# Graph-Transformation-Based Support for Model Evolution

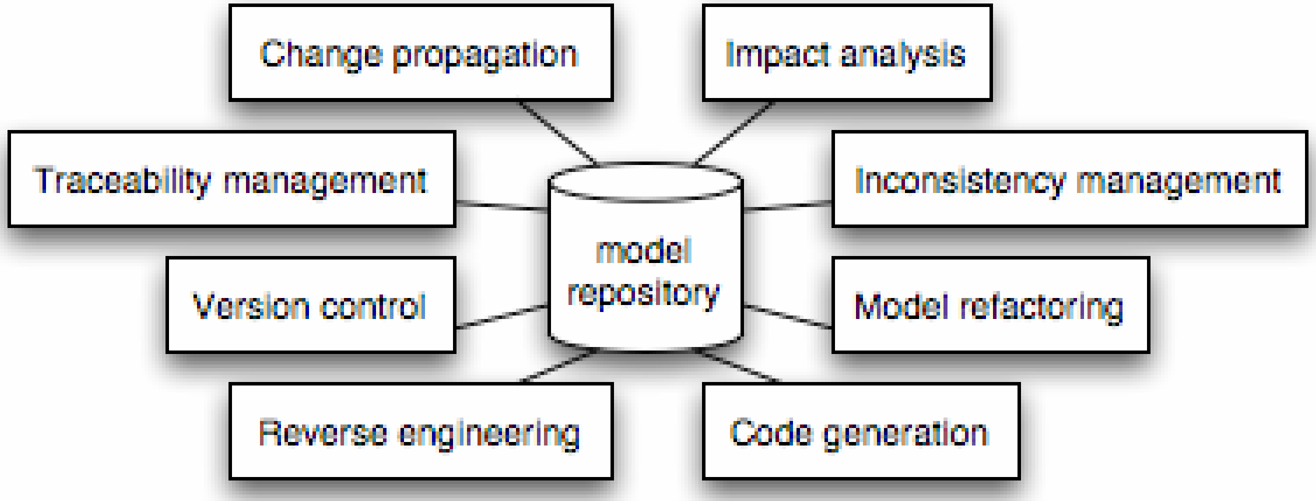
## SegraVis Advanced School on Visual Modeling Techniques

Tom **Mens**
Institut d'Informatique
Université de Mons-Hainaut
Belgium

Syntactic and Semantic Integration of Visual Modelling Techniques

# Challenge: Model-driven evolution

- Observation
  - Model-driven software engineering approaches do not adequately address software evolution problems
  - Need better support (tools, formalisms) for



# Goal of this talk

Show how graph transformation theory can help to

- gain better understanding of ...
- improve tool support for ...

... the following activities:

- inconsistency management
- model refactoring

# Model Inconsistency Management through Graph Transformation

## An Experiment

# Model inconsistency management



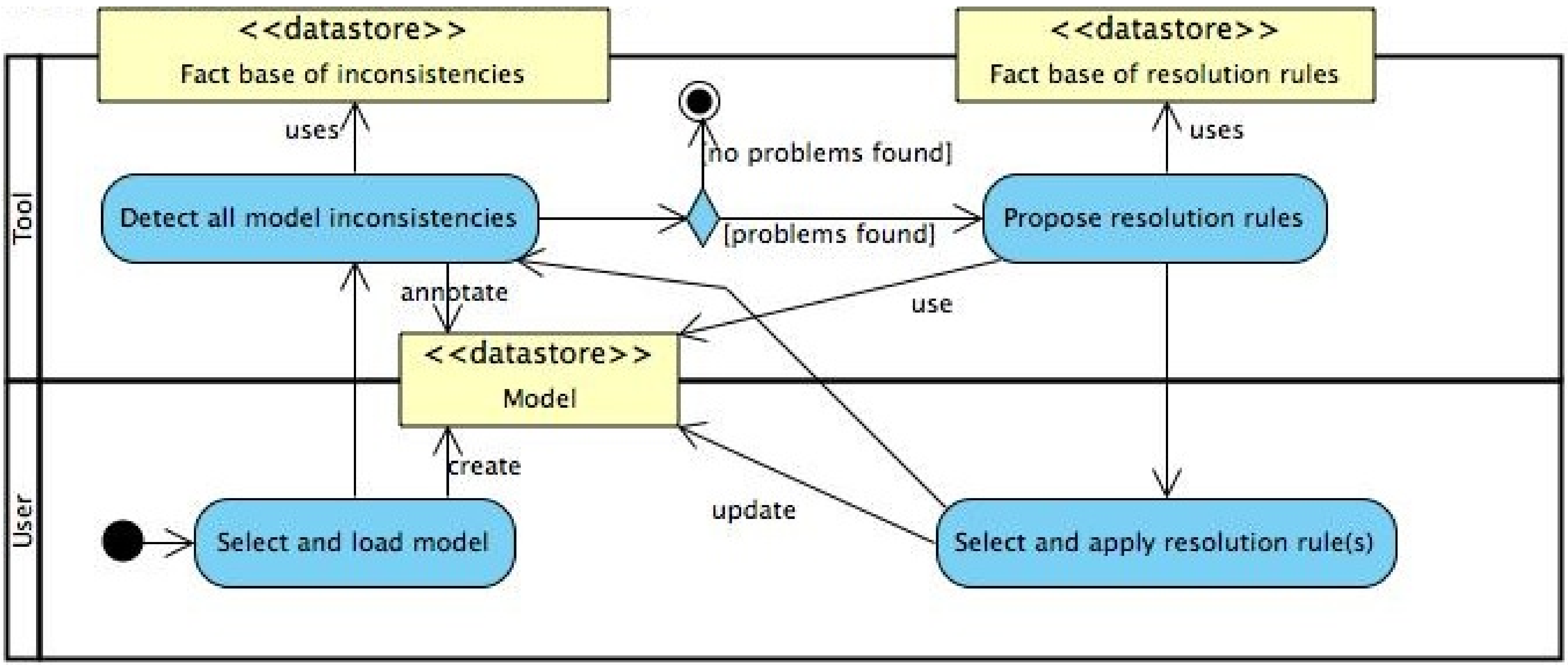
- Goal
  - Specify model inconsistencies and their resolution strategies as graph transformation rules
  - Use this formalisation to support the inconsistency management process
    - Automate the *detection* of inconsistencies
    - Interactively support the *resolution* of inconsistencies

  - Analyse transformation dependencies to optimise this process
    - Detect *sequential dependencies* between resolution rules
    - Detect *parallel conflicts* between resolution rules that cannot be applied together
    - Remove redundancy between resolution rules
    - Provide "optimal" resolution strategies

# Model inconsistency management

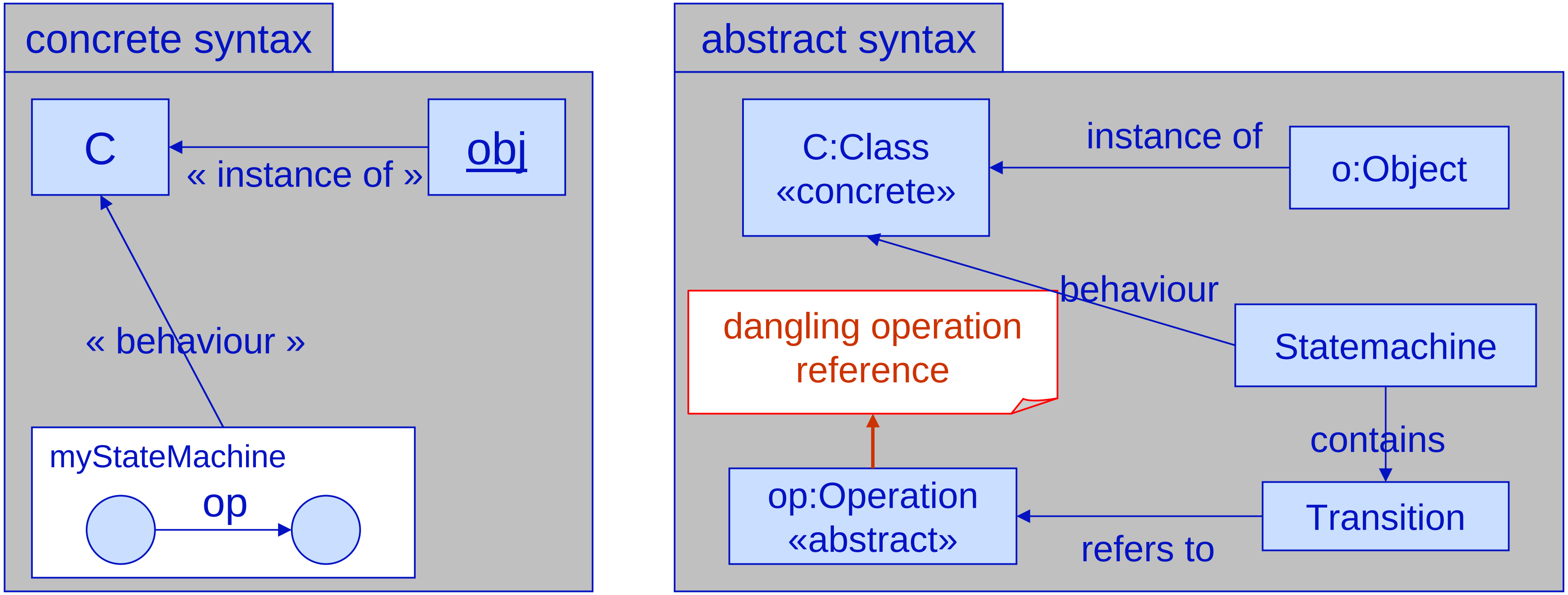
Syntactic and Semantic Integration of Visual Modelling Techniques

## Iterative inconsistency resolution process



# Example of inconsistency resolution

- Illustration of the ripple effect
  - Let’s start with a simple motivating example of an inconsistent model

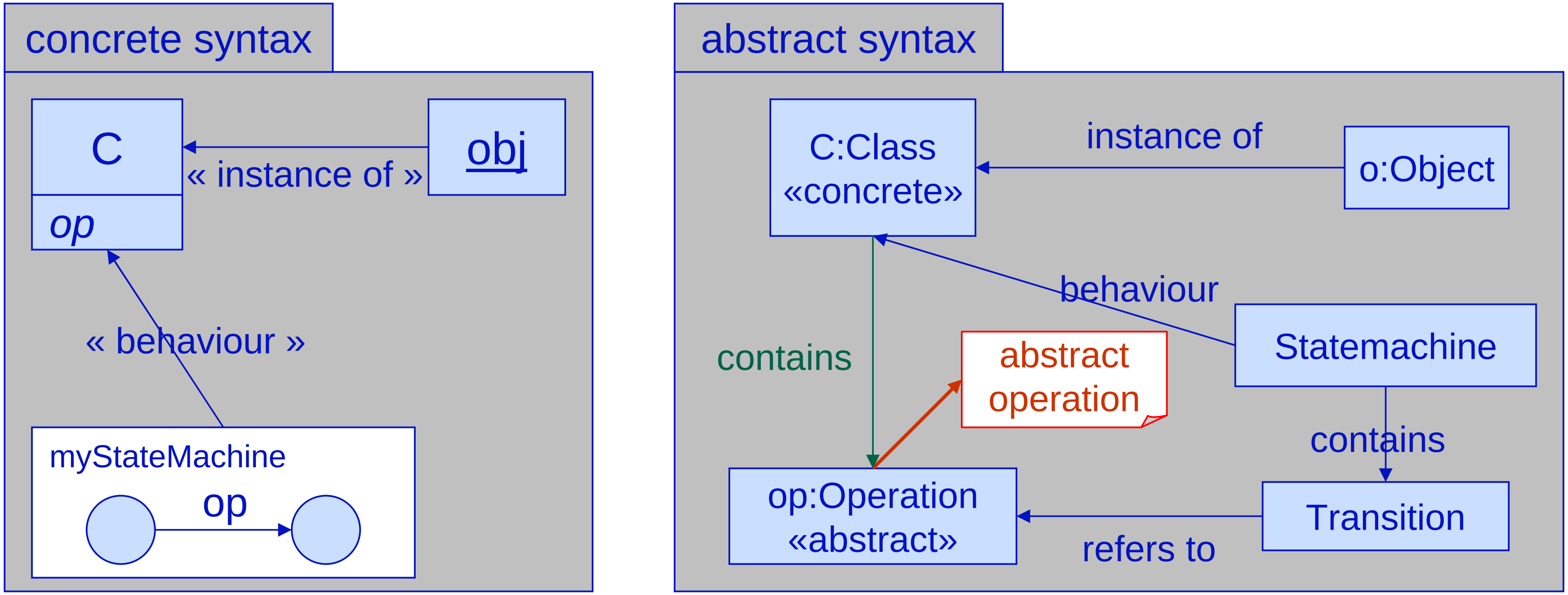


# Example of inconsistency resolution

Syntactic and Semantic Integration of Visual Modelling Techniques

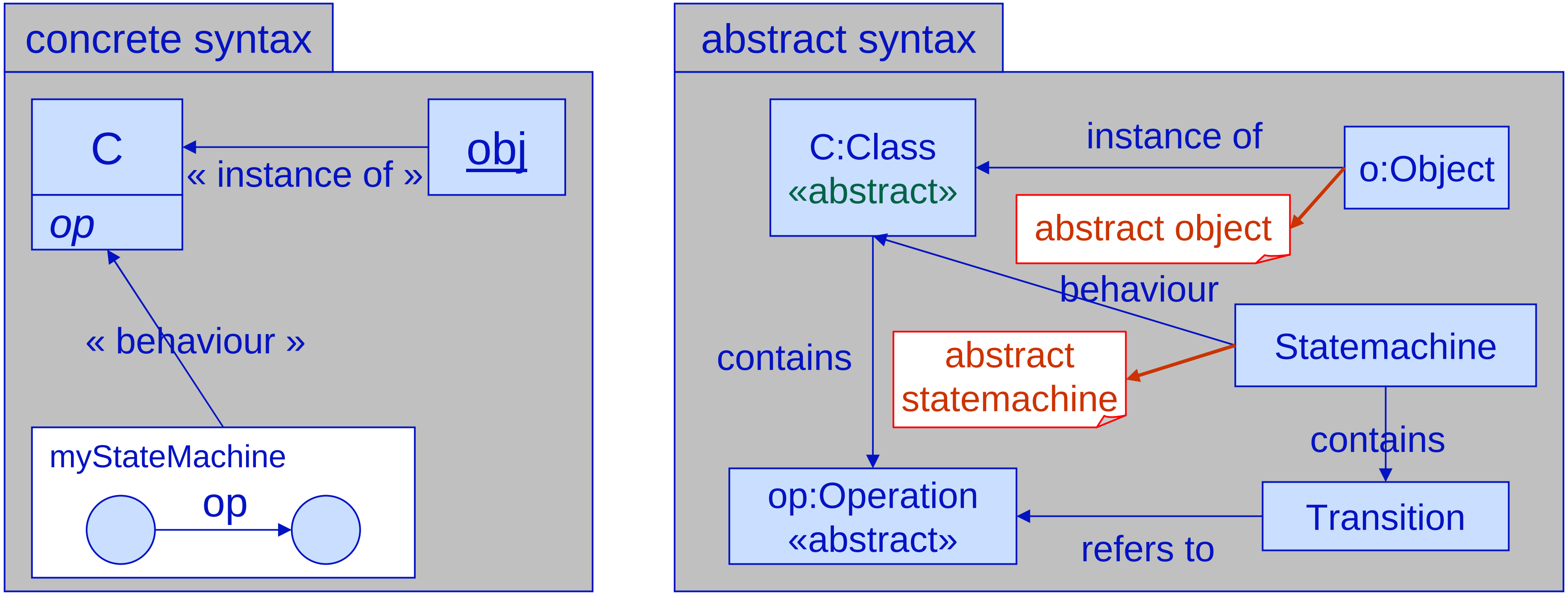
## Illustration of the ripple effect

- Resolution leads to a new inconsistency



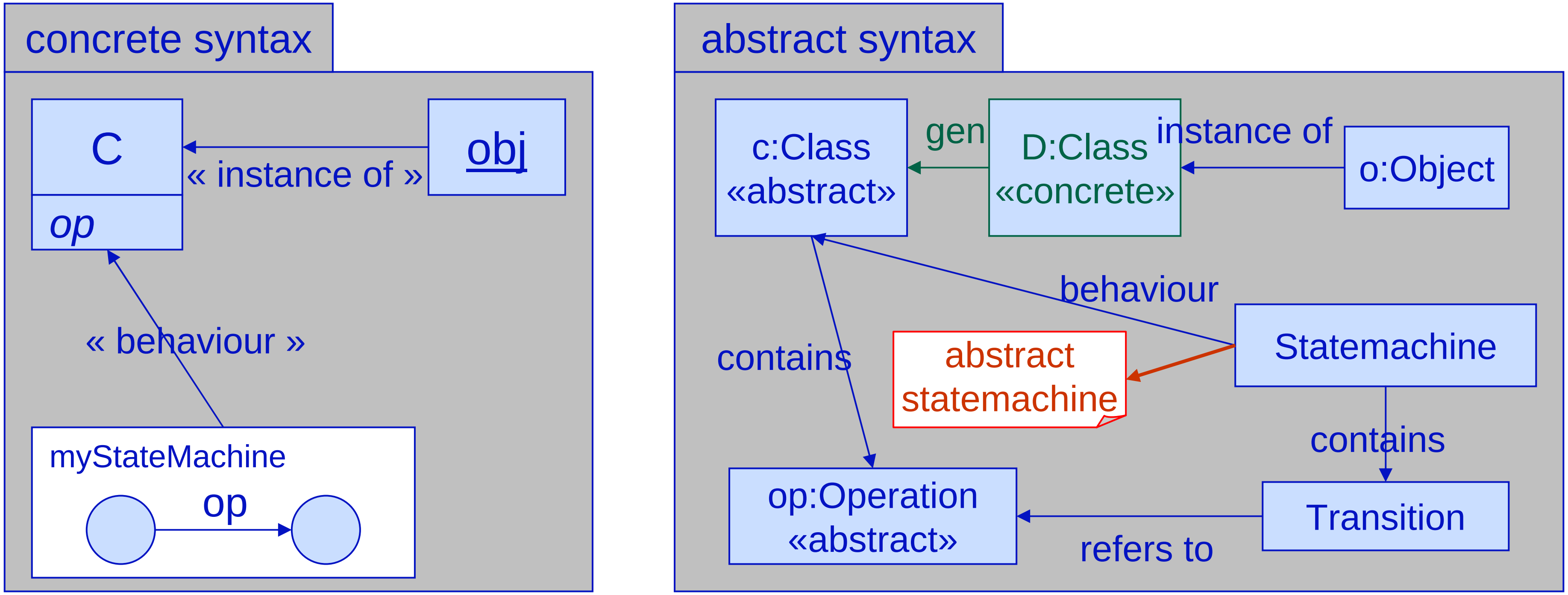
# Example of inconsistency resolution

- Illustration of the ripple effect
  - Resolution leads to 2 new inconsistencies



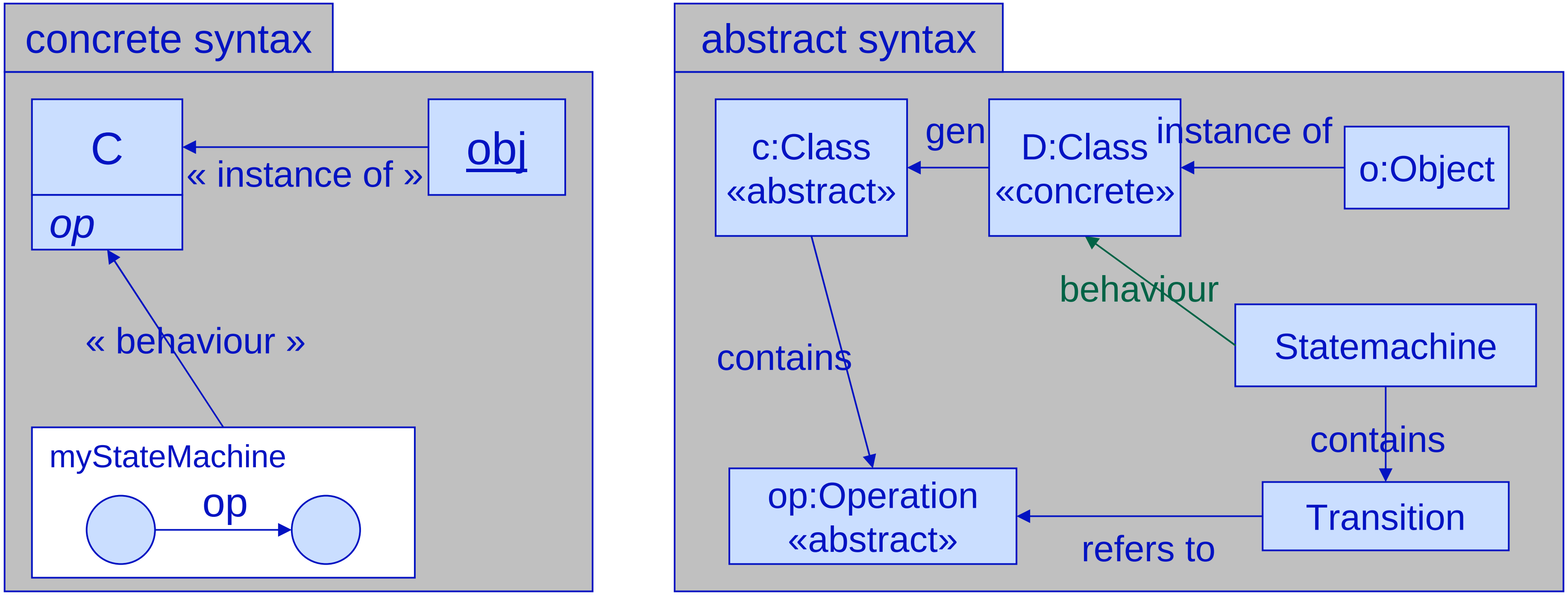
# Example of inconsistency resolution

- Illustration of the ripple effect
  - Resolution removes 1 inconsistency



# Example of inconsistency resolution

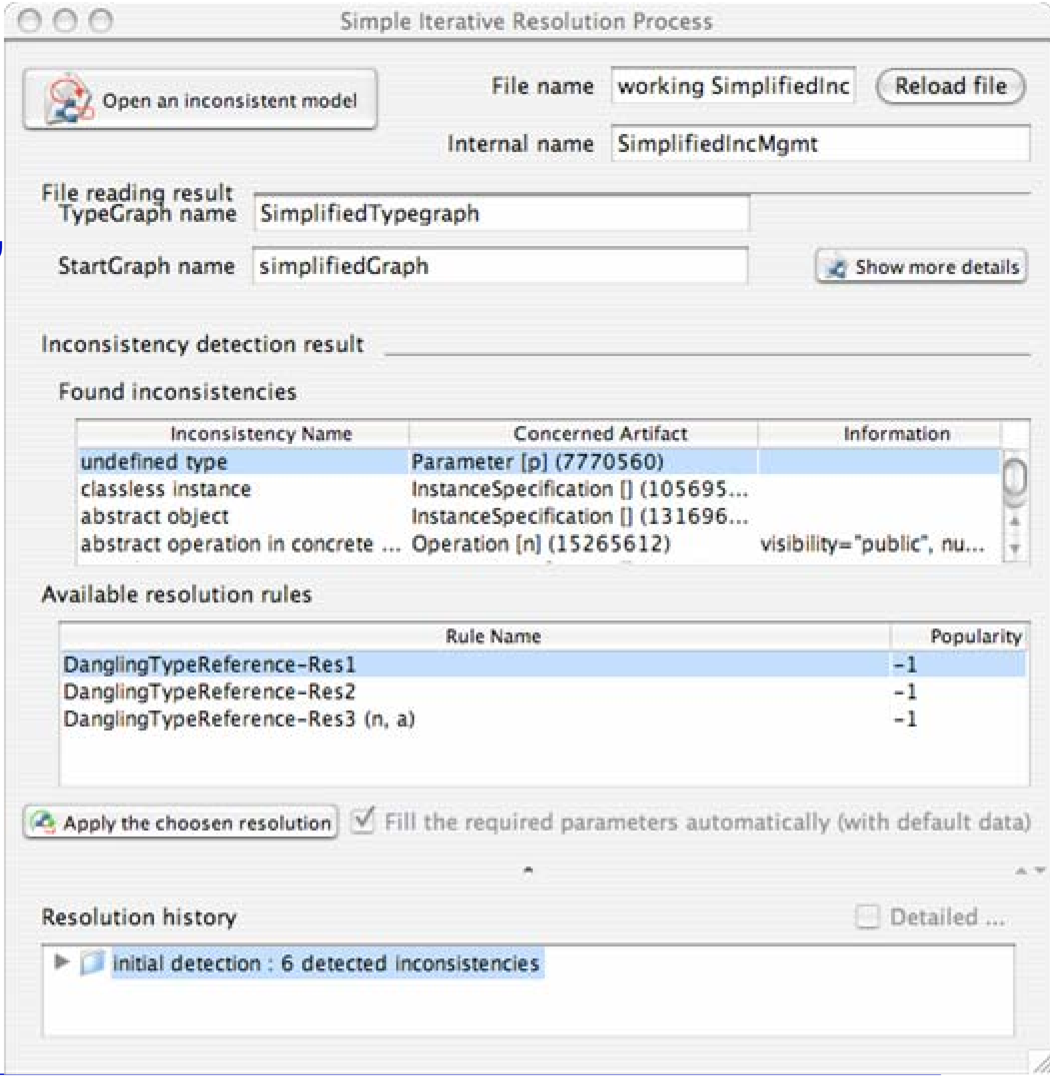
- Illustration of the ripple effect
  - Resolution finally removes last remaining inconsistency



# Tool support

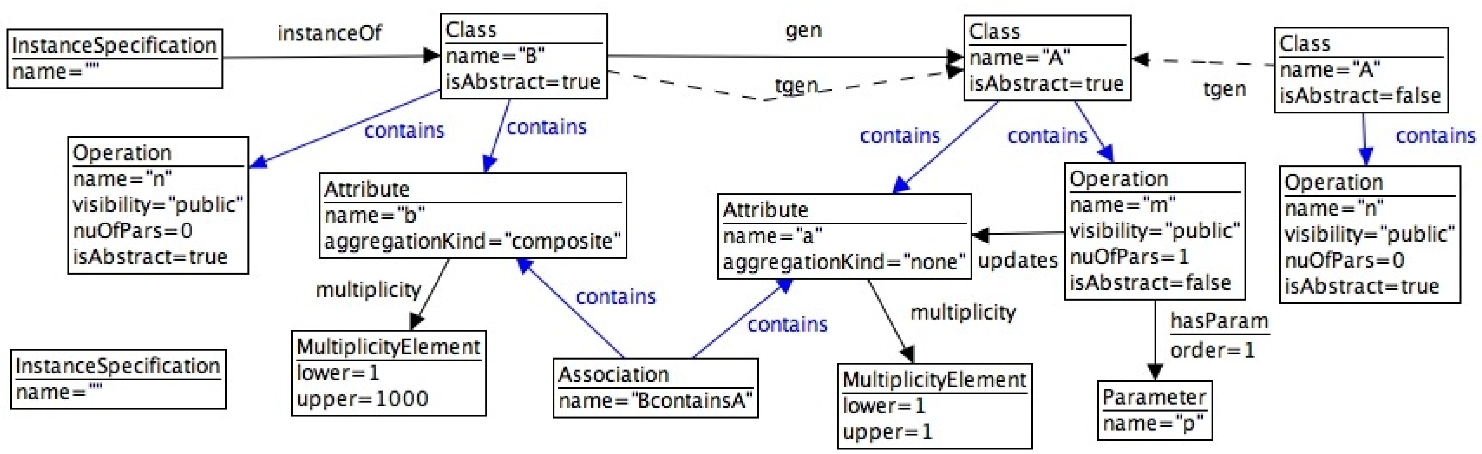


- SIRP tool
  - "Simple Iterative Resolution Process"
  - An interactive tool for selecting and resolving model inconsistencies



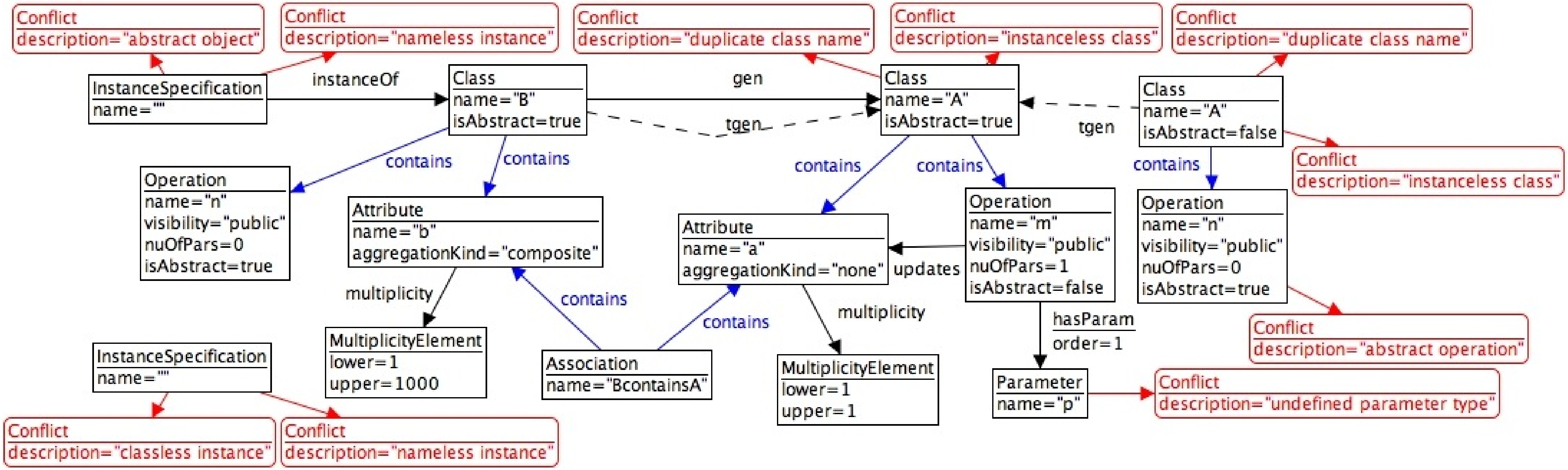
# Tool support

- SIRP tool in action
  - Before detecting any inconsistency



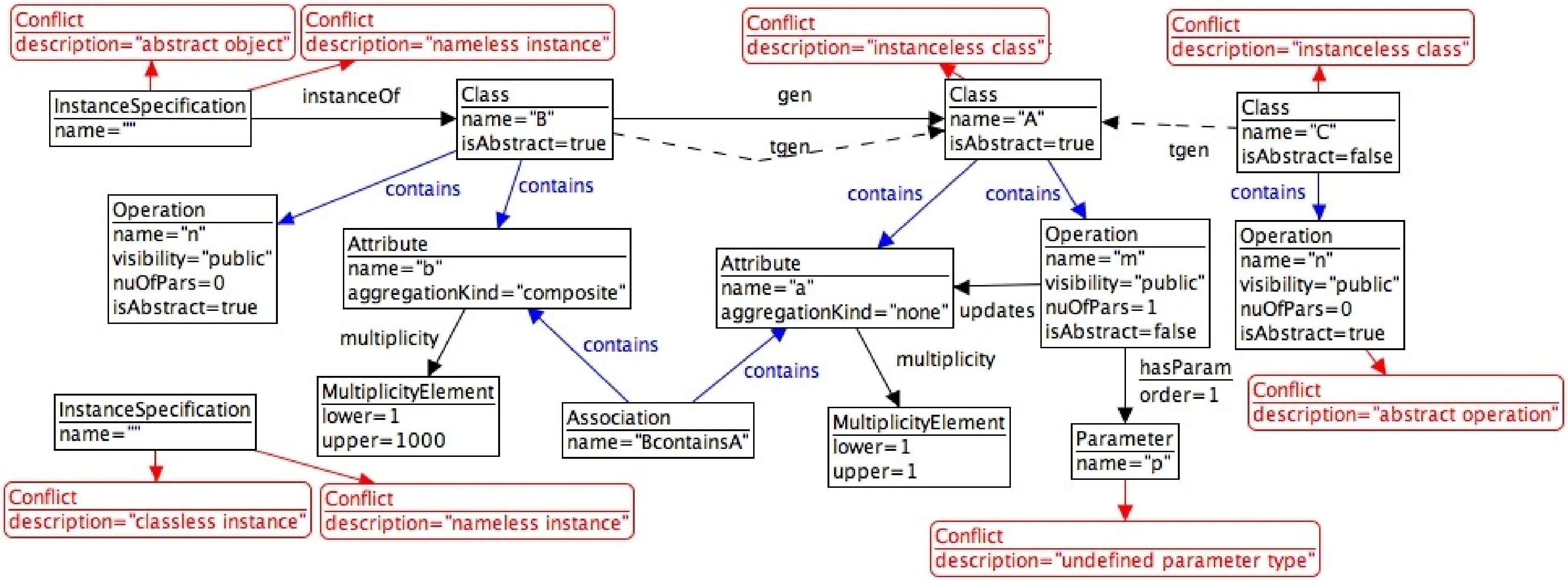
# Tool support

- SIRP tool in action
  - After detecting all inconsistencies



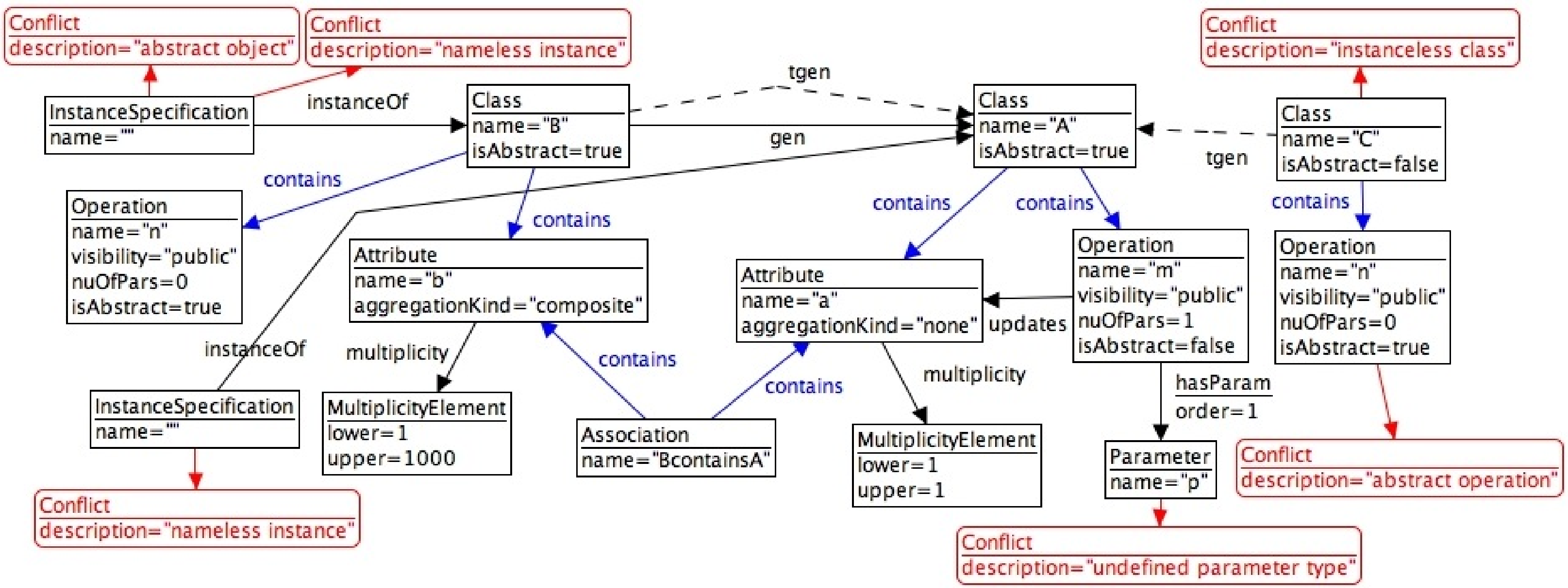
# Tool support

- SIRP tool in action
  - After resolving “*duplicate class name*”
    - Two occurrences of same inconsistency removed
    - Class renamed from “A” to “C”



Segravis
Syntactic and Semantic Integration of Visual Modelling Techniques

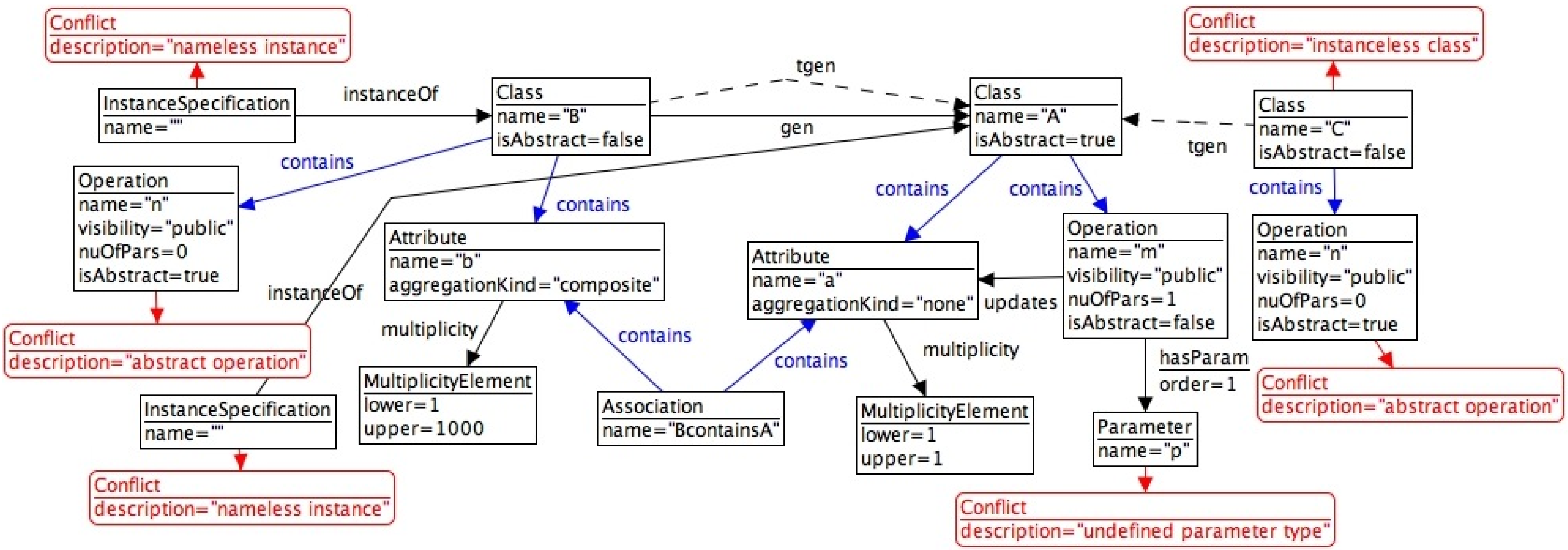
- SIRP tool in action
  - After resolving "*classless instance*"
    - One occurrence of "classless instance" removed
    - One occurrence of "instanceless class" removed



SegraVis
Syntactic and Semantic Integration of Visual Modelling Techniques

# SIRP tool in action

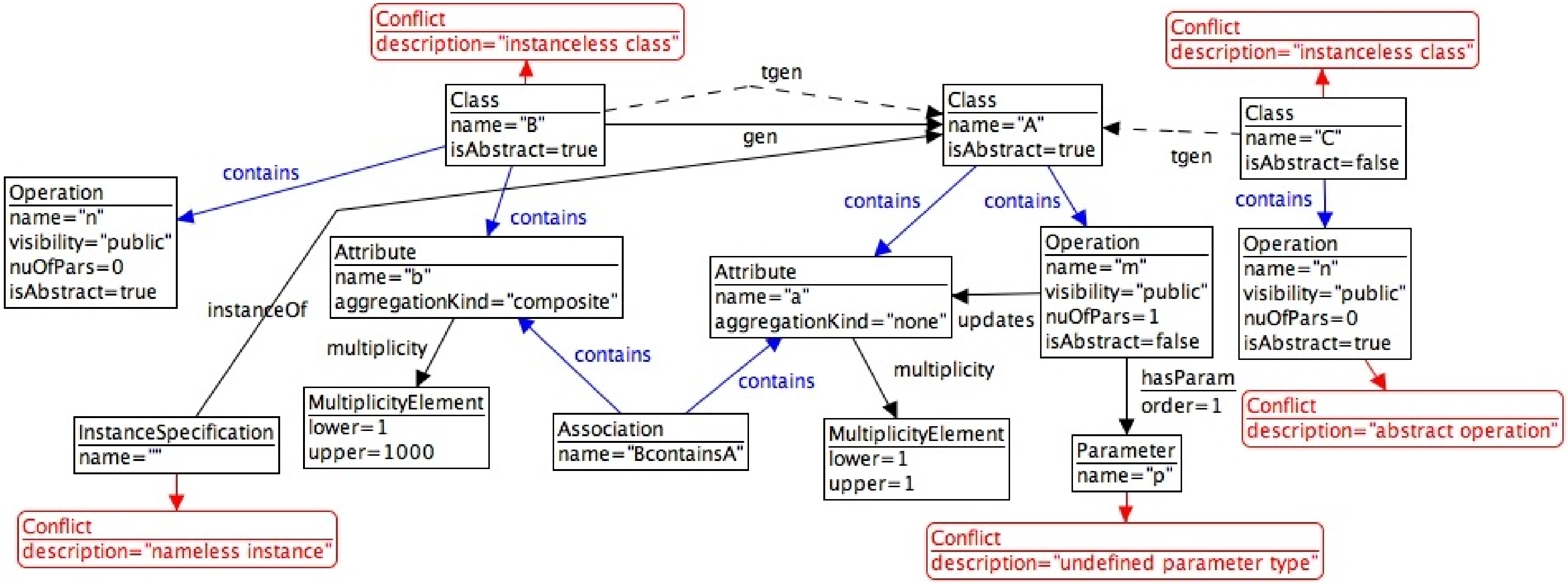
- After resolving “*abstract object*” (first try)
  - One occurrence of “abstract object” removed
  - One occurrence of “abstract operation” added !



# Tool support

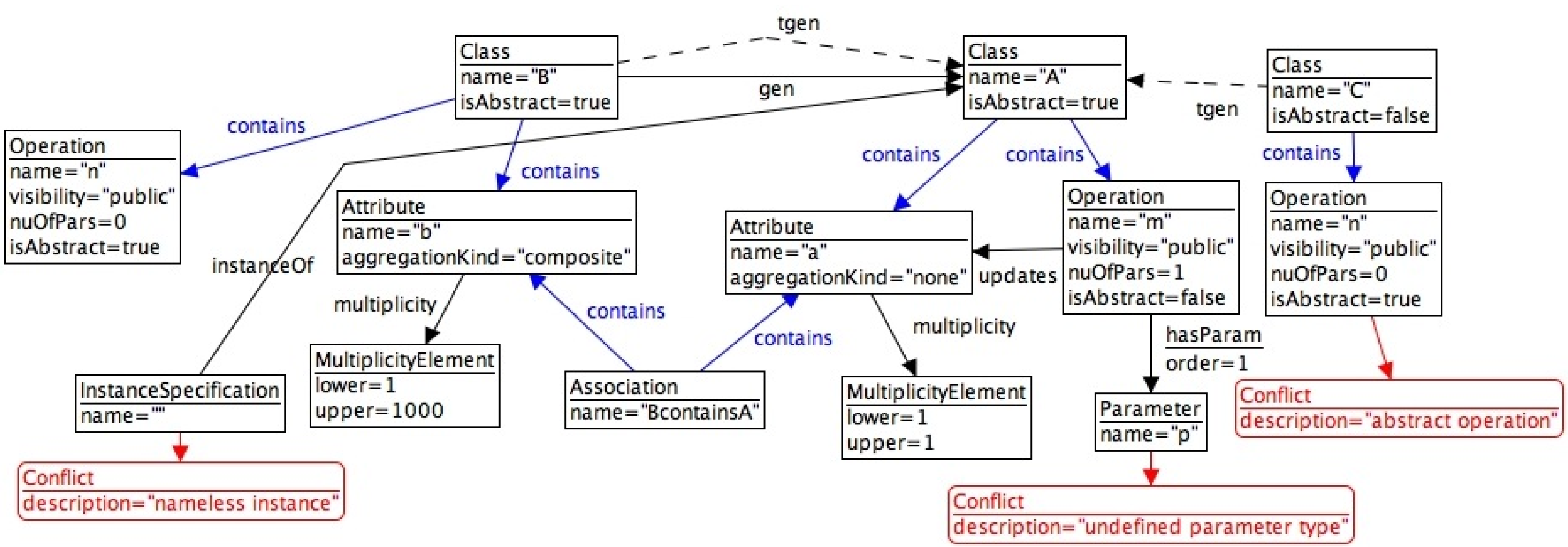
SegraVis
Syntactic and Semantic Integration of Visual Modelling Techniques

- SIRP tool in action
  - After resolving "*abstract object*" (second try)
    - One occurrence of "abstract object" removed
    - One occurrence of "nameless instance" removed
    - One occurrence of "instanceless class" added



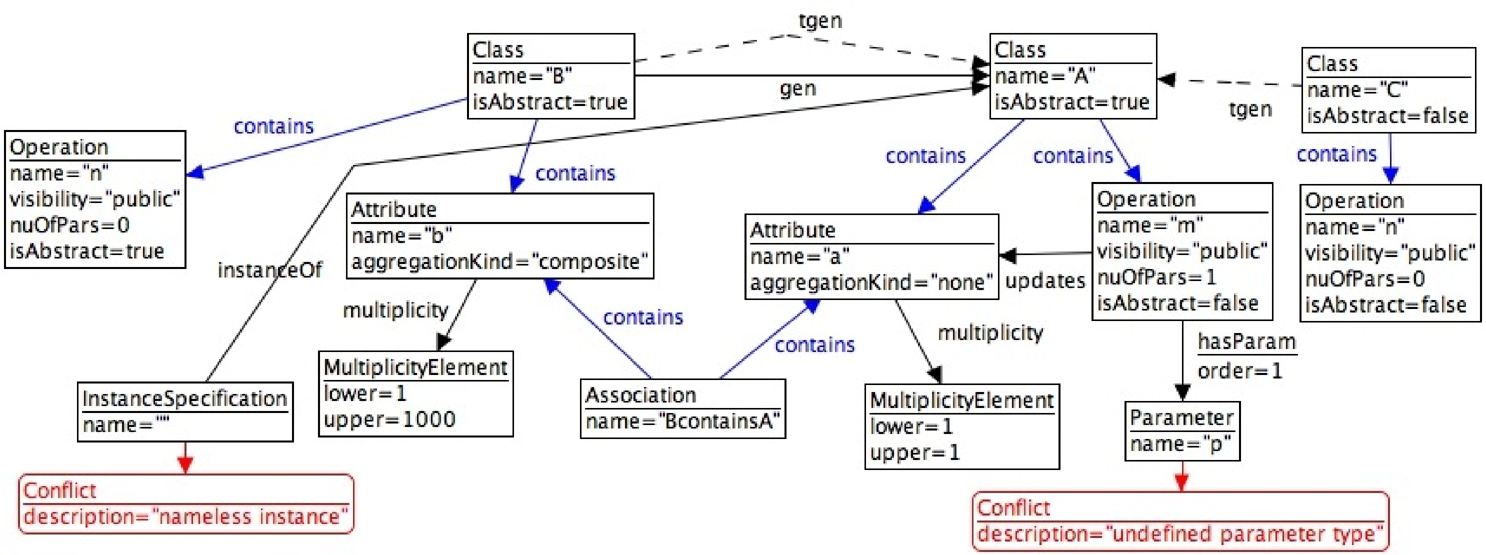
## SIRP tool in action

- After disabling the "*instanceless class*" rule
  - Two occurrences of "instanceless class" *ignored*



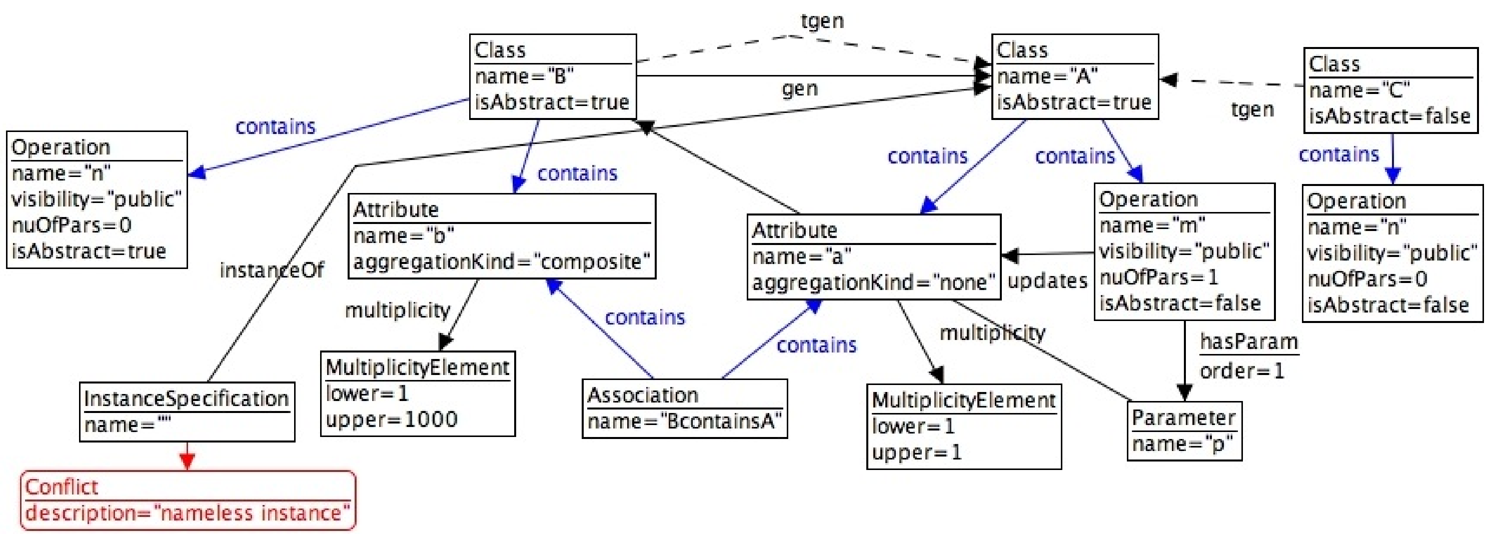
# Tool support

- SIRP tool in action
  - After resolving "*abstract operation*"
    - One occurrence of "abstract operation" removed



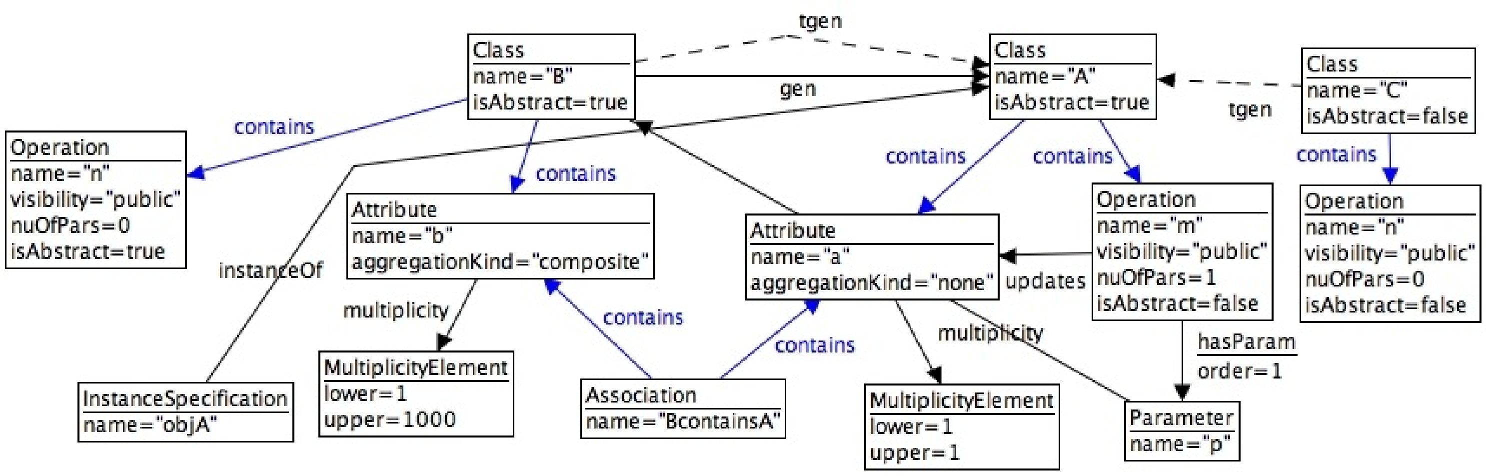
# SIRP tool in action

- After resolving “*undefined parameter type*”
  - One occurrence of “undefined parameter type” removed



# SIRP tool in action

- After resolving "*nameless instance*"
  - One occurrence of "nameless instance" removed
  - No more remaining inconsistencies !



# Tool support



- This tool relies on the underlying mechanism of graph transformation
  - for detecting inconsistencies
  - for proposing resolution rules
  - for analysing which of the proposed resolution rules is most appropriate
- But how does this all work?

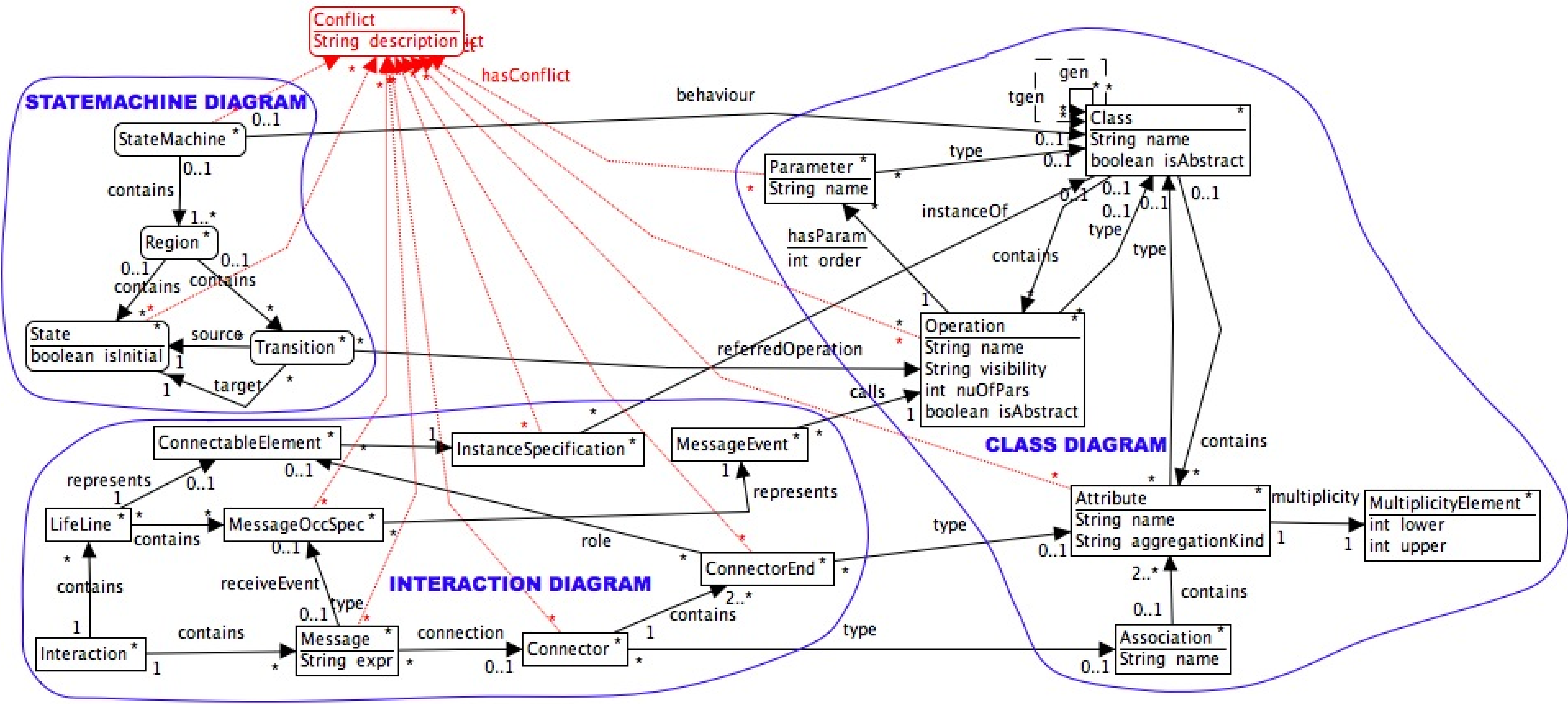
# Tool support

- The tool has been implemented on top of the AGG engine (version 1.4)
  - AGG is a general-purpose graph transformation tool
- We used AGG in the following way
  - specify the *UML metamodel* as a type graph
  - specify the *models* as graphs
  - *detect* and *resolve* model inconsistencies by means of graph transformation rules
  - analyse *mutual exclusion* relationships and *sequential dependencies* between inconsistency resolutions by means of critical pair analysis

# Step 1: Specify the metamodel

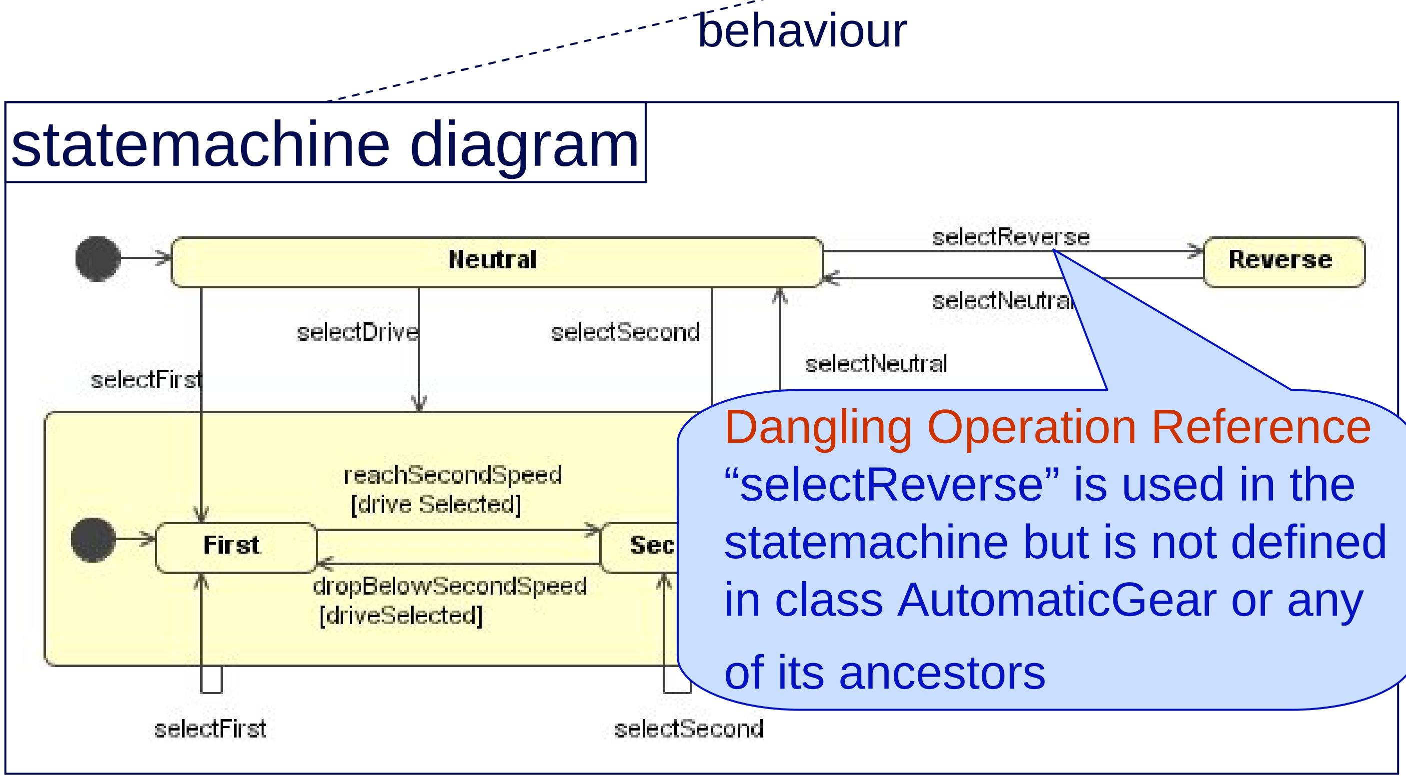
Syntactic and Semantic Integration of Visual Modelling Techniques

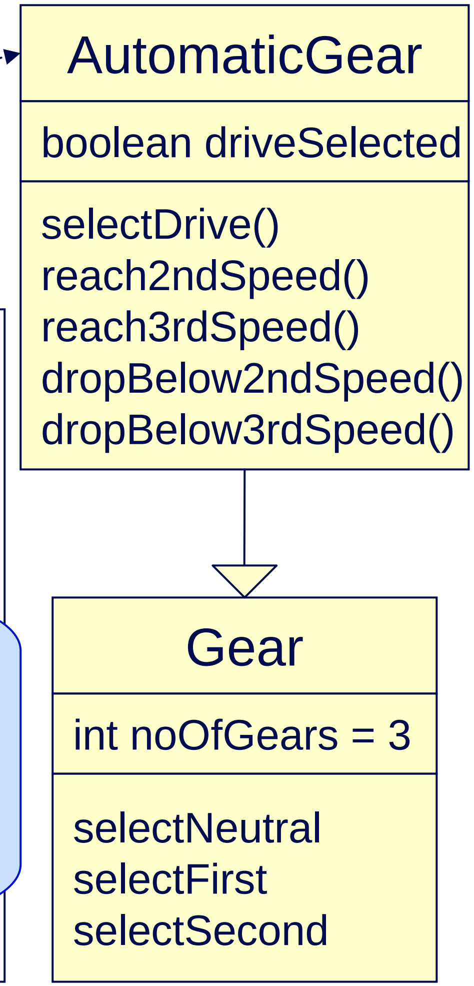
## AGG type graph



# Specify the UML model

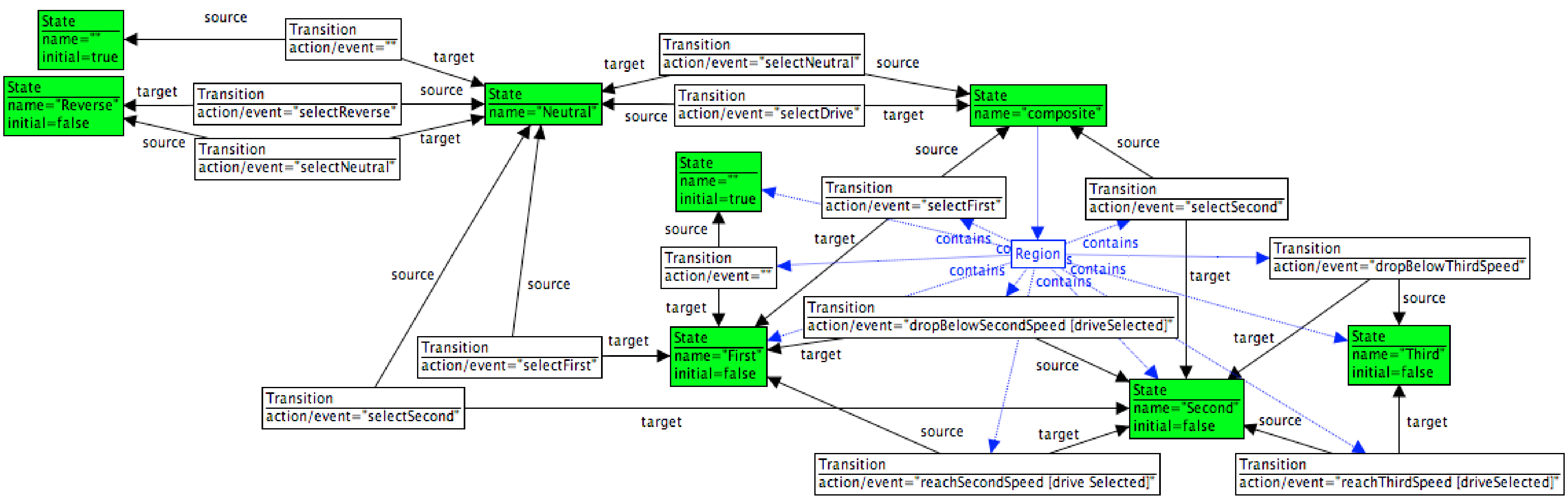
## Example of an inconsistent UML model



# Represent the UML model as a graph

- Automatic generation of the corresponding graph representation for the UML state machine
  - This graph conforms to the type graph specified before



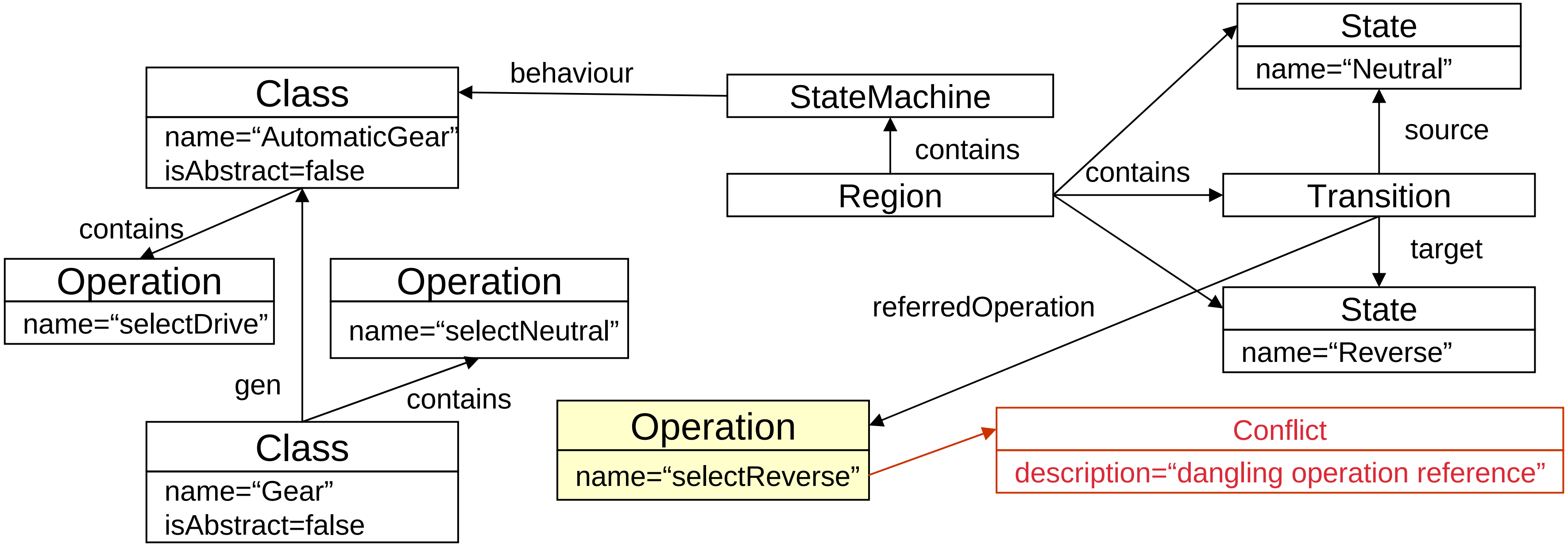
# Step 2: Classify model inconsistencies

| | |
|---|---|
| Dangling Type Reference | An operation has one or more parameters whose types are not specified |
| Classless Instance | A model contains an instance specification that is not linked to a class |
| Abstract Object | A model contains an instance specification that is an instance of an abstract class that does not have any concrete subclasses. |
| Abstract Operation | An abstract operation is defined in a concrete class. |
| Abstract State Machine | A state machine expresses the behaviour of an abstract class that does not have any concrete subclasses. |
| Cyclic Composition | A class contains at least one instance of its subclasses through a composition relationship that may lead to an infinite containment of instances of the affected classes. |
| Dangling Operation Reference | A state machine contains a transition that refers to an operation that does not belong to any class (or that belongs to a different class than the one whose behaviour is expressed by the state machine). |
| Transition Without Operation | A transition does not have a referred operation attached to it. |

# Example of a model inconsistency

## Dangling operation reference

### Using graph representation



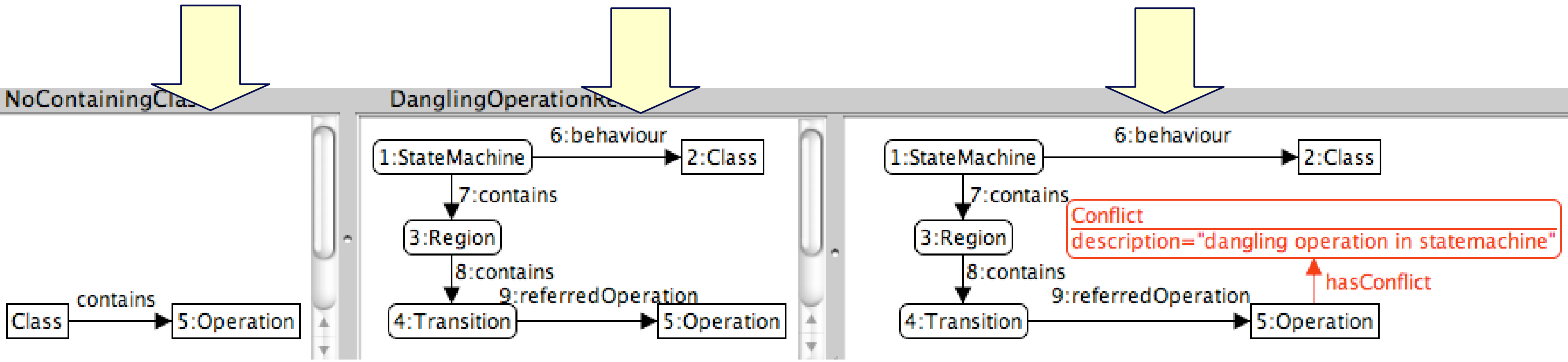
# Step 3: Specify model inconsistencies

- Using graph transformation
  - Example: Dangling operation reference

negative application condition (NAC)

left-hand side (LHS)

right-hand side (RHS)

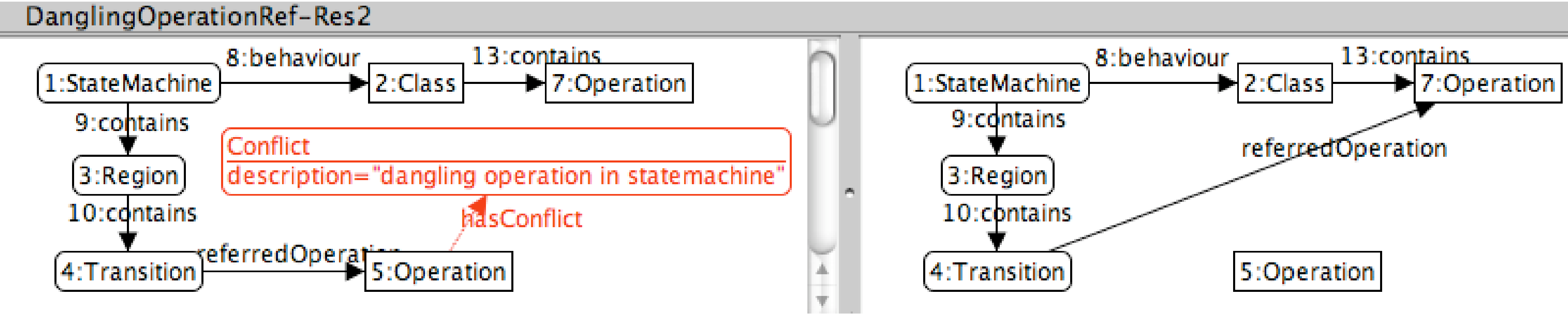


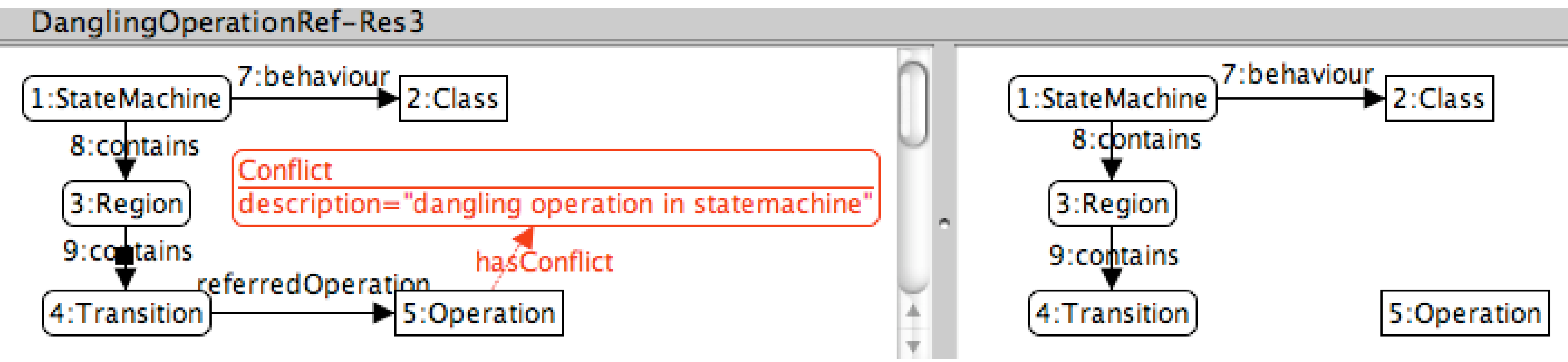
# Step 4: Identify inconsistency resolutions

| | | |
|---|---|---|
| Dangling Operation Ref. | Res1 | Add the operation to the class (or one of its ancestor classes) whose behaviour is described by the state machine. |
| | Res2 | Let the transition refer to an existing operation belonging to the class (or one of its ancestors) whose behaviour is described by the state machine. |
| | Res3 | Remove the reference from the transition to the operation. |
| | Res3 | Remove the transition. |
| Classless Instance | Res1 | Remove the instance specification. |
| | Res2 | Link the instance specification to an existing class. |
| | Res3 | Link the instance specification to a new class. |
| Abstract Object | Res1 | Change the abstract class into a concrete one. |
| | Res2 | Add a concrete descendant of the abstract class, and redirect the outgoing instance-of relation of the instance specification to this concrete descendant. |
| | Res3 | Remove the instance specification. |

# Step 5: Specify inconsistency resolutions

- Using graph transformation
  - Example: Dangling Operation Reference



# Step 6: Detect mutually conflicting resolution rules

Syntactic and Semantic Integration of Visual Modelling Techniques

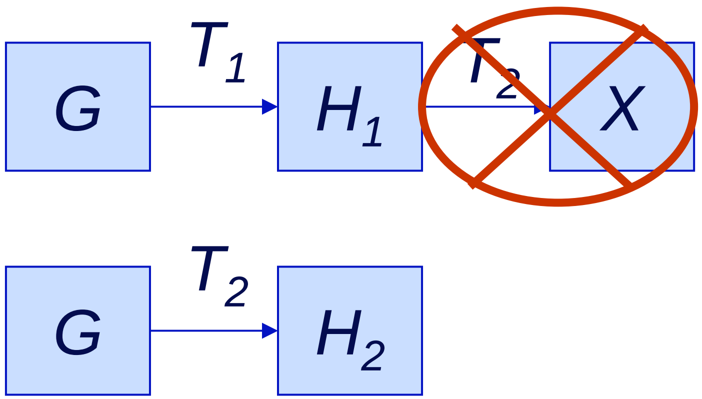
- Some resolution rules cannot be jointly applied (parallel conflict!)
  - Conflict graph can be generated by means of critical pair analysis

# Step 6: Detect mutually conflicting resolution rules

## Informal definition (parallel conflict)

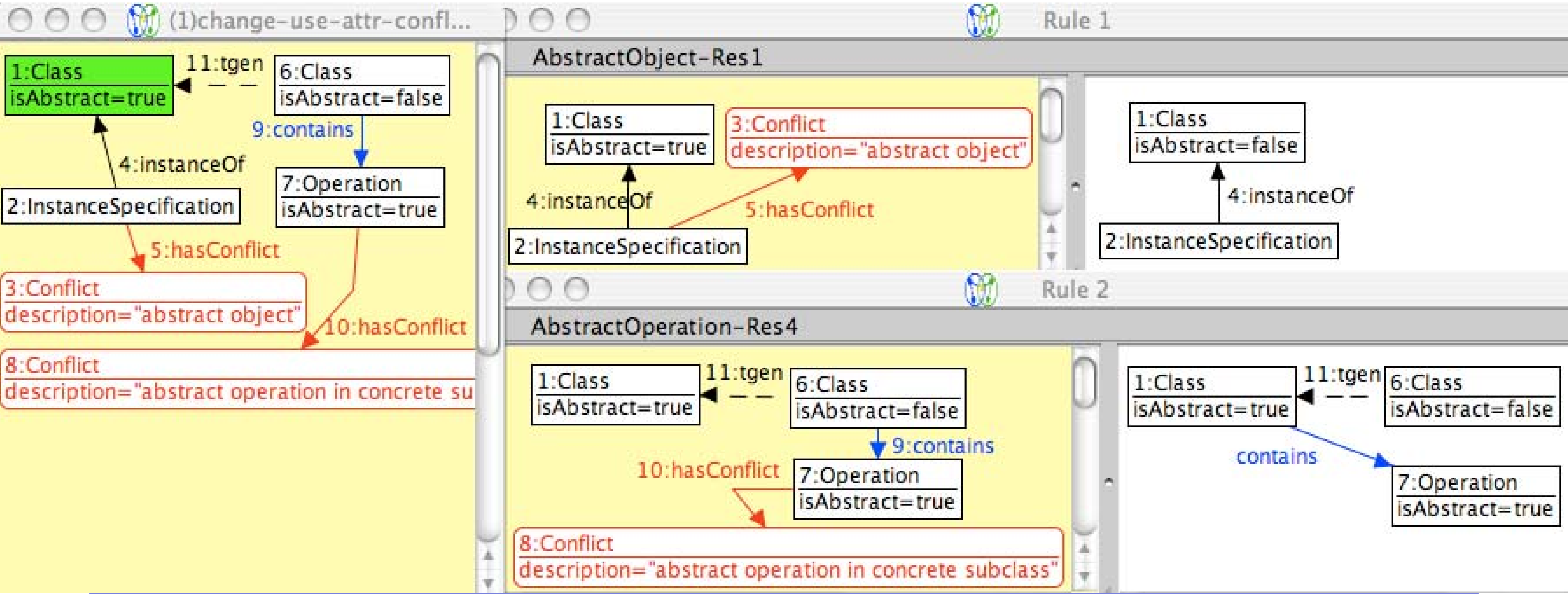
$T_1$ and $T_2$ form a *critical pair* if

- they can both be applied to the same initial graph $G$
- applying $T_1$ prohibits application of $T_2$



# Step 6: Detect mutually conflicting resolution rules

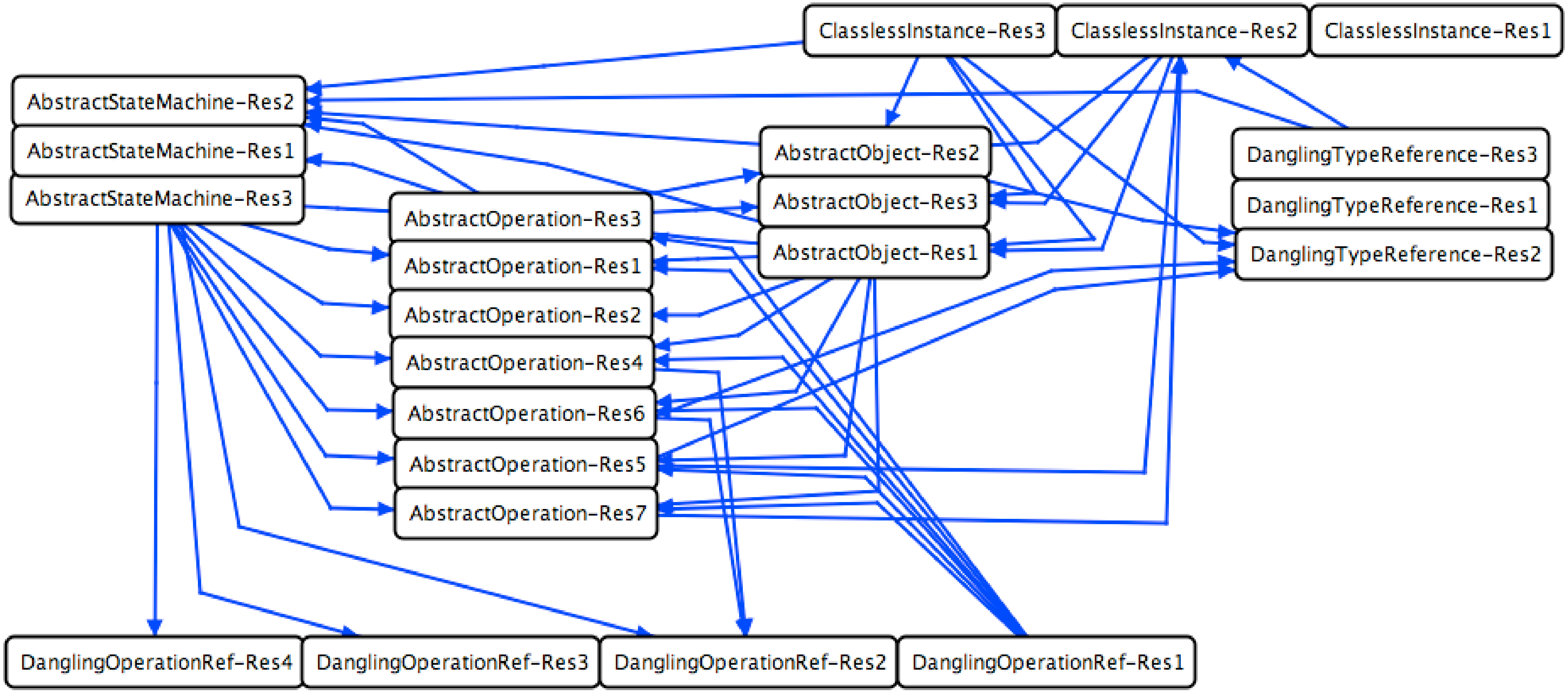
- Example of a critical pair detecting a *parallel conflict* between resolution rules
  - the resolution rules are not jointly applicable



# Step 7: Detect / analyse sequential dependencies

Syntactic and Semantic Integration of Visual Modelling Techniques

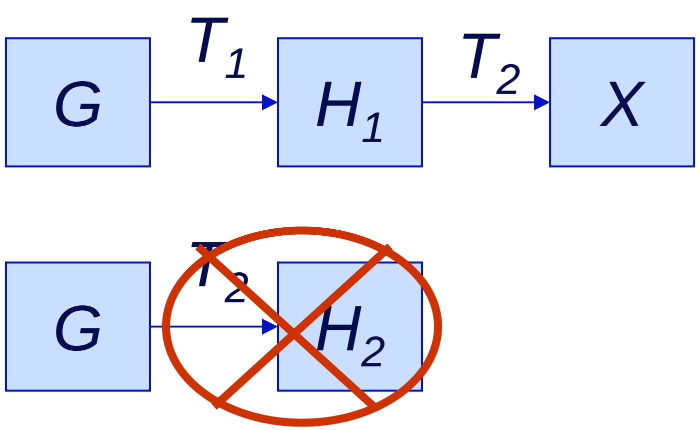
- Some resolution rules may give rise to opportunities for applying other resolution rules
  - Graph of sequential dependencies generated by AGG



# Informal definition (sequential dependency)

$T_2$ *sequentially depends* on $T_1$ if

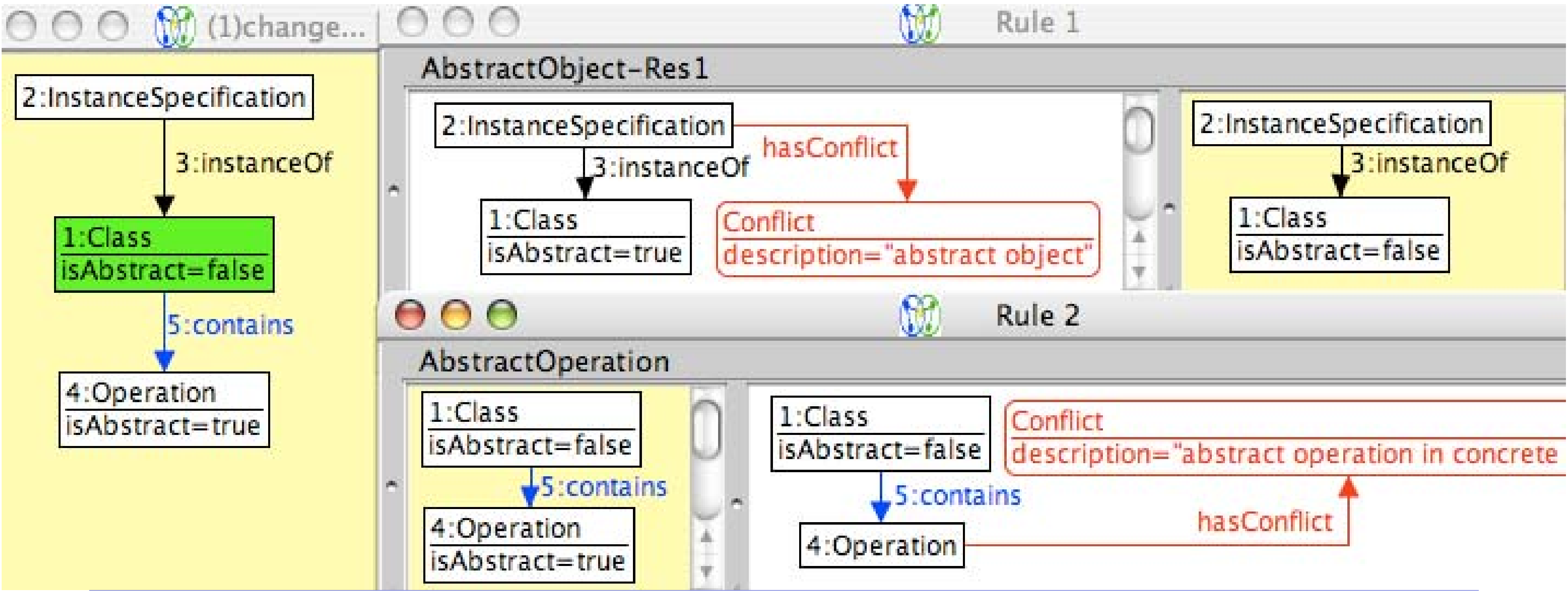
- $T_1$ can be applied to $G$ but $T_2$ cannot
- applying $T_1$ triggers application of $T_2$



# Step 7: Detect / analyse sequential dependencies

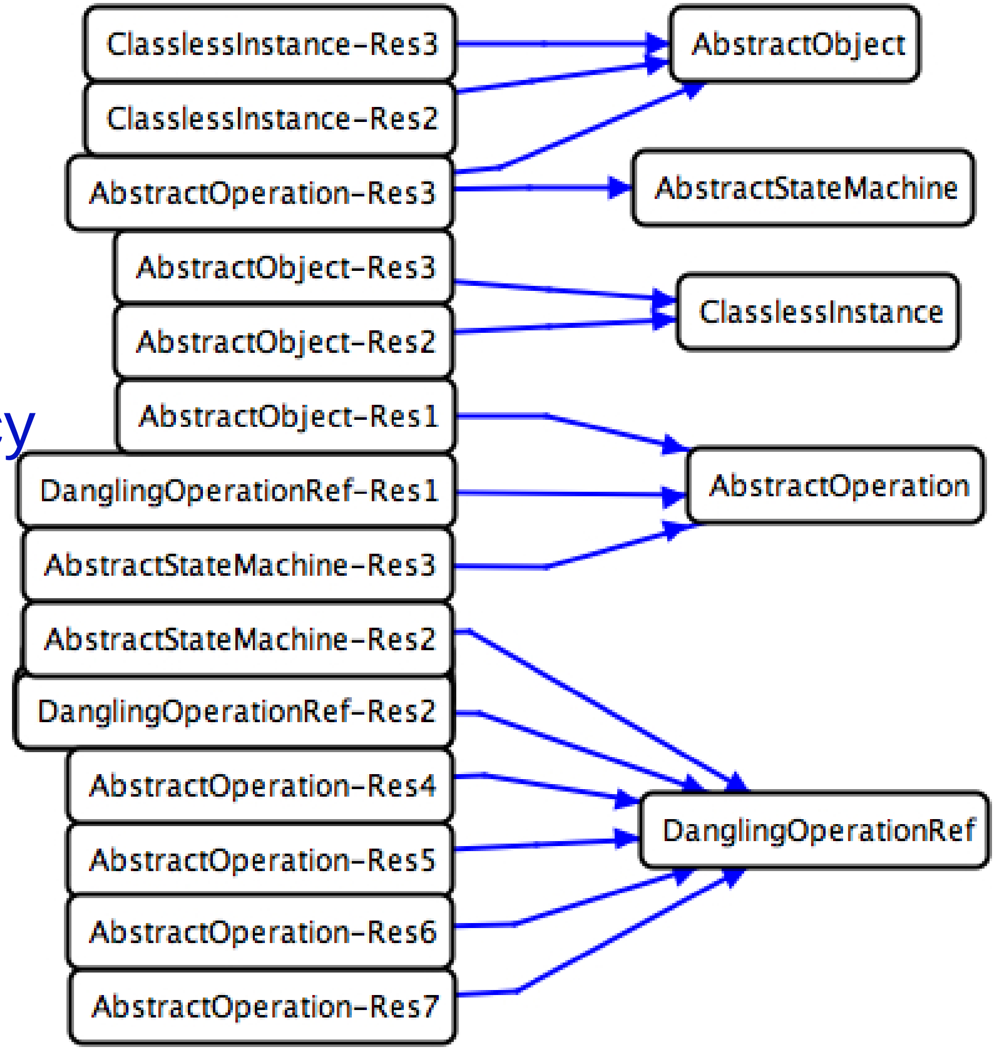
Syntactic and Semantic Integration of Visual Modelling Techniques

- Example of a sequential dependency representing an *induced inconsistency*
  - resolution rule gives rise to a new inconsistency



# Step 7: Detect / analyse sequential dependencies

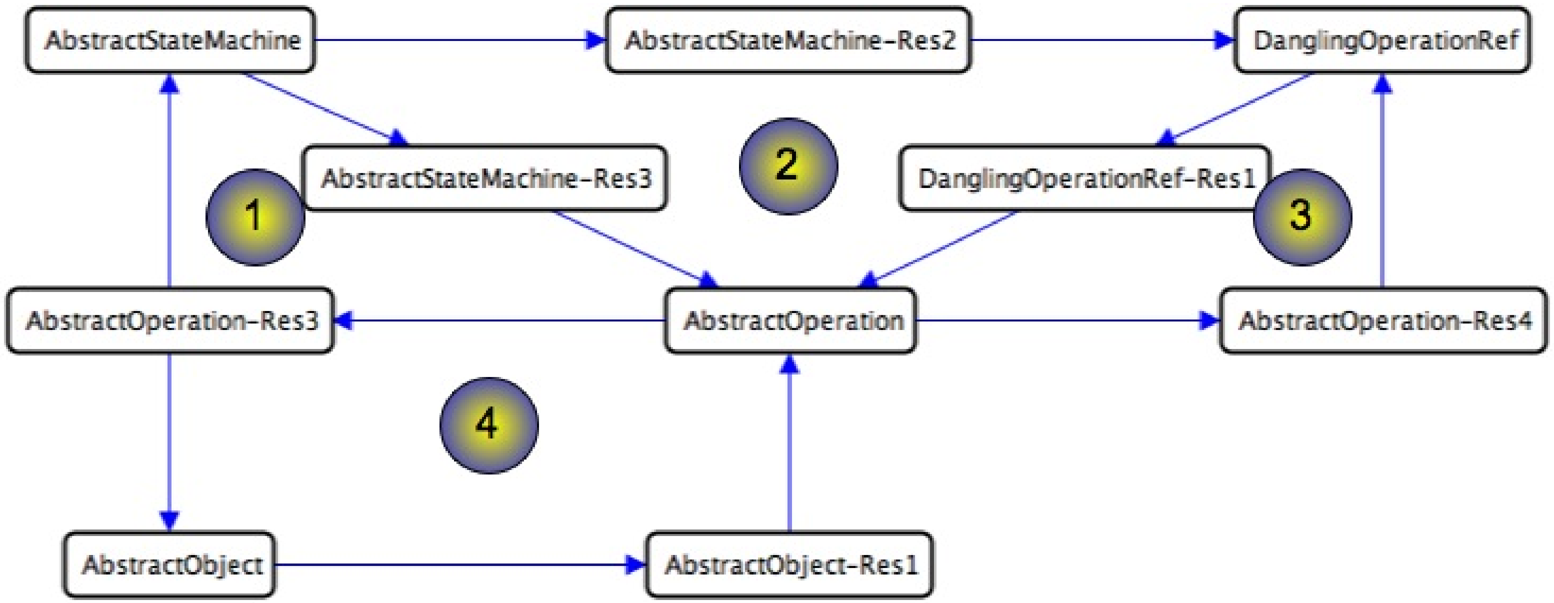
- Some resolution rules may give rise to new model inconsistencies
  - Can be detected by analysing the dependency graph



# Step 7: Detect / analyse sequential dependencies

SegraVis
Syntactic and Semantic Integration of Visual Modelling Techniques

- We can use the dependency graph to detect potential cycles in the resolution process




- Cycles should be avoided, since this implies that the resolution process may continue forever…

# Step 8: Tool support revisited

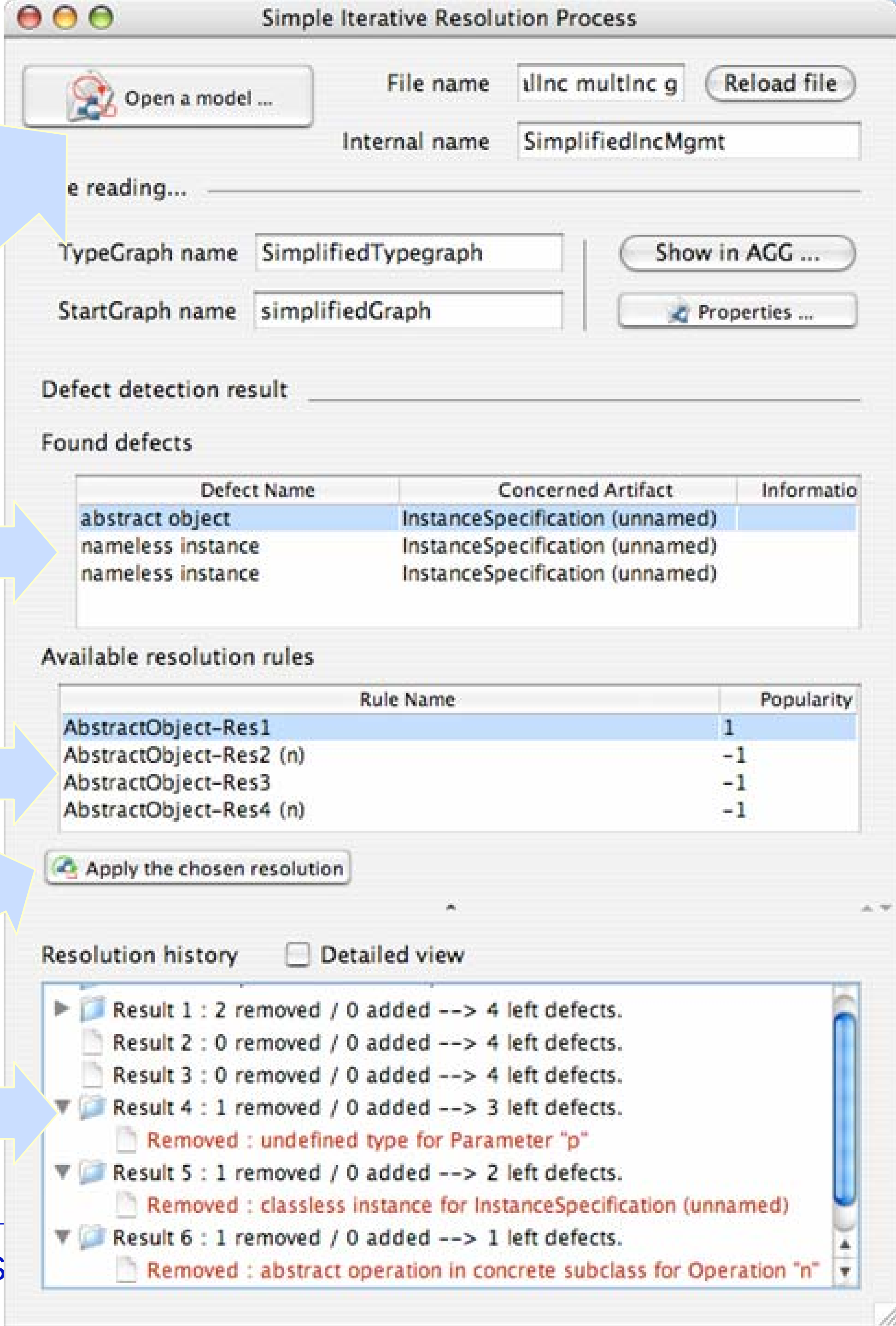
open graph and apply all *detection rules*

list all found inconsistencies (Conflict nodes in the graph)

list all *resolution rules* for selected inconsistency

*apply* selected resolution rule to the graph

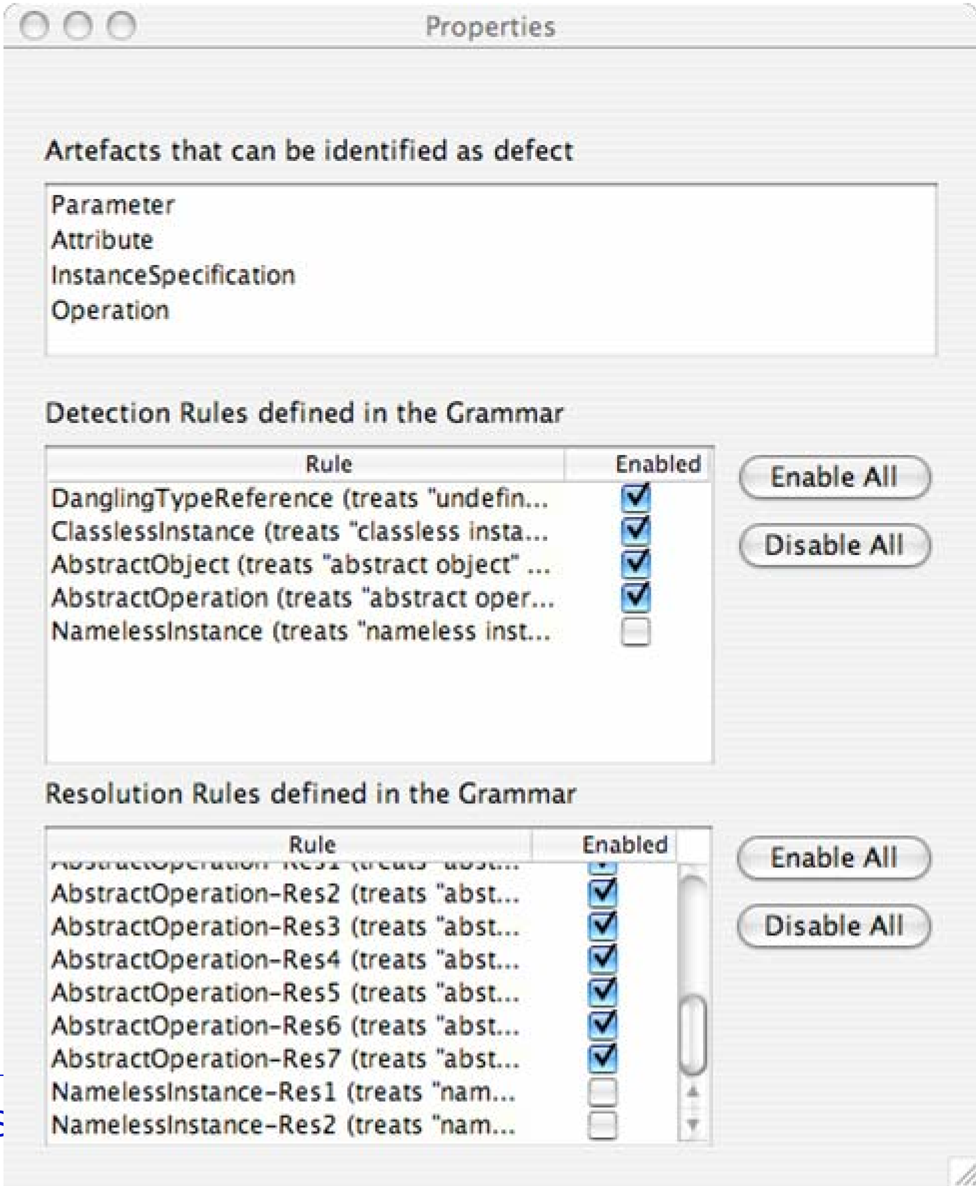
display resolution history



# Step 8: Tool support revisited

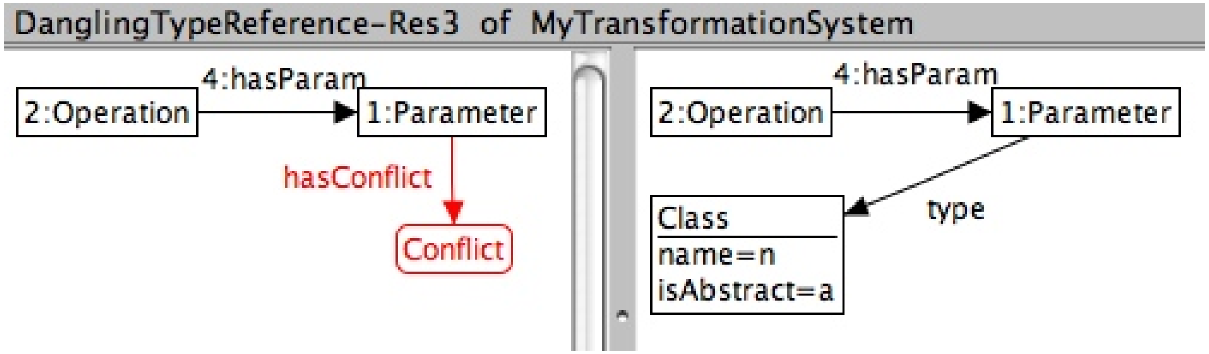


- Detection or resolution rules may be disabled by the user

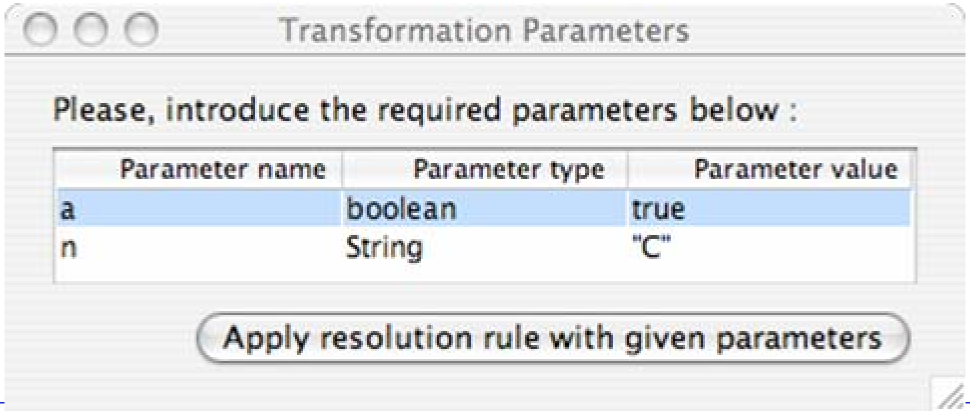


# Step 8: Tool support revisited

- A resolution rule may be parametrised
  - e.g. DanglingTypeRef-Res3(n,a)

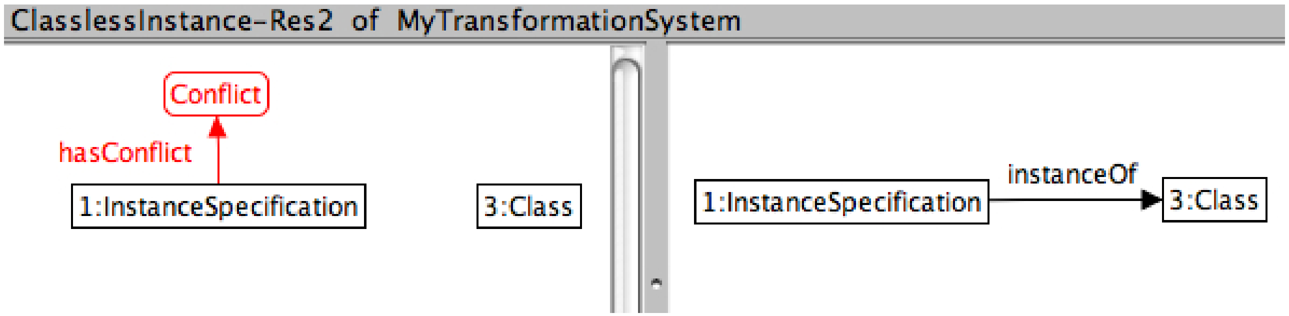


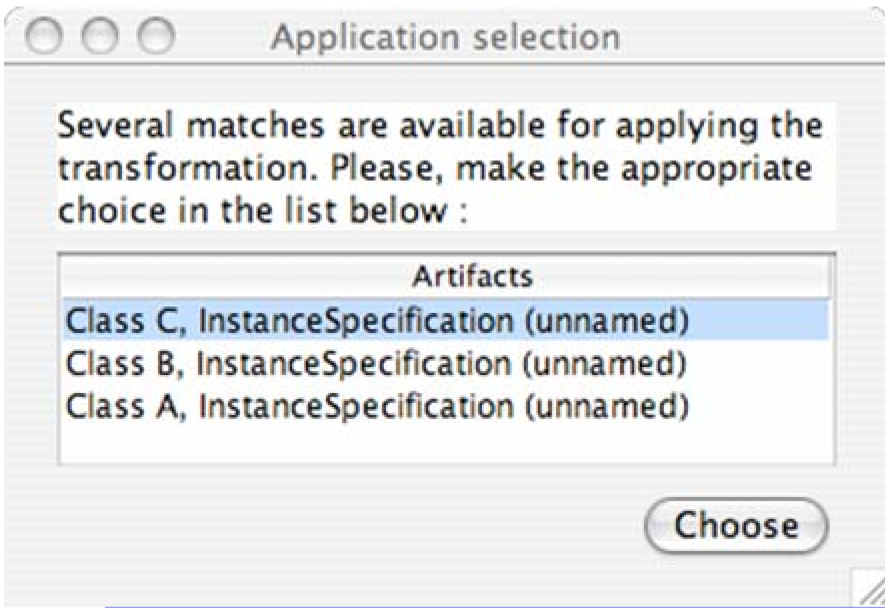

  - User needs to provide necessary input values

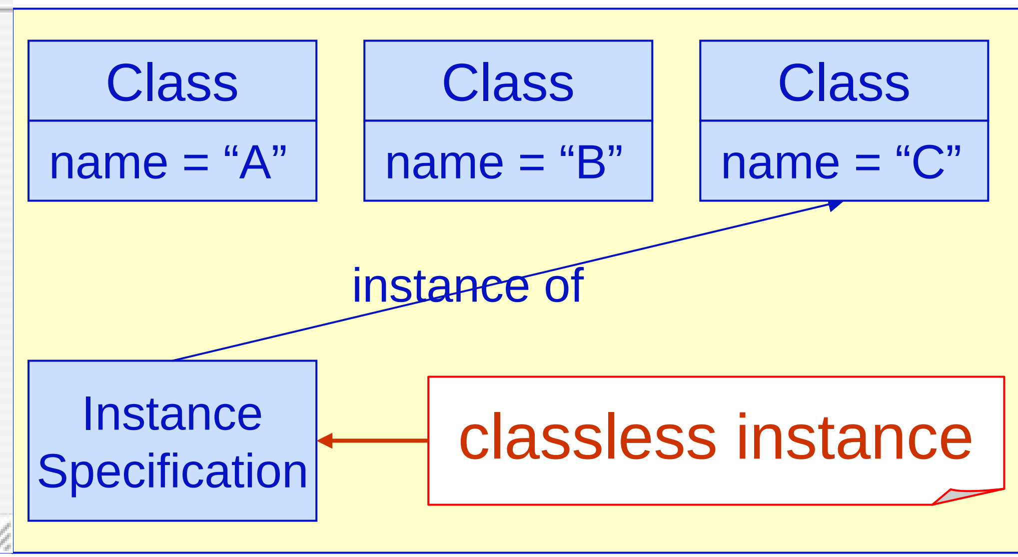


# Step 8: Tool support revisited

- A rule may have several possible matches



# Step 8: Tool support revisited

- Future work
  - Direct integration of interactive support in a UML modeling environment
  - Direct generation of type graph from the UML metamodel
  - Direct generation of inconsistency detection rules from given metamodel
  - User-friendly specification of resolution rules using some UML representation (Fujaba-like?)
  - Support for more complex (composite) resolution rules

# Discussion topics

- Optimality
- Expressiveness
- Completeness
- Compositionality
- Termination

# Discussion topics

- Optimality
  - Find an “optimal way” to resolve model inconsistencies
  - How can we define “optimality”?
  - Use heuristics and resolution strategies, locally as well as globally
    - Avoid resolution rules that introduce too many new inconsistencies
    - Prefer resolution rules that add new model elements over rules that remove model elements
    - Take into account, and “learn”, resolution rules preferred by the user
    - other strategies?

# Discussion topics

Segravis
Syntactic and Semantic Integration of Visual Modelling Techniques

- Expressiveness
  - Can all model inconsistencies be expressed?
    - Which types of model inconsistency can be expressed?
    - Graphs are well-suited for detecting structural problems
    - Behavioural problems are more difficult to express
    - Possible solution: use other formalisms (e.g. based on description logics) to detect behavioural problems
      - Integrate both formalisms in a common tool for model inconsistency management
  - Can all resolution rules be expressed?

# Discussion topics

- Completeness
  - Are all possible model inconsistencies expressed?
    - Can they be generated automatically?
  - Are all possible ways to resolve a particular inconsistency covered by the resolution rules?
    - Can they be generated automatically?

# Discussion topics

Syntactic and Semantic Integration of Visual Modelling Techniques

- Compositionality
  - How to define resolution strategies as a composition of primitive resolution rules?
    - Using sequencing, branching, looping constructs
  - How does this affect the conflict and dependency analysis?

# Discussion topics

- Termination
  - Given a set of resolution rules, can we prove that it will resolve all detected inconsistencies and terminate in a finite amount of time?
    - Rely on termination results of graph transformation

# Conclusion

- Graph transformation seems to be a viable option to support certain activities in model-driven software engineering

But it is no “silver bullet”

Alternative mechanisms and formalisms are also needed

# Conclusion