algebraic Semantics of OCL-Constrained Metamodel Specifications. In Manuel Oriol and Bertrand Meyer, editors, *TOOLS (47)*, LNBIP 33, pages 96–115. Springer, 2009.
5. Achim D. Brucker and Burkhart Wolff. HOL-OCL: A Formal Proof Environment for UML/OCL. In José Luiz Fiadeiro and Paola Inverardi, editors, *FASE*, LNCS 4961, pages 97–100. Springer, 2008.
6. Fabian Büttner and Jordi Cabot. Lightweight String Reasoning for OCL. In Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle, and Dimitrios S. Kolovos, editors, *ECMFA*, LNCS 7349, pages 244–258. Springer, 2012.
7. Jordi Cabot, Robert Clarisó, and Daniel Riera. UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models using Constraint Programming. In R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer, editors, *ASE*, pages 547–548. ACM, 2007.
8. Marco Cadoli, Diego Calvanese, Giuseppe De Giacomo, and Toni Mancini. Finite Model Reasoning on UML Class Diagrams Via Constraint Programming. In Roberto Basili and Maria Teresa Pazienza, editors, *AI*IA*, LNCS 4733, pages 36–47. Springer, 2007.
9. Andrea Calì, Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Querying UML Class Diagrams. In Lars Birkedal, editor, *FoSSaCS*, LNCS 7213, pages 1–25. Springer, 2012.
10. Kalou Cabrera Castillos, Frédéric Dadeau, Jacques Julliand, and Safouan Taha. Measuring Test Properties Coverage for Evaluating UML/OCL Model-Based Tests. In Burkhart Wolff and Fatiha Zaïdi, editors, *ICTSS*, LNCS 7019, pages 32–47. Springer, 2011.
11. Joanna Dobroslawa Chimiak-Opoka and Birgit Demuth. A Feature Model for an IDE4OCL. *ECEASST*, 36, 2010.
12. Manuel Clavel and Marina Egea. ITP/OCL: A Rewriting-Based Validation Tool for UML+OCL Static Class Diagrams. In Michael Johnson and Varmo Vene, editors, *AMAST*, volume 4019 of *LNCS 4019*, pages 368–373. Springer, 2006.
13. Manuel Clavel, Marina Egea, and Miguel Angel García de Dios. Checking Unsatisfiability for OCL Constraints. *Electronic Communications of the EASST*, 24:1–13, 2009.
14. Martin Gogolla, Fabian Büttner, and Jordi Cabot. Initiating a Benchmark for UML and OCL Analysis Tools: Additional Material. Technical report, University of Bremen, 2013. http://www.db.informatik.uni-bremen.de/publications/intern/GBC2013addon.pdf.
15. Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34, 2007.

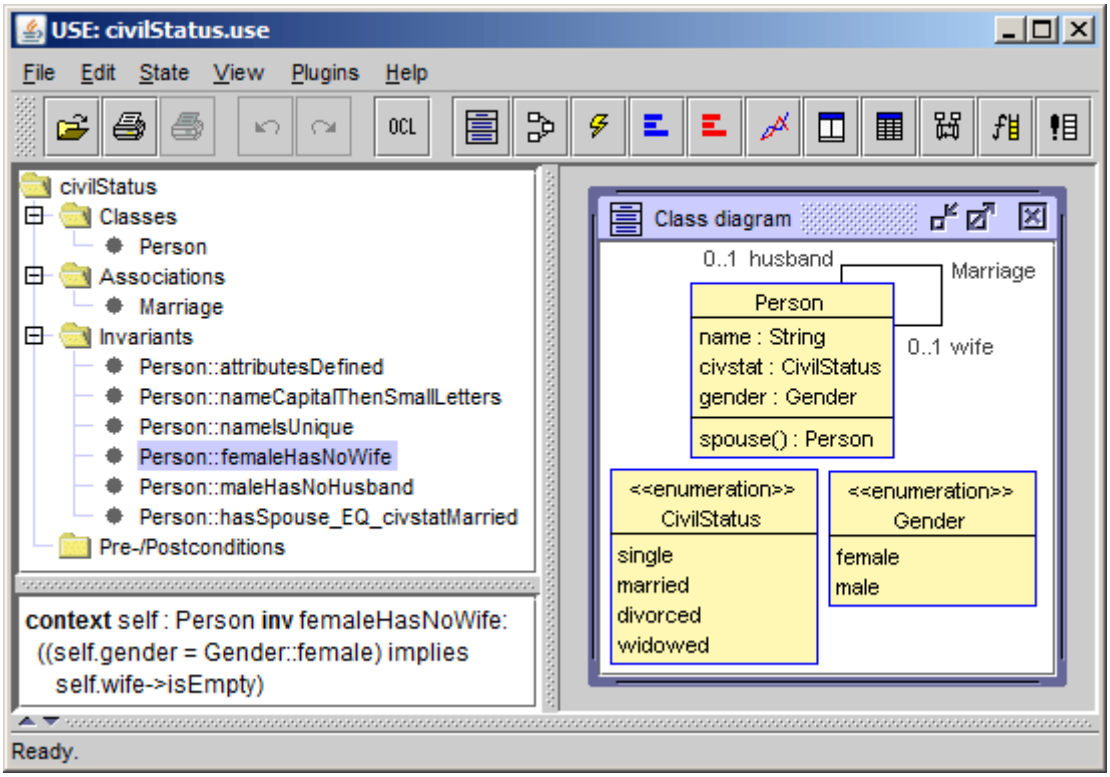
16. Carlos A. Gonzalez, Fabian Büttner, Robert Clariso, and Jordi Cabot. EMFtoCSP: A Tool for the Lightweight Verification of EMF Models. In Stefania Gnesi, Stefan Gruner, Nico Plat, and Bernhard Rumpe, editors, *Proc. ICSE 2012 Workshop Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, 2012.
17. Heinrich Hußmann, Birgit Demuth, and Frank Finger. Modular Architecture for a Toolset Supporting OCL. *Sci. Comput. Program.*, 44(1):51–69, 2002.
18. Marcel Kyas, Harald Fecher, Frank S. de Boer, Joost Jacob, Jozef Hooman, Mark van der Zwaag, Tamarah Arons, and Hillel Kugler. Formalizing UML Models and OCL Constraints in PVS. *Electr. Notes Theor. Comput. Sci.*, 115:39–47, 2005.
19. Azzam Maraee and Mira Balaban. Efficient Reasoning About Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets. In David H. Akehurst, Régis Vogel, and Richard F. Paige, editors, *ECMDA-FA*, LNCS 4530, pages 17–31. Springer, 2007.
20. Anna Queralt, Alessandro Artale, Diego Calvanese, and Ernest Teniente. OCL-Lite: Finite Reasoning on UML/OCL Conceptual Schemas. *Data Knowl. Eng.*, 73:1–22, 2012.
21. Manuel Roldán and Francisco Durán. Dynamic Validation of OCL Constraints with mOdCL. *ECEASST*, 44, 2011.
22. Miriam Schleipen. A Concept for Conformance Testing of AutomationML Models by Means of Formal Proof using OCL. In *ETFA*, pages 1–5. IEEE, 2010.
23. Robert Wille, Mathias Soeken, and Rolf Drechsler. Debugging of Inconsistent UML/OCL Models. In Wolfgang Rosenstiel and Lothar Thiele, editors, *DATE*, pages 1078–1083. IEEE, 2012.
24. Edward D. Willink. Re-Engineering Eclipse MDT/OCL for Xtext. *ECEASST*, 36, 2010.
25. Kenro Yatake and Toshiaki Aoki. SMT-Based Enumeration of Object Graphs from UML Class Diagrams. *ACM SIGSOFT Software Engineering Notes*, 37(4):1–8, 2012.

Initiating a Benchmark for UML and OCL Analysis Tools: Additional Material

Martin Gogolla, Fabian Büttner, Jordi Cabot
University of Bremen, INRIA / Ecole des Mines de Nantes

1. Current Benchmark Models (Version 2013-02-01)

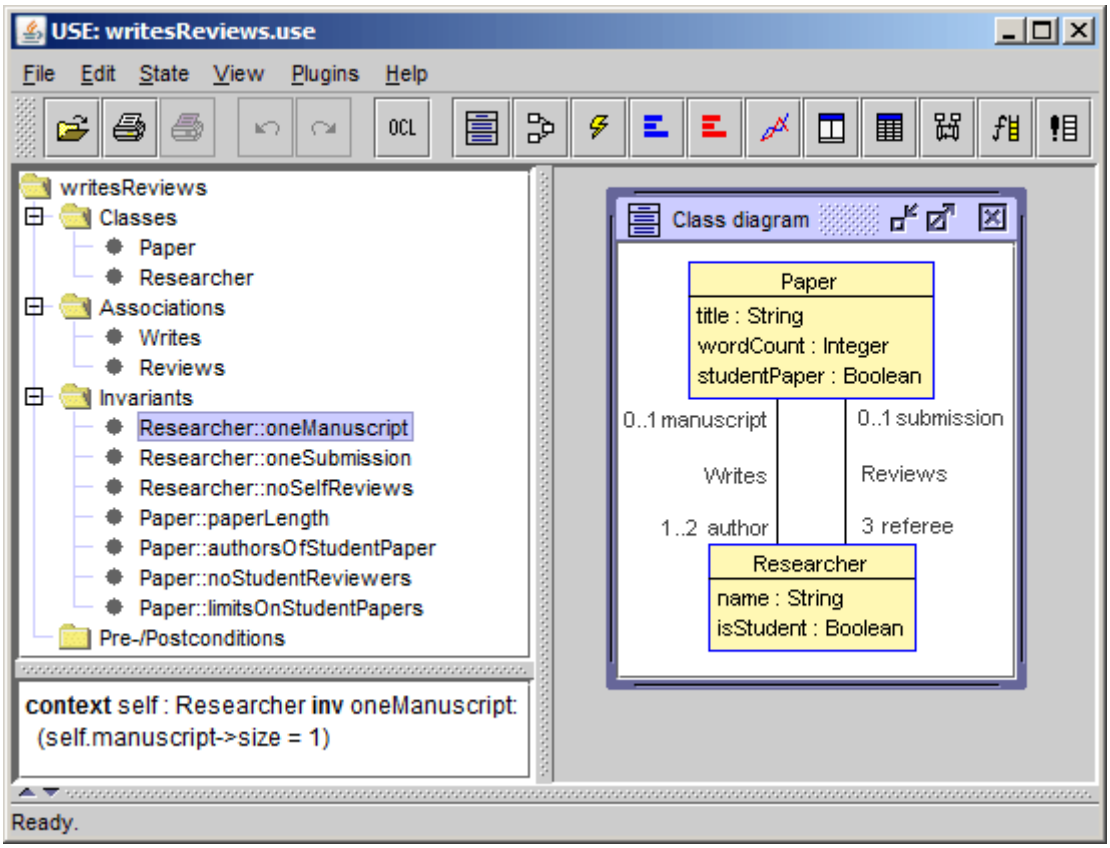
1.1 CivilStatus (CS)




```
model civilStatus

enum CivilStatus {single, married, divorced, widowed}
enum Gender {female, male}

class Person
attributes
  name:String
  civstat:CivilStatus
  gender:Gender
operations
  spouse():Person=
    if gender=#female then husband else
      if gender=#male then wife else null endif endif
constraints
  inv attributesDefined: name<>null and civstat<>null and
      gender<>null
  inv nameCapitalThenSmallLetters:
      let small:Set(String)=
        Set{'a','b','c','d','e','f','g','h','i','j','k','l','m',
            'n','o','p','q','r','s','t','u','v','w','x','y','z'} in
      let capital:Set(String)=
        Set{'A','B','C','D','E','F','G','H','I','J','K','L','M',
            'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'} in
      capital->includes(name.substring(1,1)) and
      Set{2..name.size}->forAll(i |
        small->includes(name.substring(i,i)))
```

```
  inv nameIsUnique: Person.allInstances->forAll(self2|
      self<>self2 implies self.name<>self2.name)
  inv femaleHasNoWife: gender=#female implies wife->isEmpty
  inv maleHasNoHusband: gender=#male implies husband->isEmpty
  inv hasSpouse_EQ_civstatMarried: (spouse()<>null)=(civstat=#married)
end

association Marriage between
  Person [0..1] role wife
  Person [0..1] role husband
end
```

1.2 WritesReviews (WR)

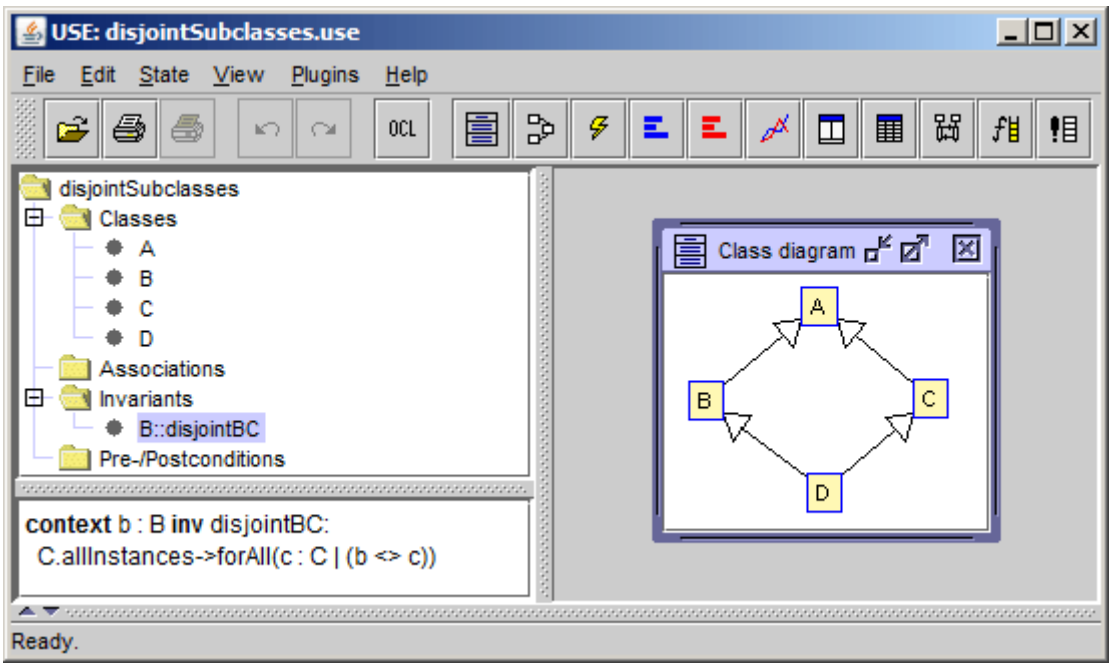



```
model writesReviews

class Paper
attributes
  title:String
  wordCount:Integer
  studentPaper:Boolean
end

class Researcher
attributes
  name:String
  isStudent:Boolean
end
```

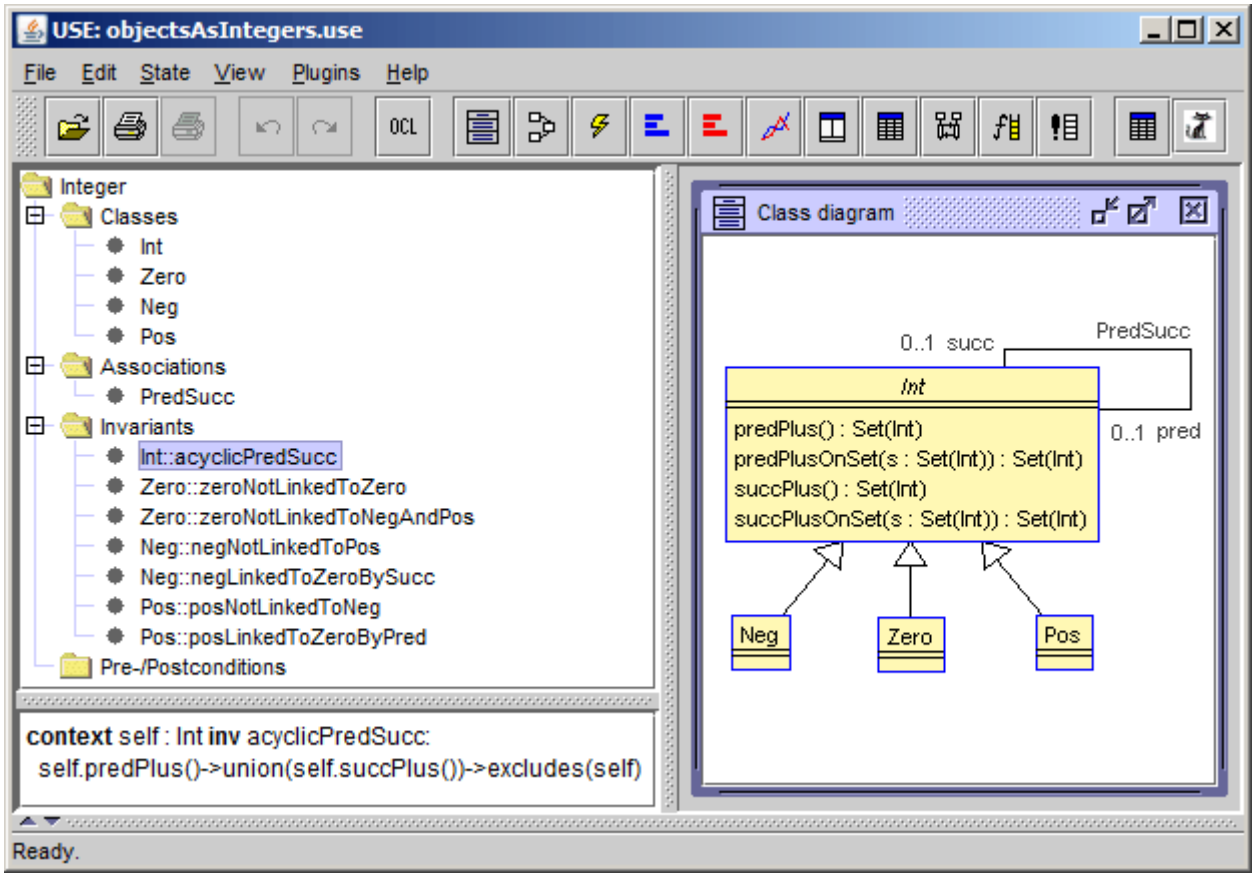
```
association Writes between
  Researcher[1..2] role author
  Paper[0..1] role manuscript
end

association Reviews between
  Researcher[3] role referee
  Paper[0..1] role submission
end

constraints

context Researcher inv oneManuscript:
  self.manuscript->size=1

context Researcher inv oneSubmission:
  self.submission->size=1

context Researcher inv noSelfReviews:
  self.submission->excludes(self.manuscript)

context Paper inv paperLength:
  self.wordCount < 10000

context Paper inv authorsOfStudentPaper:
  self.studentPaper=self.author->exists(x | x.isStudent)

context Paper inv noStudentReviewers:
  self.referee->forAll(r | not r.isStudent)

context Paper inv limitsOnStudentPapers:
  Paper.allInstances->exists(p | p.studentPaper) and
  Paper.allInstances->select(p | p.studentPaper)->size < 5
```

1.3 DisjointSubclasses (DS)

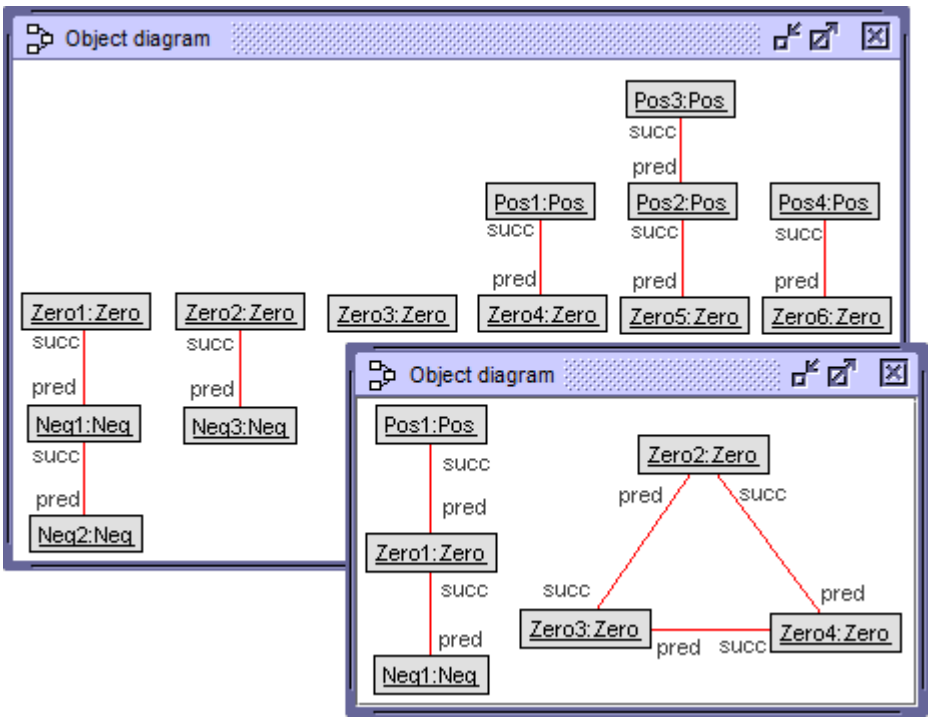
```
model disjointSubclasses

class A
end

class B < A
end

class C < A
end

class D < B,C
end

constraints

context b:B inv disjointBC: C.allInstances->forAll(c|b<>c)
```



1.4 ObjectsAsIntegers (OAI)



```
model ObjectsAsInteger

abstract class Int
operations
predPlus():Set(Int)=
  predPlusOnSet(Set{pred}->excluding(null))
predPlusOnSet(s:Set(Int)):Set(Int)=
  let oneStep=s.pred->excluding(null)->asSet in
  if oneStep->exists(i|s->excludes(i))
    then predPlusOnSet(s->union(oneStep)) else s endif
succPlus():Set(Int)=
  succPlusOnSet(Set{succ}->excluding(null))
succPlusOnSet(s:Set(Int)):Set(Int)=
  let oneStep=s.succ->excluding(null)->asSet in
  if oneStep->exists(i|s->excludes(i))
    then succPlusOnSet(s->union(oneStep)) else s endif
constraints
  inv acyclicPredSucc:
    predPlus()->union(succPlus())->excludes(self)
end

association PredSucc between
  Int[0..1] role pred
  Int[0..1] role succ
end

class Zero < Int
constraints
  inv zeroNotLinkedToZero:
    not predPlus()->union(succPlus())->exists(i|
      i.oclIsTypeOf(Zero))
  inv zeroNotLinkedToNegAndPos:
    not predPlus()->union(succPlus())->exists(n,p|
      n.oclIsTypeOf(Neg) and p.oclIsTypeOf(Pos))
end

class Neg < Int
constraints
  inv negNotLinkedToPos:
    not predPlus()->union(succPlus())->exists(p|
      p.oclIsTypeOf(Pos))
  inv negLinkedToZeroBySucc:
    succPlus()->exists(z|z.oclIsTypeOf(Zero))
end

class Pos < Int
constraints
  inv posNotLinkedToNeg:
    not predPlus()->union(succPlus())->exists(n|
      n.oclIsTypeOf(Neg))
  inv posLinkedToZeroByPred:
    predPlus()->exists(z|z.oclIsTypeOf(Zero))
end
```



2. Details of Benchmark Analysis with USE

2.1 CivilStatus (CS)

```
--------------------------------------------------------------------------

procedure generateWorld(numFemale:Integer, numMale:Integer,
                        numMarriage:Integer)
var females:Sequence(Person), males:Sequence(Person),
    f:Person, m:Person;
begin
females:=CreateN(Person,[numFemale]);
males:=CreateN(Person,[numMale]);
for i:Integer in [Sequence{1..numFemale}] begin
  [females->at(i)].name:=
    Try([Sequence{'Ada','Bel','ada','bel','Sam','Vic',null}]);
  [females->at(i)].civstat:=
    Try([Sequence{#single,#married,#divorced,#widowed}]);
  [females->at(i)].gender:=Try([Sequence{#female,#male}]); end;
for i:Integer in [Sequence{1..numMale}] begin
  [males->at(i)].name:=
    Try([Sequence{'Ali','Bob','ali','bob','Sam','Vic',null}]);
  [males->at(i)].civstat:=
    Try([Sequence{#single,#married,#divorced,#widowed,null}]);
  [males->at(i)].gender:=Try([Sequence{#female,#male,null}]); end;
for i:Integer in [Sequence{1..numMarriage}] begin
  f:=Try([females->select(p|p.husband->isEmpty)]);
  m:=Try([males->select(p|p.wife->isEmpty)]);
  Insert(Marriage,[f],[m]); end;
end;

--------------------------------------------------------------------------
```

```
procedure largerWorld(numFemale:Integer, numMale:Integer,
numMarriage:Integer)
-- numMarriage<=numFemale<=26, numMarriage<=numMale<=26
var females:Sequence(Person), males:Sequence(Person),
    f:Person, m:Person;
begin
females:=CreateN(Person,[numFemale]);
males:=CreateN(Person,[numMale]);
for i:Integer in [Sequence{1..numFemale}] begin
  [females->at(i)].name:=Any([Sequence{'Ada','Bel','Cam','Day',
    'Eva','Flo','Gen','Hao','Ina','Jen','Kia','Lan','Mae','Nan','Oki',
    'Pam','Quao','Rae','Sen','Tip','Una','Vea','Wan','Xia','Yan','Zoe'}
    ->reject(n|Person.allInstances->exists(p|p.name=n))]);
  [females->at(i)].civstat:=[#single];
  [females->at(i)].gender:=[#female]; end;
for i:Integer in [Sequence{1..numMale}] begin
  [males->at(i)].name:=Any([Sequence{'Ali','Bob','Cyd','Dan',
    'Eli','Fox','Gil','Hal','Ike','Jan','Kim','Leo','Max','Nam','Ole',
    'Pat','Quin','Rex','Sam','Tom','Ulf','Vic','Wei','Xan','Yul','Zan'}
    ->reject(n|Person.allInstances->exists(p|p.name=n))]);
  [males->at(i)].civstat:=[#single];
  [males->at(i)].gender:=[#male]; end;
for i:Integer in [Sequence{1..numMarriage}] begin
  f:=Any([females->reject(p|p.husband.isDefined)]);
  m:=Any([males->reject(p|p.wife.isDefined)]);
  [f].civstat:=[#married]; [m].civstat:=[#married];
  Insert(Marriage,[f],[m]); end;
end;

-------------------------------------------------------------------------

procedure attemptBigamy()
var p:Person, w:Person, h:Person, thePersons:Sequence(Person);
--    w -wife---husb-   p   -wife---husb-   h
begin
thePersons:=CreateN(Person,[3]);
for i:Integer in [Sequence{1..3}] begin
  [thePersons->at(i)].name:=Try([Sequence{'Alex','Bobby','Chris'}]);
  [thePersons->at(i)].civstat:=
    Try([Sequence{#single,#married,#divorced,#widowed}]);
  [thePersons->at(i)].gender:=Try([Sequence{#female,#male}]); end;
p:=Try([thePersons]);
w:=Try([thePersons->excluding(p)]);
h:=Try([thePersons->excluding(p)->excluding(w)]);
Insert(Marriage,[w],[p]); Insert(Marriage,[p],[h]);
end;

-------------------------------------------------------------------------
```

```
use> open civilStatus.use

use> gen start civilStatus.assl generateWorld(1,1,1)
     Progress of first Try in ASSL-Procedure (7 combinations):
     |-----------------------------------------------|
     #######
use> gen result
     Random number generator was initialized with 1119.
     Checked 68 snapshots in 0,219s (311 snapshots/s).
     Checked 215 times in 0,219s (982 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
     Barriers blocked 147 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Person('Person1')
     !new Person('Person2')
     !@Person1.name := 'Ada'
     !@Person1.civstat := CivilStatus::married
     !@Person1.gender := Gender::female
     !@Person2.name := 'Ali'
     !@Person2.civstat := CivilStatus::married
     !@Person2.gender := Gender::male
     !insert (@Person1,@Person2) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Person::attributesDefined': OK.
     checking invariant (2) `Person::femaleHasNoWife': OK.
     checking invariant (3) `Person::hasSpouse_EQ_civstatMarried': OK.
     checking invariant (4) `Person::maleHasNoHusband': OK.
     checking invariant (5) `Person::nameCapitalThenSmallLetters': OK.
     checking invariant (6) `Person::nameIsUnique': OK.
     checked 6 invariants in 0.015s, 0 failures.

use> reset

use> gen flags Person::attributesDefined           +n
use> gen flags Person::femaleHasNoWife             -d
use> gen flags Person::hasSpouse_EQ_civstatMarried -d
use> gen flags Person::maleHasNoHusband            -d
use> gen flags Person::nameCapitalThenSmallLetters -d
use> gen flags Person::nameIsUnique                -d

use> gen start civilStatus.assl generateWorld(1,1,1)
     Progress of first Try in ASSL-Procedure (7 combinations):
     |-----------------------------------------------|
     #######
use> gen result
     Random number generator was initialized with 178.
     Checked 57 snapshots in 0,093s (613 snapshots/s).
     Checked 213 times in 0,093s (2.290 checks/s).
```

```
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
     Barriers blocked 156 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Person('Person1')
     !new Person('Person2')
     !@Person1.name := 'Ada'
     !@Person1.civstat := CivilStatus::married
     !@Person1.gender := Gender::female
     !@Person2.name := 'Ali'
     !@Person2.civstat := CivilStatus::single
     !@Person2.gender := Undefined
     !insert (@Person1,@Person2) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Person::attributesDefined': FAILED.
       -> false : Boolean
     checking invariant (2) `Person::femaleHasNoWife': OK.
     checking invariant (3) `Person::hasSpouse_EQ_civstatMarried': OK.
     checking invariant (4) `Person::maleHasNoHusband': OK.
     checking invariant (5) `Person::nameCapitalThenSmallLetters': OK.
     checking invariant (6) `Person::nameIsUnique': OK.
     checked 6 invariants in 0.031s, 1 failure.

use> reset

use> gen flags Person::attributesDefined           -d
use> gen flags Person::femaleHasNoWife             +n
use> gen flags Person::hasSpouse_EQ_civstatMarried -d
use> gen flags Person::maleHasNoHusband            -d
use> gen flags Person::nameCapitalThenSmallLetters -d
use> gen flags Person::nameIsUnique                -d

use> gen start civilStatus.assl generateWorld(1,1,1)
     Progress of first Try in ASSL-Procedure (7 combinations):
     |---------------------------------------------|
     #######
use> gen result
     Random number generator was initialized with 5792.
     Checked 65 snapshots in 0,110s (591 snapshots/s).
     Checked 211 times in 0,110s (1.918 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
     Barriers blocked 146 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Person('Person1')
     !new Person('Person2')
     !@Person1.name := 'Ada'
```

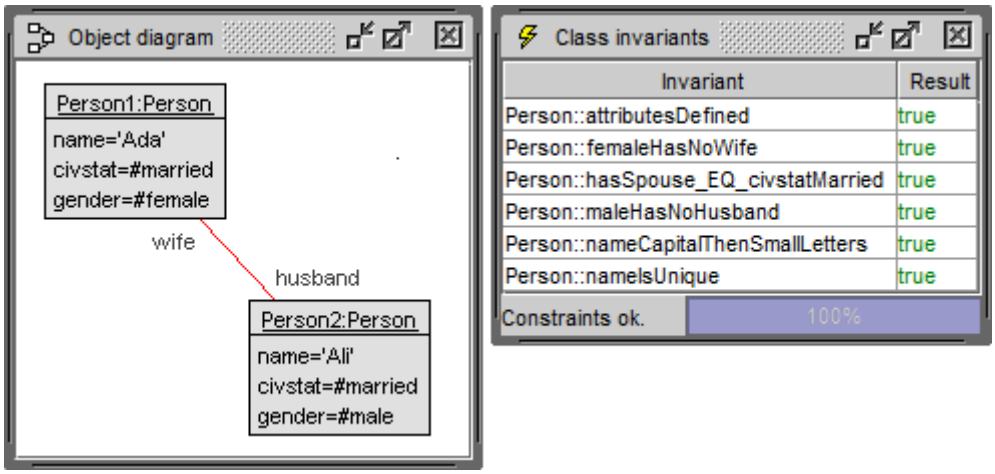
```
     !@Person1.civstat := CivilStatus::married
     !@Person1.gender := Gender::female
     !@Person2.name := 'Ali'
     !@Person2.civstat := CivilStatus::single
     !@Person2.gender := Gender::female
     !insert (@Person1,@Person2) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Person::attributesDefined': OK.
     checking invariant (2) `Person::femaleHasNoWife': FAILED.
       -> false : Boolean
     checking invariant (3) `Person::hasSpouse_EQ_civstatMarried': OK.
     checking invariant (4) `Person::maleHasNoHusband': OK.
     checking invariant (5) `Person::nameCapitalThenSmallLetters': OK.
     checking invariant (6) `Person::nameIsUnique': OK.
     checked 6 invariants in 0.016s, 1 failure.

use> reset

use> gen flags Person::attributesDefined           -d
use> gen flags Person::femaleHasNoWife             -d
use> gen flags Person::hasSpouse_EQ_civstatMarried +n
use> gen flags Person::maleHasNoHusband            -d
use> gen flags Person::nameCapitalThenSmallLetters -d
use> gen flags Person::nameIsUnique                -d

use> gen start civilStatus.assl generateWorld(1,1,1)
     Progress of first Try in ASSL-Procedure (7 combinations):
     |-------------------------------------------------|
     #######
use> gen result
     Random number generator was initialized with 3516.
     Checked 2 snapshots in 0,016s (125 snapshots/s).
     Checked 2 times in 0,016s (125 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
     Barriers blocked 0 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Person('Person1')
     !new Person('Person2')
     !@Person1.name := 'Ada'
     !@Person1.civstat := CivilStatus::single
     !@Person1.gender := Gender::female
     !@Person2.name := 'Ali'
     !@Person2.civstat := CivilStatus::single
     !@Person2.gender := Gender::male
     !insert (@Person1,@Person2) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> check
```

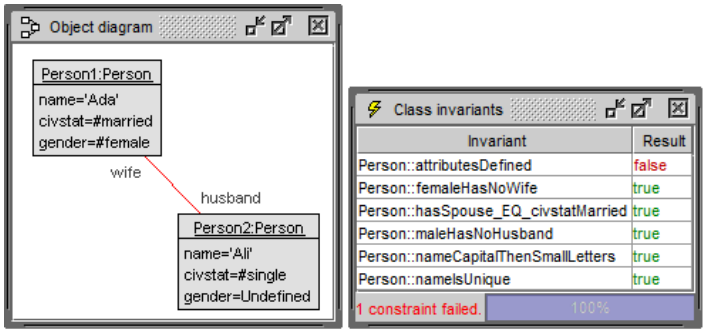
```
     checking structure...
     checking invariants...
     checking inv (1) `Person::attributesDefined': OK.
     checking inv (2) `Person::femaleHasNoWife': OK.
     checking inv (3) `Person::hasSpouse_EQ_civstatMarried': FAILED.
       -> false : Boolean
     checking inv (4) `Person::maleHasNoHusband': OK.
     checking inv (5) `Person::nameCapitalThenSmallLetters': OK.
     checking inv (6) `Person::nameIsUnique': OK.
     checked 6 invariants in 0.031s, 1 failure.

use> reset

use> gen flags Person::attributesDefined           -d
use> gen flags Person::femaleHasNoWife             -d
use> gen flags Person::hasSpouse_EQ_civstatMarried -d
use> gen flags Person::maleHasNoHusband            +n
use> gen flags Person::nameCapitalThenSmallLetters -d
use> gen flags Person::nameIsUnique                -d

use> gen start civilStatus.assl generateWorld(1,1,1)
     Progress of first Try in ASSL-Procedure (7 combinations):
     |-----------------------------------------------|
     #######
use> gen result
     Random number generator was initialized with 5061.
     Checked 36 snapshots in 0,047s (766 snapshots/s).
     Checked 110 times in 0,047s (2.340 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
     Barriers blocked 74 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Person('Person1')
     !new Person('Person2')
     !@Person1.name := 'Ada'
     !@Person1.civstat := CivilStatus::single
     !@Person1.gender := Gender::male
     !@Person2.name := 'Ali'
     !@Person2.civstat := CivilStatus::married
     !@Person2.gender := Gender::male
     !insert (@Person1,@Person2) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Person::attributesDefined': OK.
     checking invariant (2) `Person::femaleHasNoWife': OK.
     checking invariant (3) `Person::hasSpouse_EQ_civstatMarried': OK.
     checking invariant (4) `Person::maleHasNoHusband': FAILED.
       -> false : Boolean
     checking invariant (5) `Person::nameCapitalThenSmallLetters': OK.
     checking invariant (6) `Person::nameIsUnique': OK.
```

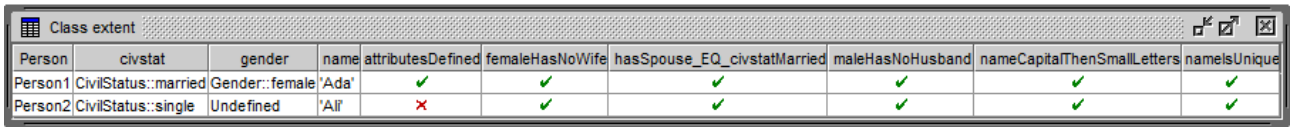
```
     checked 6 invariants in 0.016s, 1 failure.

use> reset

use> gen flags Person::attributesDefined           -d
use> gen flags Person::femaleHasNoWife             -d
use> gen flags Person::hasSpouse_EQ_civstatMarried -d
use> gen flags Person::maleHasNoHusband            -d
use> gen flags Person::nameCapitalThenSmallLetters +n
use> gen flags Person::nameIsUnique                -d

use> gen start civilStatus.assl generateWorld(1,1,1)
     Progress of first Try in ASSL-Procedure (7 combinations):
     |-----------------------------------------------|
     #######
use> gen result
     Random number generator was initialized with 3251.
     Checked 36 snapshots in 0,078s (462 snapshots/s).
     Checked 245 times in 0,078s (3.141 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
     Barriers blocked 209 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Person('Person1')
     !new Person('Person2')
     !@Person1.name := 'Ada'
     !@Person1.civstat := CivilStatus::married
     !@Person1.gender := Gender::female
     !@Person2.name := 'ali'
     !@Person2.civstat := CivilStatus::married
     !@Person2.gender := Gender::male
     !insert (@Person1,@Person2) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> check
     checking structure...
     checking invariants...
     checking inv (1) `Person::attributesDefined': OK.
     checking inv (2) `Person::femaleHasNoWife': OK.
     checking inv (3) `Person::hasSpouse_EQ_civstatMarried': OK.
     checking inv (4) `Person::maleHasNoHusband': OK.
     checking inv (5) `Person::nameCapitalThenSmallLetters': FAILED.
       -> false : Boolean
     checking inv (6) `Person::nameIsUnique': OK.
     checked 6 invariants in 0.031s, 1 failure.

use> reset

use> gen flags Person::attributesDefined           -d
use> gen flags Person::femaleHasNoWife             -d
use> gen flags Person::hasSpouse_EQ_civstatMarried -d
use> gen flags Person::maleHasNoHusband            -d
use> gen flags Person::nameCapitalThenSmallLetters -d
```

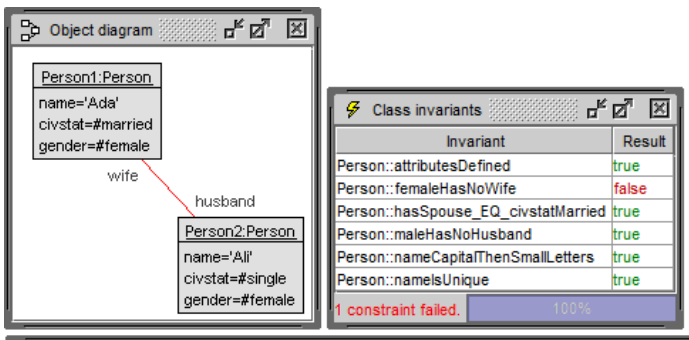
```
use> gen flags Person::nameIsUnique                +n

use> gen start civilStatus.assl generateWorld(1,1,1)
     Progress of first Try in ASSL-Procedure (7 combinations):
     |-----------------------------------------------|
     ###################################
use> gen result
     Random number generator was initialized with 7632.
     Checked 20 snapshots in 0,328s (61 snapshots/s).
     Checked 3.635 times in 0,328s (11.082 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
     Barriers blocked 3.615 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Person('Person1')
     !new Person('Person2')
     !@Person1.name := 'Sam'
     !@Person1.civstat := CivilStatus::married
     !@Person1.gender := Gender::female
     !@Person2.name := 'Sam'
     !@Person2.civstat := CivilStatus::married
     !@Person2.gender := Gender::male
     !insert (@Person1,@Person2) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Person::attributesDefined': OK.
     checking invariant (2) `Person::femaleHasNoWife': OK.
     checking invariant (3) `Person::hasSpouse_EQ_civstatMarried': OK.
     checking invariant (4) `Person::maleHasNoHusband': OK.
     checking invariant (5) `Person::nameCapitalThenSmallLetters': OK.
     checking invariant (6) `Person::nameIsUnique': FAILED.
       -> false : Boolean
     checked 6 invariants in 0.015s, 1 failure.

use> open civilStatus.use

use> gen load bigamy.invs
     Added invariants:
     Person::bigamy
use> gen start civilStatus.assl attemptBigamy()
     Progress of first Try in ASSL-Procedure (3 combinations):
     |-----------------------------------------------|
     ###################################################
use> gen result
     Random number generator was initialized with 5954.
     Checked 18.432 snapshots in 1,107s (16.650 snapshots/s).
     Checked 29.184 times in 1,107s (26.363 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 3 barriers.
```

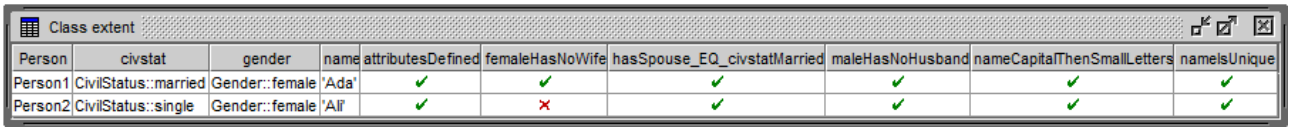
```
Barriers blocked 10.752 times.
Result: No valid state found.
```

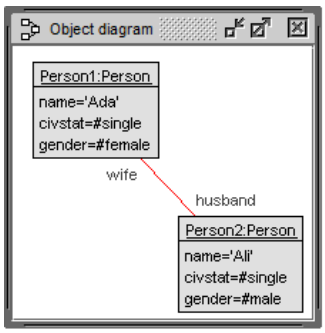












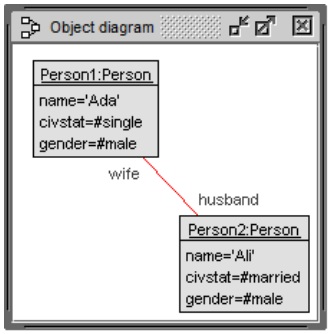
Class invariants

Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	true
Person::hasSpouse_EQ_civstatMarried	false
Person::maleHasNoHusband	true
Person::nameCapitalThenSmallLetters	true
Person::namelsUnique	true

1 constraint failed. 100%

Class extent

Person	civstat	gender	name	attributesDefined	femaleHasNoWife	hasSpouse_EQ_civstatMarried	maleHasNoHusband	nameCapitalThenSmallLetters	namelsUnique
Person1	CivilStatus::single	Gender::female	'Ada'	✔	✔	✘	✔	✔	✔
Person2	CivilStatus::single	Gender::male	'Ali'	✔	✔	✘	✔	✔	✔



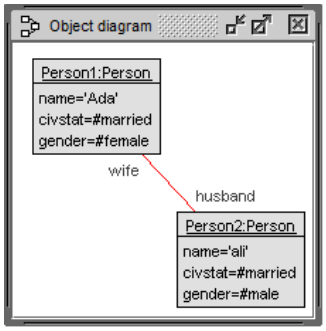
Class invariants

Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	true
Person::hasSpouse_EQ_civstatMarried	true
Person::maleHasNoHusband	false
Person::nameCapitalThenSmallLetters	true
Person::namelsUnique	true

1 constraint failed. 100%

Class extent

Person	civstat	gender	name	attributesDefined	femaleHasNoWife	hasSpouse_EQ_civstatMarried	maleHasNoHusband	nameCapitalThenSmallLetters	namelsUnique
Person1	CivilStatus::single	Gender::male	'Ada'	✔	✔	✔	✘	✔	✔
Person2	CivilStatus::married	Gender::male	'Ali'	✔	✔	✔	✔	✔	✔



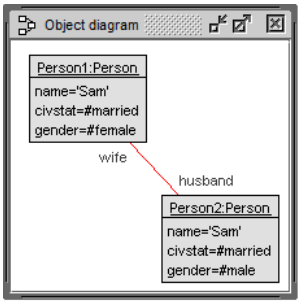
Class invariants

Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	true
Person::hasSpouse_EQ_civstatMarried	true
Person::maleHasNoHusband	true
Person::nameCapitalThenSmallLetters	false
Person::namelsUnique	true

1 constraint failed. 100%

Class extent

Person	civstat	gender	name	attributesDefined	femaleHasNoWife	hasSpouse_EQ_civstatMarried	maleHasNoHusband	nameCapitalThenSmallLetters	namelsUnique
Person1	CivilStatus::married	Gender::female	'Ada'	✔	✔	✔	✔	✔	✔
Person2	CivilStatus::married	Gender::male	'ali'	✔	✔	✔	✔	✘	✔



Class invariants

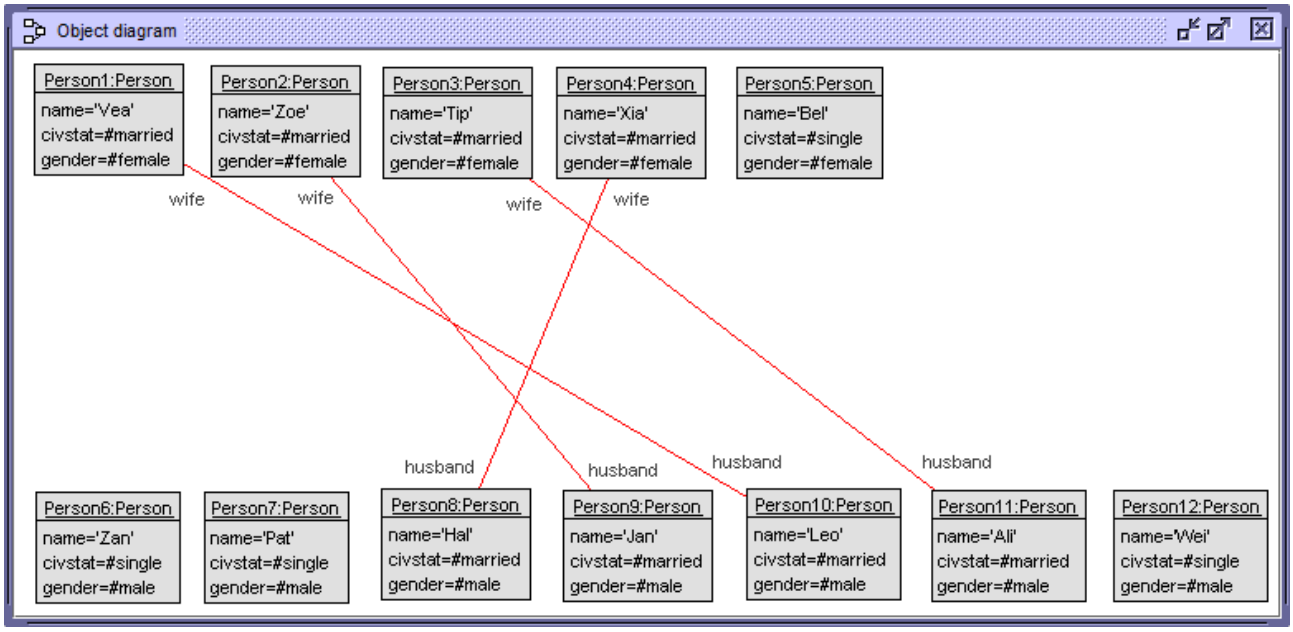
Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	true
Person::hasSpouse_EQ_civstatMarried	true
Person::maleHasNoHusband	true
Person::nameCapitalThenSmallLetters	true
Person::namelsUnique	false

1 constraint failed. 100%

Class extent

Person	civstat	gender	name	attributesDefined	femaleHasNoWife	hasSpouse_EQ_civstatMarried	maleHasNoHusband	nameCapitalThenSmallLetters	namelsUnique
Person1	CivilStatus::married	Gender::female	'Sam'	✔	✔	✔	✔	✔	✘
Person2	CivilStatus::married	Gender::male	'Sam'	✔	✔	✔	✔	✔	✘

```
use> open civilStatus.use

use> gen start civilStatus.assl largerWorld(5,7,4)
use> gen result
use> gen result accept
```

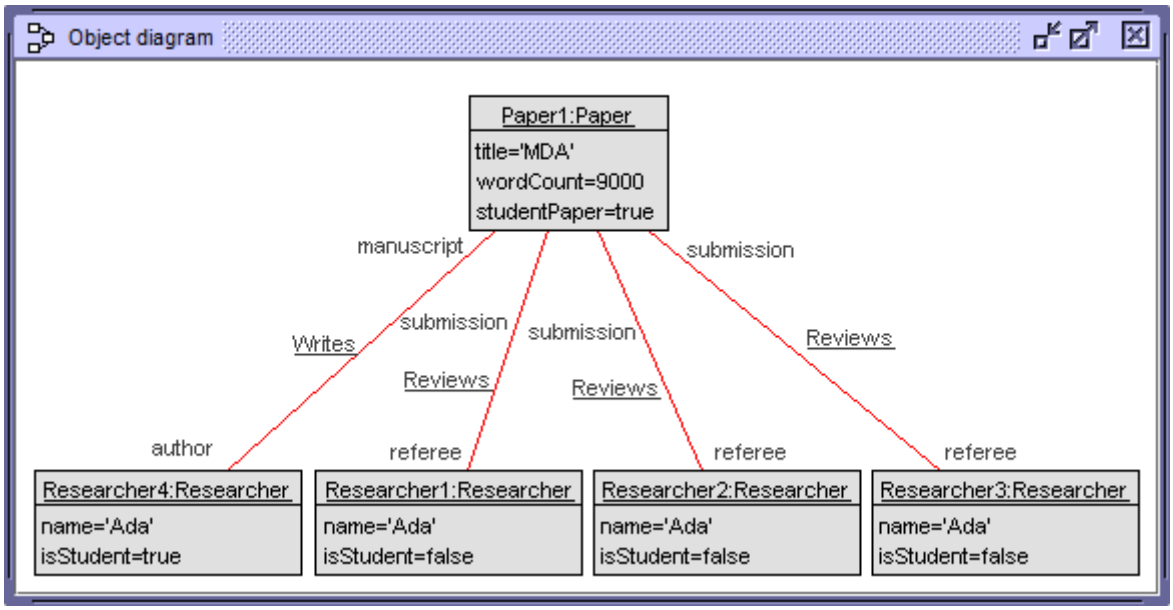


2.2 WritesReviews (WR)

```
procedure generateWorld(numPap:Integer,numRes:Integer,fillAttr:Boolean)
var papers:Sequence(Paper), researchers:Sequence(Researcher);
begin
papers:=CreateN(Paper,[numPap]);
researchers:=CreateN(Researcher,[numRes]);
Try(Writes,[researchers],[papers]);
Try(Reviews,[researchers],[papers]);
if [fillAttr] then begin
  for i:Integer in [Sequence{1..numPap}] begin
    [papers->at(i)].title:=Try([Sequence{'MDA','OCL','UML'}]);
    [papers->at(i)].wordCount:=Try([Sequence{9000,10000,11000}]);
    [papers->at(i)].studentPaper:=Try([Sequence{false,true}]);
    end;
  for i:Integer in [Sequence{1..numRes}] begin
    [researchers->at(i)].name:=Try([Sequence{'Ada','Bob','Cyd'}]);
    [researchers->at(i)].isStudent:=Try([Sequence{false,true}]);
    end;
  end;
end;
```

```
use> open writesReviews.use

use> -- Paper::authorsOfStudentPaper
use> -- Paper::limitsOnStudentPapers
use> -- Paper::noStudentReviewers
use> -- Paper::paperLength
use> -- Researcher::noSelfReviews
use> -- Researcher::oneManuscript
use> -- Researcher::oneSubmission

use> gen start writesReviews.assl generateWorld(4,4,false)
     Progress of first Try in ASSL-Procedure (625 combinations):
     |-------------------------------------------------|
     ##################################################use> gen result
     Random number generator was initialized with 7327.
     Checked 20.736 snapshots in 3,822s (5.425 snapshots/s).
     Checked 160.369 times in 3,822s (41.959 checks/s).
     Made 0 try cuts.
     Ignored at least 16.682.127 useless link combinations.
     Added 3 barriers.
     Barriers blocked 139.633 times.
     Result: No valid state found.
use> gen result accept
     No commands available.

use> reset

use> -- Paper::authorsOfStudentPaper
use> -- Paper::limitsOnStudentPapers
use> -- Paper::noStudentReviewers
use> -- Paper::paperLength
use> -- Researcher::noSelfReviews
use> gen flags Researcher::oneManuscript +d
use> gen flags Researcher::oneSubmission +d

use> gen start writesReviews.assl generateWorld(1,4,true)
     Progress of first Try in ASSL-Procedure (16 combinations):
     |-----------------------------------------------|
     ######use> gen result
     Random number generator was initialized with 6192.
     Checked 537.842 snapshots in 3,495s (153.889 snapshots/s).
     Checked 537.849 times in 3,495s (153.891 checks/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 1 barriers.
     Barriers blocked 7 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Paper('Paper1')
     !new Researcher('Researcher1')
     !new Researcher('Researcher2')
     !new Researcher('Researcher3')
     !new Researcher('Researcher4')
     !insert (@Researcher4,@Paper1) into Writes
     !insert (@Researcher3,@Paper1) into Reviews
```

```
      !insert (@Researcher2,@Paper1) into Reviews
      !insert (@Researcher1,@Paper1) into Reviews
      !@Paper1.title := 'MDA'
      !@Paper1.wordCount := 9000
      !@Paper1.studentPaper := true
      !@Researcher1.name := 'Ada'
      !@Researcher1.isStudent := false
      !@Researcher2.name := 'Ada'
      !@Researcher2.isStudent := false
      !@Researcher3.name := 'Ada'
      !@Researcher3.isStudent := false
      !@Researcher4.name := 'Ada'
      !@Researcher4.isStudent := true
use> gen result accept
     Generated result (system state) accepted.
```



2.3 DisjointSubclasses (DS)

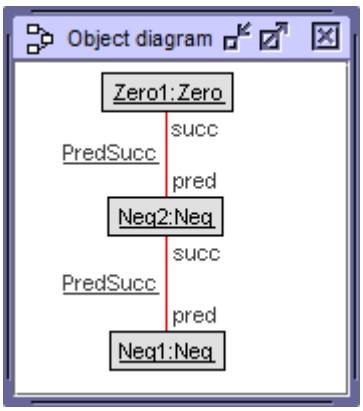
```
procedure generateWorld(noA:Integer,noB:Integer,noC:Integer,noD:Integer)
var a_s:Sequence(A), b_s:Sequence(B), c_s:Sequence(C), d_s:Sequence(D);
begin
a_s:=CreateN(A,[noA]);
b_s:=CreateN(B,[noB]);
c_s:=CreateN(C,[noC]);
d_s:=CreateN(D,[noD]);
end;

use> open disjointSubclasses.use

use> gen disjointSubclasses.assl generateWorld(1,1,1,0)
use> gen result
     Random number generator was initialized with 4117.
     Checked 1 snapshots in 0,000s (NaN snapshots/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
```

```
     Added 0 barriers.
     Barriers blocked 0 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new A('A1')
     !new B('B1')
     !new C('C1')
use> gen result accept
     Generated result (system state) accepted.

use> reset
use> gen start disjointSubclasses.assl generateWorld(1,1,1,1)
use> gen result
     Random number generator was initialized with 3866.
     Checked 1 snapshots in 0,000s (NaN snapshots/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 0 barriers.
     Barriers blocked 0 times.
     Result: No valid state found.
use> gen result accept
     No commands available.

use> reset
use> gen flags B::disjointBC +d
use> gen start disjointSubclasses.assl generateWorld(1,1,1,1)
use> gen result
     Random number generator was initialized with 1939.
     Checked 1 snapshots in 0,000s (NaN snapshots/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 0 barriers.
     Barriers blocked 0 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new A('A1')
     !new B('B1')
     !new C('C1')
     !new D('D1')
use> gen result accept
     Generated result (system state) accepted.
use> ?D.allInstances
     Set{@D1} : Set(D)
use> ?C.allInstances
     Set{@C1,@D1} : Set(C)
use> ?B.allInstances
     Set{@B1,@D1} : Set(B)
use> ?A.allInstances
     Set{@A1,@B1,@C1,@D1} : Set(A)
```

2.4 ObjectsAsIntegers (OAI)

```
procedure generateWorld(intNum:Integer,predSuccNum:Integer)
var ints:Sequence(Int), aux:Int,
    nzp:Integer, pInt:Int, sInt:Int;
begin
ints:=[Sequence{}];
for i:Integer in [Sequence{1..intNum}] begin
  nzp:=Any([Sequence{-1,0,+1}]);
  if [nzp=-1] then begin
    aux:=Create(Neg); ints:=[ints->append(aux)]; end;
  if [nzp=0]  then begin
    aux:=Create(Zero); ints:=[ints->append(aux)]; end;
  if [nzp=+1] then begin
    aux:=Create(Pos); ints:=[ints->append(aux)]; end;
  end;
for i:Integer in [Sequence{1..predSuccNum}] begin
  pInt:=Any([ints->select(i|i.succ->isEmpty)]);
  sInt:=Any([ints->select(i|i.pred->isEmpty)]);
  Insert(PredSucc,[pInt],[sInt]);
  end;
end;

--------------------------------------------------------------------------

use> open objectsAsIntegers.use

use> gen flags Int::acyclicPredSucc           +d
use> gen flags Neg::negLinkedToZeroBySucc     +d
use> gen flags Neg::negNotLinkedToPos         +d
use> gen flags Pos::posLinkedToZeroByPred     +d
use> gen flags Pos::posNotLinkedToNeg         +d
use> gen flags Zero::zeroNotLinkedToNegAndPos +d
use> gen flags Zero::zeroNotLinkedToZero      +d

use> gen start -r 4900 objectsAsIntegers.assl generateWorld(3,2)
use> gen result
     Random number generator was initialized with 4900.
     Checked 1 snapshots in 0,016s (63 snapshots/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 0 barriers.
     Barriers blocked 0 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Neg('Neg1')
     !new Neg('Neg2')
     !new Zero('Zero1')
     !insert (@Neg2,@Zero1) into PredSucc
     !insert (@Neg1,@Neg2) into PredSucc
use> gen result accept
     Generated result (system state) accepted.

use> check
     checking structure...
```

```
     checking invariants...
     checking invariant (1) `Int::acyclicPredSucc': OK.
     checking invariant (2) `Neg::negLinkedToZeroBySucc': OK.
     checking invariant (3) `Neg::negNotLinkedToPos': OK.
     checking invariant (4) `Pos::posLinkedToZeroByPred': OK.
     checking invariant (5) `Pos::posNotLinkedToNeg': OK.
     checking invariant (6) `Zero::zeroNotLinkedToNegAndPos': OK.
     checking invariant (7) `Zero::zeroNotLinkedToZero': OK.
     checked 7 invariants in 0.016s, 0 failures.

use> reset

use> gen flags Int::acyclicPredSucc           +d
use> gen flags Neg::negLinkedToZeroBySucc     +d
use> gen flags Neg::negNotLinkedToPos         +d
use> gen flags Pos::posLinkedToZeroByPred     +d
use> gen flags Pos::posNotLinkedToNeg         +d
use> gen flags Zero::zeroNotLinkedToNegAndPos +d
use> gen flags Zero::zeroNotLinkedToZero      +d

use> gen start -r 9590 objectsAsIntegers.assl generateWorld(3,2)
use> gen result
     Random number generator was initialized with 9590.
     Checked 1 snapshots in 0,016s (63 snapshots/s).
     Made 0 try cuts.
     Ignored at least 0 useless link combinations.
     Added 0 barriers.
     Barriers blocked 0 times.
     Result: Valid state found.
     Commands to produce the valid state:
     !new Neg('Neg1')
     !new Neg('Neg2')
     !new Zero('Zero1')
     !insert (@Zero1,@Neg1) into PredSucc
     !insert (@Neg2,@Zero1) into PredSucc
use> gen result accept
     Generated result (system state) accepted.

use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Int::acyclicPredSucc': OK.
     checking invariant (2) `Neg::negLinkedToZeroBySucc': FAILED.
       -> false : Boolean
     checking invariant (3) `Neg::negNotLinkedToPos': OK.
     checking invariant (4) `Pos::posLinkedToZeroByPred': OK.
     checking invariant (5) `Pos::posNotLinkedToNeg': OK.
     checking invariant (6) `Zero::zeroNotLinkedToNegAndPos': OK.
     checking invariant (7) `Zero::zeroNotLinkedToZero': OK.
     checked 7 invariants in 0.015s, 1 failure.
```


Object diagram
Zero1:Zero
succ
PredSucc
pred
Neg2:Neg
succ
PredSucc
pred
Neg1:Neg

3 Details of Benchmark Analysis with EMFtoCSP

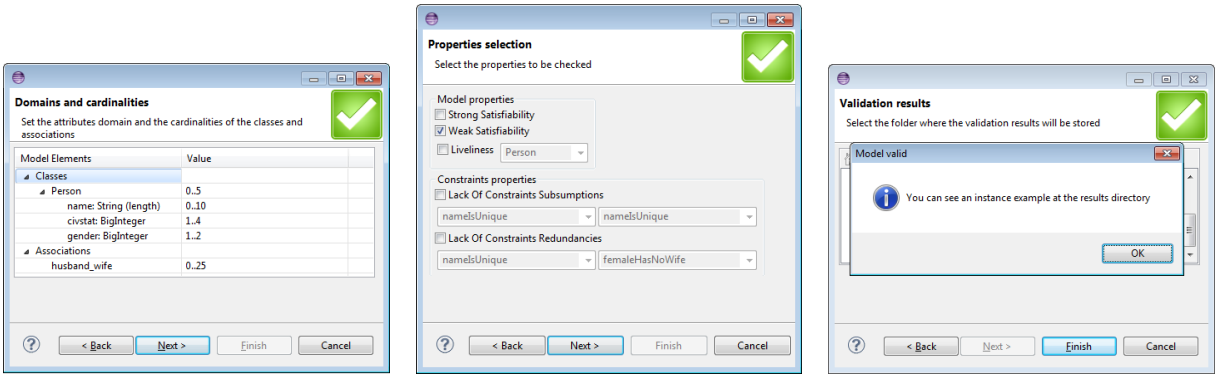
3.1 CivilStatus (CS)

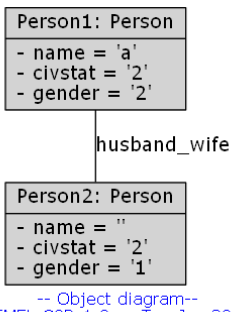
CivilStatus.ecore:

```
package CivilStatus {
  class Person {
    attribute name : String;
    -- 1 = #single, 2 = #married, 3 = #divorced, 4 = #widowed
    attribute civstat : Integer;
    -- 1 = #female, 2 = #male
    attribute gender : Integer;
    property wife#husband : Person[0..1];
    property husband#wife : Person[0..1];
    invariant nameIsUnique: Person.allInstances()->forAll(self2 : Person |
        self <> self2 implies self.name <> self2.name);
    invariant femaleHasNoWife: gender = 1 implies wife.oclAsSet()->isEmpty();
    invariant maleHasNoHusband: gender = 2 implies husband.oclAsSet()->isEmpty();
    invariant hasSpouse_EQ_civstatMarried:
      if gender = 1
        then husband
        else if gender = 2 then wife else null endif
      endif.oclAsSet()->notEmpty() = (civstat = 2);
  }
}
```

Remarks: The example requires some 'syntactic desugaring' in EMFtoCSP: (1) Representation of enumerations as integers; (2) explicit unfolding of query operations; (3) explicit use of 'oclAsSet()' when applying collection operations to object-valued properties (with 0..1 multiplicities). Ecore does not have association names; the association names are generated by EMFtoECL from the role names (e.g., *Marriage* becomes *husband_wife*).

Checking ConsistentInvariants



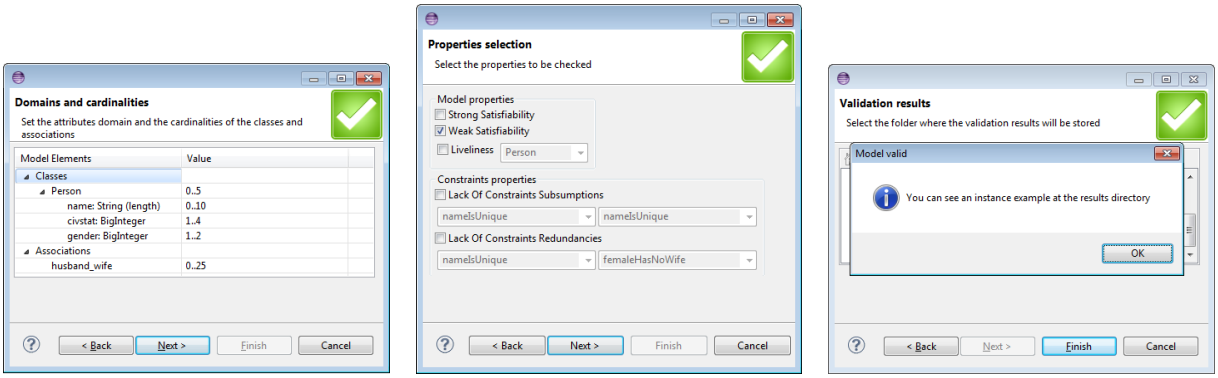


-- Object diagram--
Generated by EMFtoCSPv1.0 on Tue Jan 29 14:50:41 2013

Checking Independence

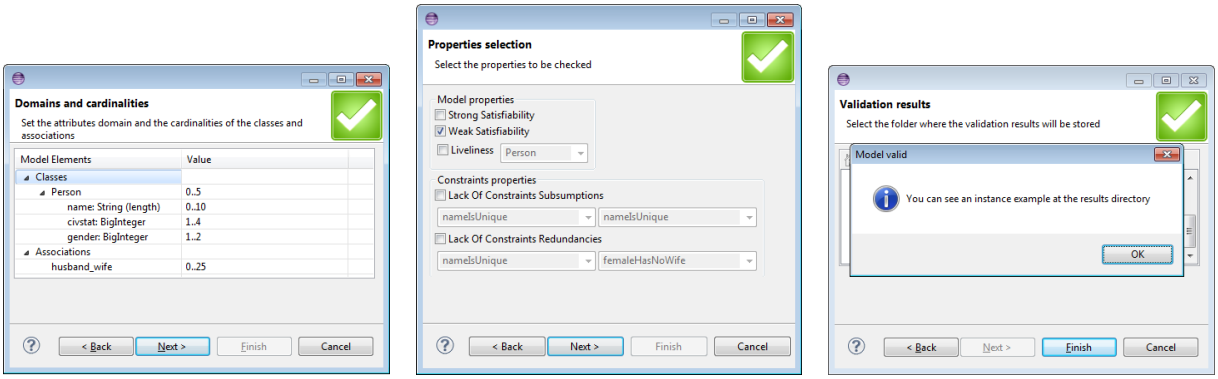
CivilStatusNotNameIsUnique.ecore:

```
package CivilStatusNotNameIsUnique {
  class Person {
    -- other attributes, properties, and invariants as in CivilStatus.ecore
    -- invariant nameIsUnique: Person.allInstances()->forAll(self2 : Person |
        self <> self2 implies self.name <> self2.name);
    invariant notNameIsUnique: not (Person.allInstances()->forAll(self2 : Person |
        self <> self2 implies self.name <> self2.name));
  }
}
```



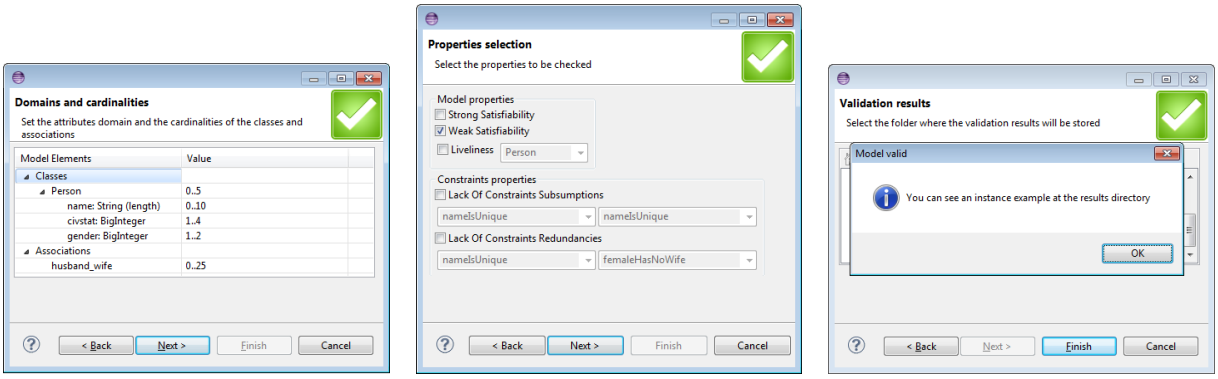
CivilStatusNotFemaleHasNoWife.ecore:

```
package CivilStatusNotFemaleHasNoWife {
  class Person {
    -- other attributes, properties, and invariants as in CivilStatus.ecore
    --invariant femaleHasNoWife: gender = 1 implies wife.oclAsSet()->isEmpty();
    invariant notFemaleHasNoWife: not (gender = 1 implies wife.oclAsSet()->isEmpty());
  }
}
```



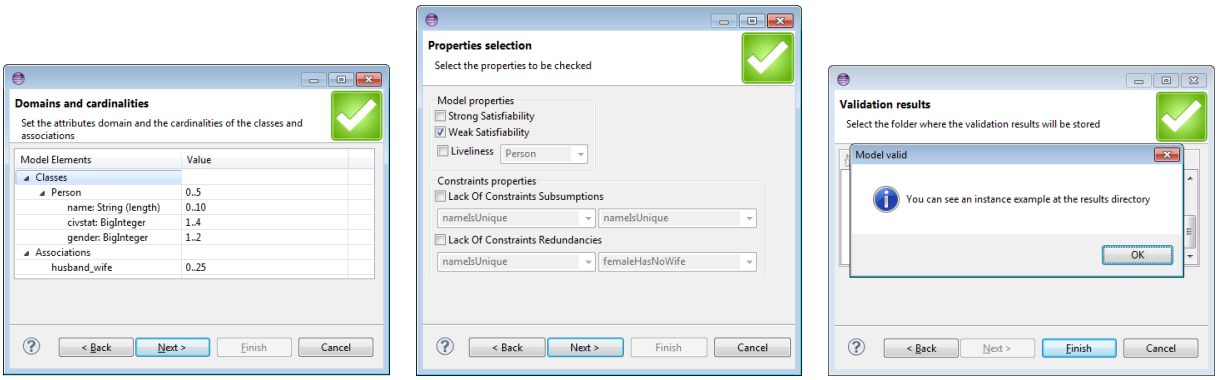
CivilStatusNotMaleHasNoHusband.ecore:

```
package CivilStatusNotMaleHasNoHusband {
  class Person {
    -- other attributes, properties, and invariants as in CivilStatus.ecore
    -- invariant maleHasNoHusband: gender = 2 implies husband.oclAsSet()->isEmpty();
    invariant notMaleHasNoHusband: not (gender = 2 implies husband.oclAsSet()->isEmpty());
  }
}
```



CivilStatusNotHasSpouseEqCivstatMarried.ecore:

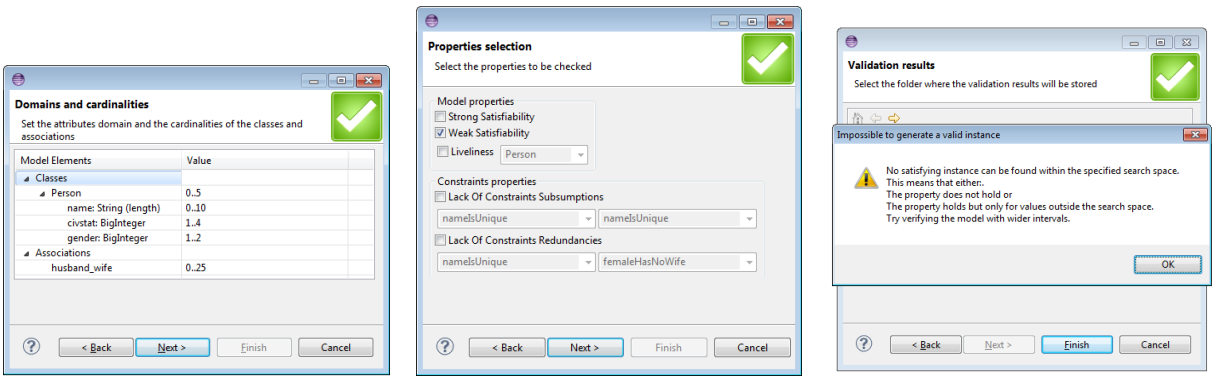
```
package CivilStatusNotHasSpouseEqCivstatMarried {
  class Person {
    -- other attributes, properties, and invariants as in CivilStatus.ecore
    -- invariant hasSpouse_EQ_civstatMarried:
    --   if gender = 1
    --     then husband
    --     else if gender = 2 then wife else null endif
    --   endif.oclAsSet()->notEmpty() = (civstat = 2);
    invariant notHasSpouse_EQ_civstatMarried:
      not (if gender = 1
        then husband
        else if gender = 2 then wife else null endif
      endif.oclAsSet()->notEmpty() = (civstat = 2));
  }
}
```



Checking Consequences

CivilStatusBigamy.ecore:

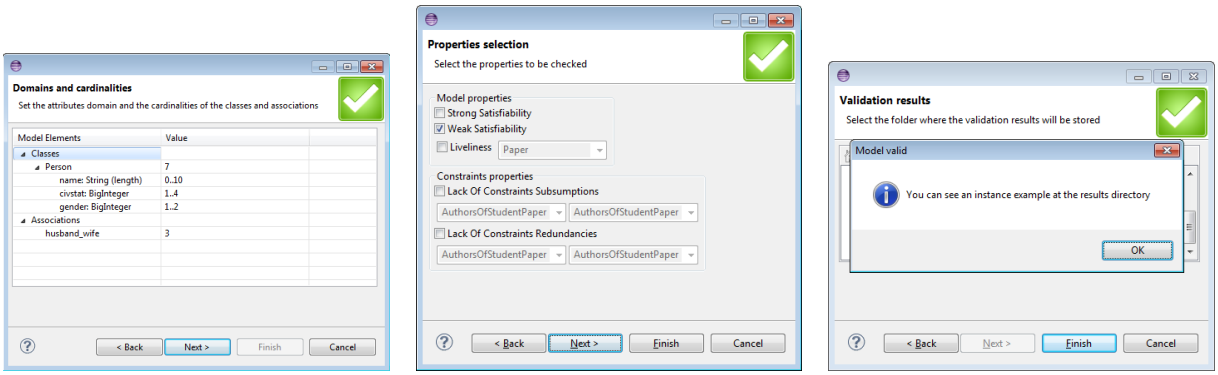
```
package CivilStatusBigamy {
  class Person {
    -- other attributes, properties, and invariants as in CivilStatus.ecore
    invariant notIsBigamyFree:
      not (not (husband->oclAsSet()->notEmpty() and
                wife.oclAsSet()->notEmpty()));
  }
}
```

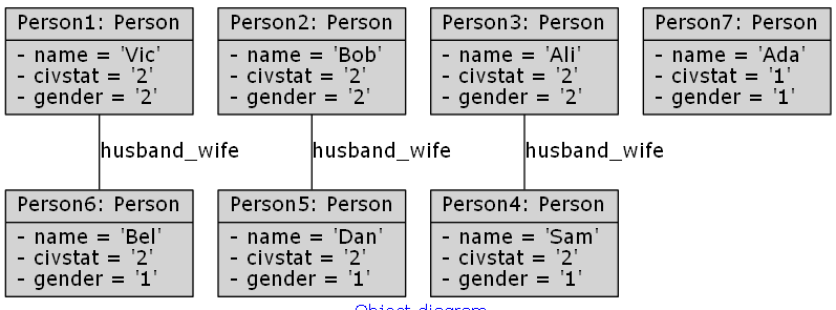


Checking LargeState

CivilStatusLargeState.ecore:

```
package CivilStatusLargeState {
  class Person {
    -- other attributes, properties, and invariants as in CivilStatus.ecore
    invariant niceInstance:
      (name = 'Ada' or name = 'Ali' or name = 'Bel' or name = 'Bob' or
       name = 'Dan' or name = 'Sam' or name = 'Vic') and
      (name = 'Ada' implies gender = 1) and
      (name = 'Ali' implies gender = 2) and
      (name = 'Bel' implies gender = 1) and
      (name = 'Bob' implies gender = 2);
  }
}
```





-- Object diagram--
Generated by EMFtoCSPv1.0 on Tue Jan 29 17:33:08 2013

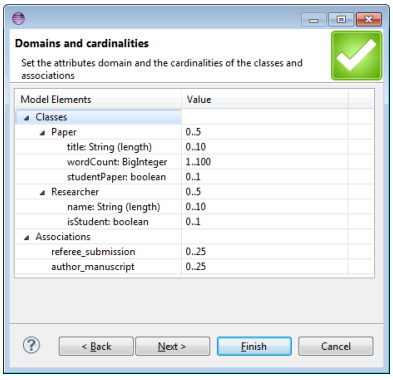
3.2 WritesReviews

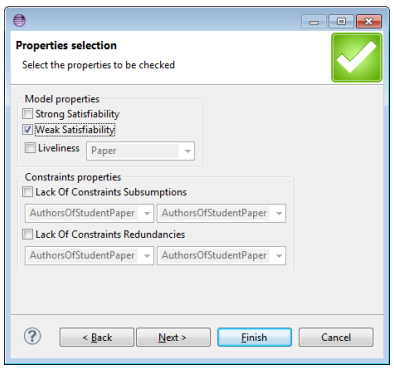
WritesReviews.ecore:

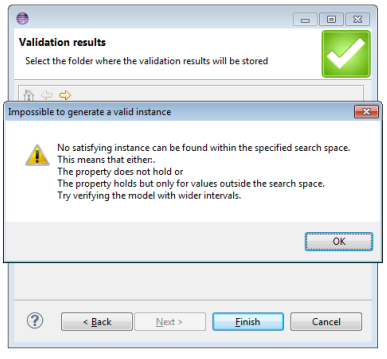
```
package WritesReviews {
  class Paper {
    attribute title : String;
    attribute wordCount : Integer;
    attribute studentPaper : Boolean;
    property referee#submission : Researcher[3..3];
    property author#manuscript : Researcher[1..2];
    invariant AuthorsOfStudentPaper:
      self.studentPaper = self.author->exists(x|x.isStudent);
    invariant LimitsOnStudentPapers:
      Paper.allInstances()->exists(p|p.studentPaper) and
      Paper.allInstances()->select(p|p.studentPaper)->size() < 5;
    invariant PaperLength:
      self.wordCount < 10000;
    invariant NoStudentReviewers:
      referee->forAll(r| not r.isStudent);
  }
  class Researcher
  {
    attribute name : String;
    attribute isStudent : Boolean;
    property submission#referee : Paper[?];
    property manuscript#author : Paper[?];
    invariant oneManuscript:
      manuscript.oclAsSet()->size() = 1;
    invariant oneSubmission:
      submission.oclAsSet()->size() = 1;
    invariant noSelfReviews:
      submission.oclAsSet()->excludes(manuscript);
  }
}
```

Checking ConsistentMultiplicity

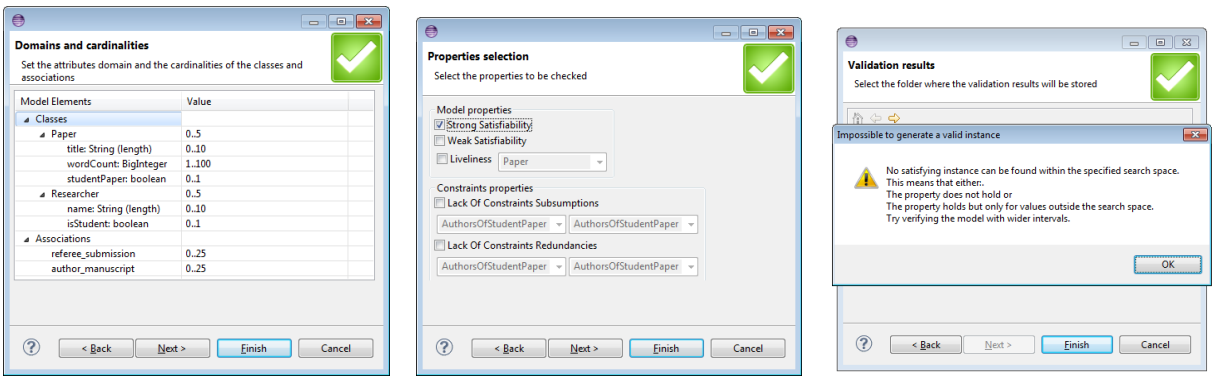
Checking InstantiateNonemptyClass







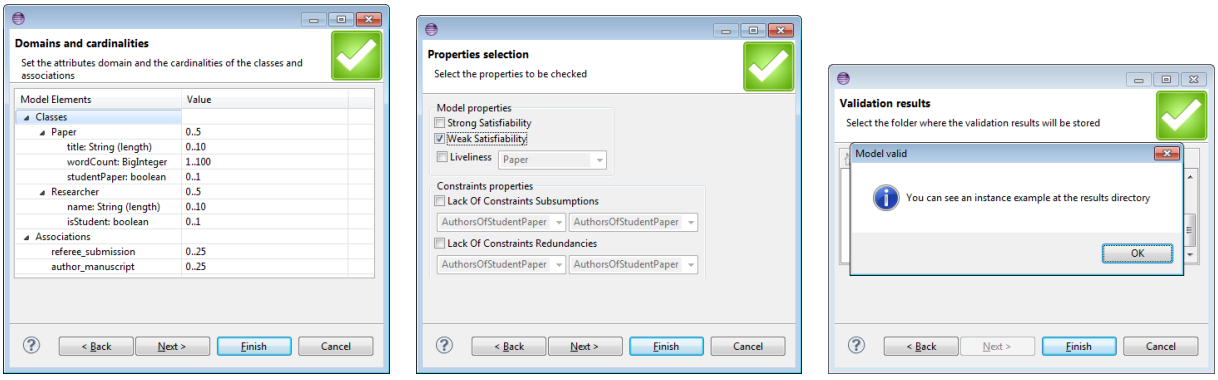
Checking InstantiateNonemptyAssoc

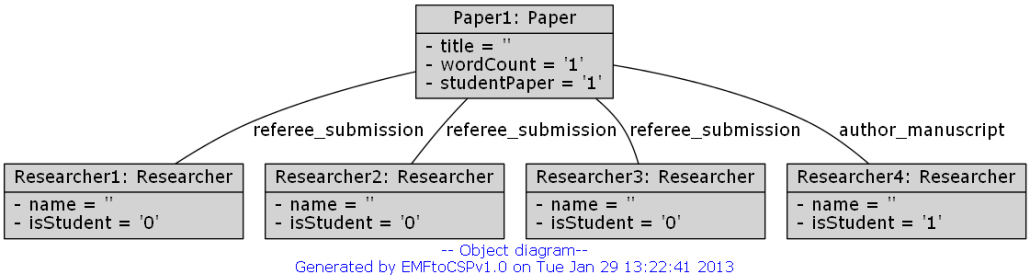


Checking InstantiateInvariantIgnore

WritesReviewsIgnoreOneManuscriptAndOneSubmission.ecore:

```
package WritesReviewsIgnoreOneManuscriptAndOneSubmission {
  class Paper {
    -- as in WritesReviews.ecore
  }
  class Researcher
  {
    -- other attributes, properties, and invariants as in WritesReviews.ecore
    --invariant oneManuscript:
    --  manuscript.oclAsSet()->size() = 1;
    --invariant oneSubmission:
    --  submission.oclAsSet()->size() = 1;
  }
}
```





3.3 DisjointSubclasses (DS)

Not applicable. Multiple inheritance is currently not accepted by EMFtoCSP (although the formal approach behind EMFtoCSP supports it).

3.4 ObjectsAsIntegers (OAI)

Not applicable. Recursive operations are currently not accepted by EMFtoCSP (although the formal approach behind EMFtoCSP supports it).