OCL Contracts for the Verification of Model Transformations

**Eric Cariou, Nicolas Belloir,
Franck Barbier and Nidal Djemam**

LIUPPA Laboratory – Self-* team
Université de Pau et des Pays de l'Adour – France

OCL Workshop, MoDELS – 5 October 09 – Denver

Context

- MDE based software process
  - Composed of a set of model transformations
  - Process and transformations are rarely fully automatized
    - Designers can / have to intervene on models manually
- Need to verify
  - That transformations have been carried out correctly
  - An even more important issue when designers intervene manually on models

Goal

- Being able to verify that a couple of models is the valid result of a transformation
- No assumptions on the way models are obtained
  - Outputs of any tool
  - Can be created or modified by hand
- Solution
  - Model transformation contracts written in full standard OCL
  - Applied on endogenous transformations
    - Source and target models are conformed to the same meta-model

Model transformation contract

- Design by contract approach
  - Specification of invariants on elements
  - Specification of operations of these elements
    - Pre and post-conditions
- Application to model transformation
  - Specification of the model transformation operation
  - Transformation operation
    - Take a single source model as input and generates a single target model as output
    - (Our approach is generalizable to several single or several target models)

Model transformation contract

- Definition of a model transformation contract
  - Constraints to be respected by the source model
    - For being able to be transformed
  - Constraints to be respected by the target model
    - For being considered as a valid result of the transformation
    - Decomposed into two sets of constraints
      - General constraints on the target model, independently of the model contents
      - Constraints on relationships between source and target elements
- Transformation contracts = 3 sets of constraints

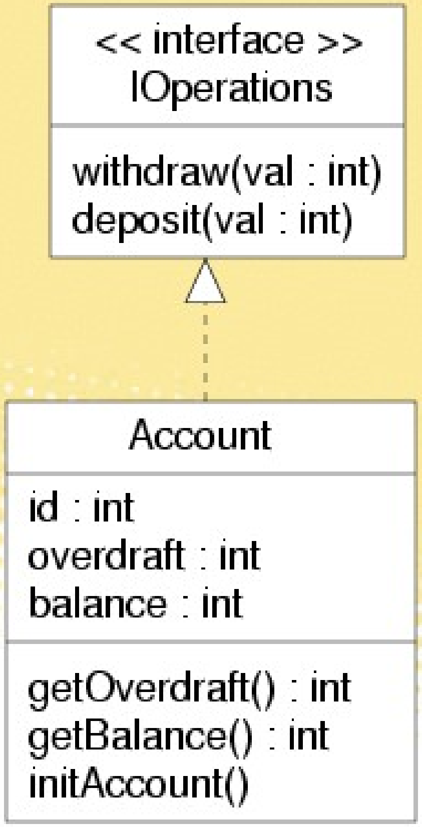
OCL for expressing contracts

- Why choosing OCL ?
  - By nature dedicated to express constraints and then contracts
  - Open standard
    - Available for several technological spaces (UML, MOF, Ecore ...)
  - Relatively well known language
    - Integrated in tools, used or extended in model transformations languages (QVT, ATL ...)
  - Formal but relatively easy to use
    - Accessible to the "lambda" designer

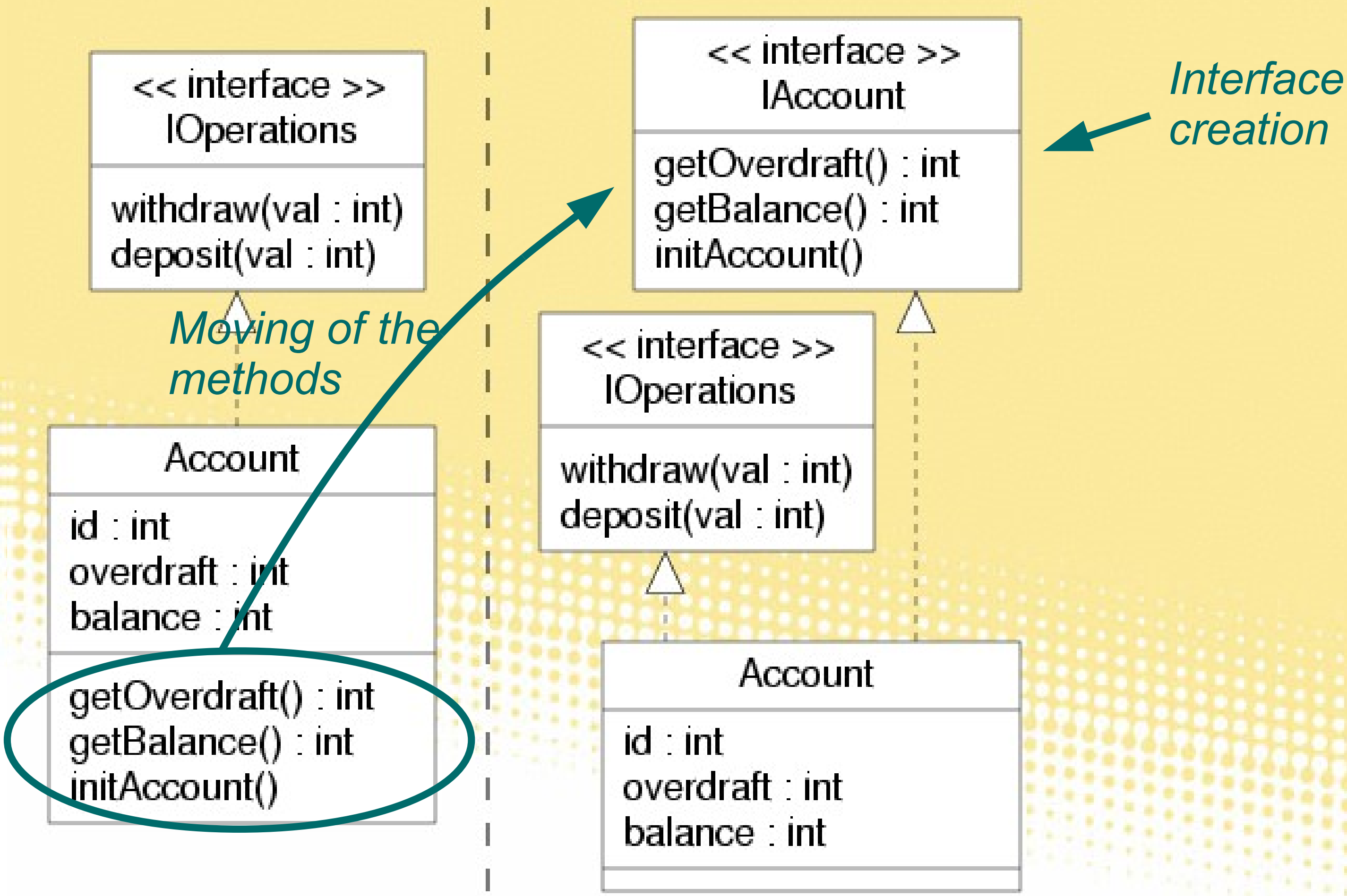
Transformation example

- On an "UML kind" class diagram
  - Refinement: addition of interfaces on classes
- Realization of the transformation
  - First step: automatic generation of an interface
    - Creation of a default interface for each class
    - Moving of all class methods to this interface
  - Second step: designer can modify the transformed diagram
    - Modification of the localization of methods
    - Modification of interfaces (renaming, addition, removing ...)

Transformation example



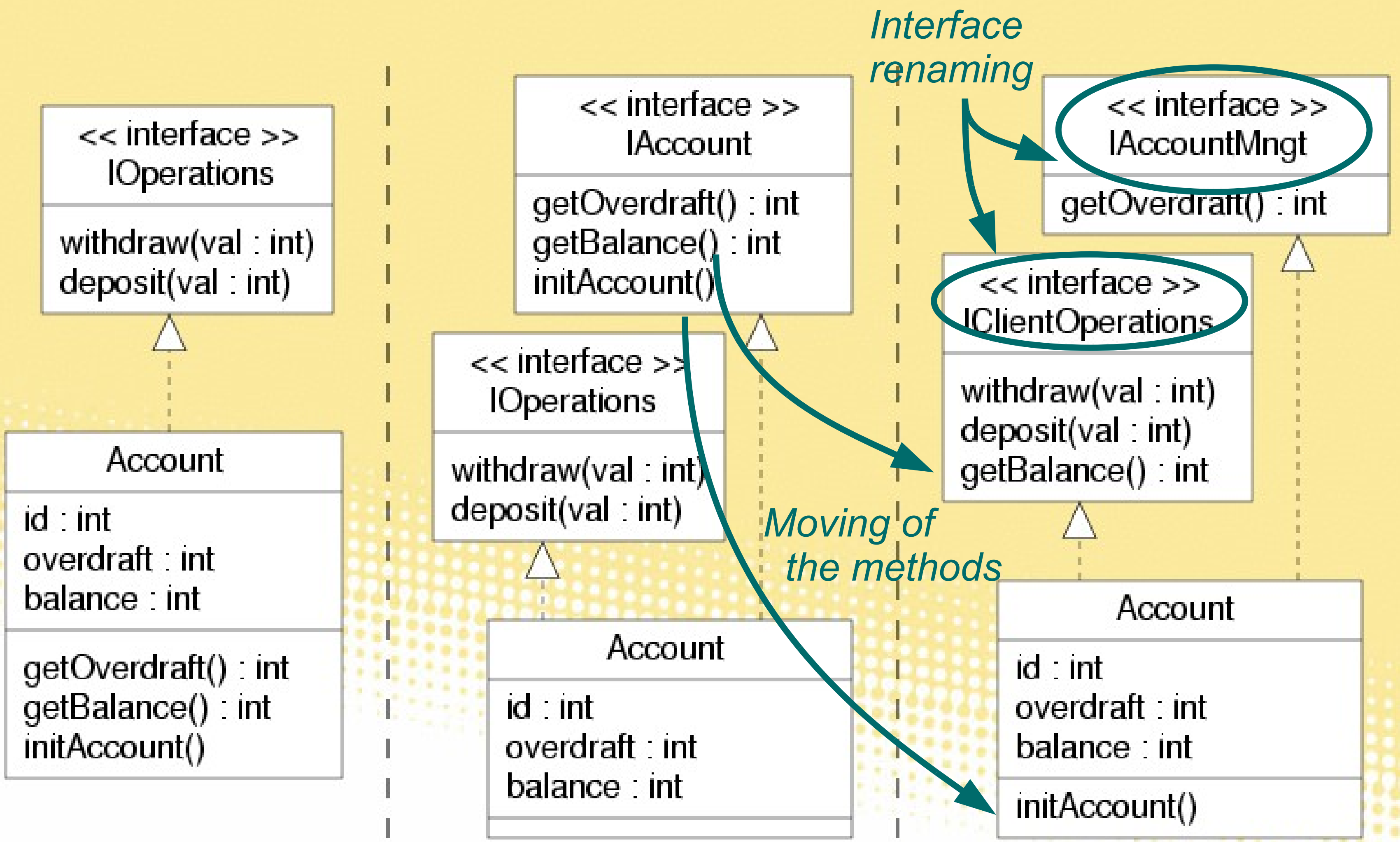
Transformation example



Initial model (source)

Intermediate model

Transformation example

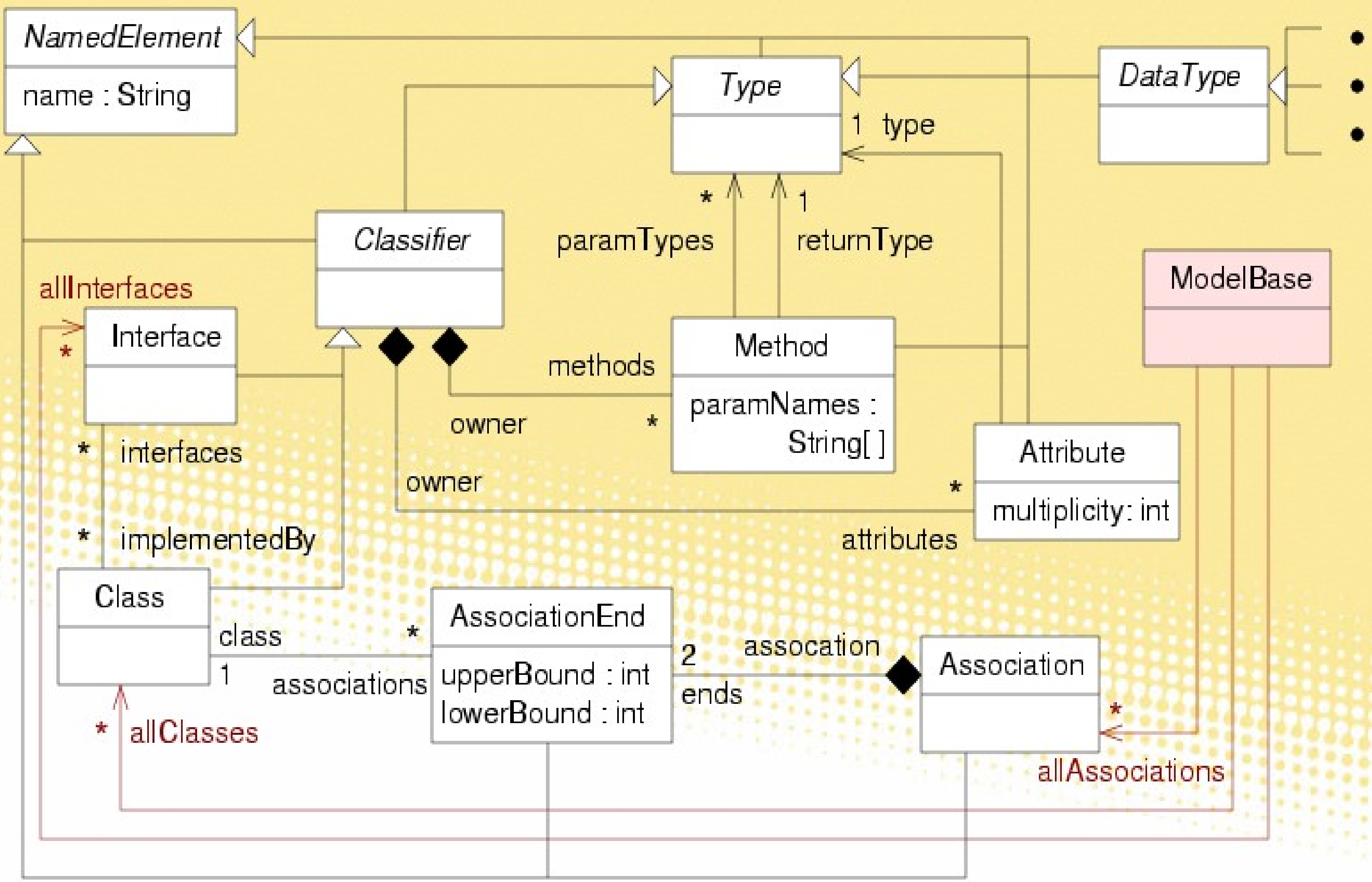


Initial model (source) | Intermediate model | Final model (target)

Transformation example

- Contract associated with this transformation
  - Constraints on source model
    - None, any class diagram can be transformed
  - General constraints on target model
    - Each class implements at least one interface
  - Evolution constraints between source and target model elements
    - After the transformation, each class still implements the same method set
      - Directly or via its interfaces
    - Can also express that some elements are not modified: associations, attributes of classes ...

Simplified meta-model of class diagram



LIUPPA Laboratory – Self-* Team

Transformation operation specification

- Definition of the contract
  - Specifying through pre and post-condition the transformation operation
    - **context** ModelBase::addInterfaces()
      pre: - - *constraints on the source model: none*
      post:
      - - *constraints on the target model*
      allClasses -> forAll (c | c.interfaces -> notEmpty()) **and**
      - - *evolution constraints between source and target*
      allClasses -> size() = allClasses@*pre* -> size() **and** ...
  - Pre-condition: reference the source model
  - Post-condition: reference the target model
    - @pre OCL construction: allows the handling of source model elements in post-condition

Transformation operation specification

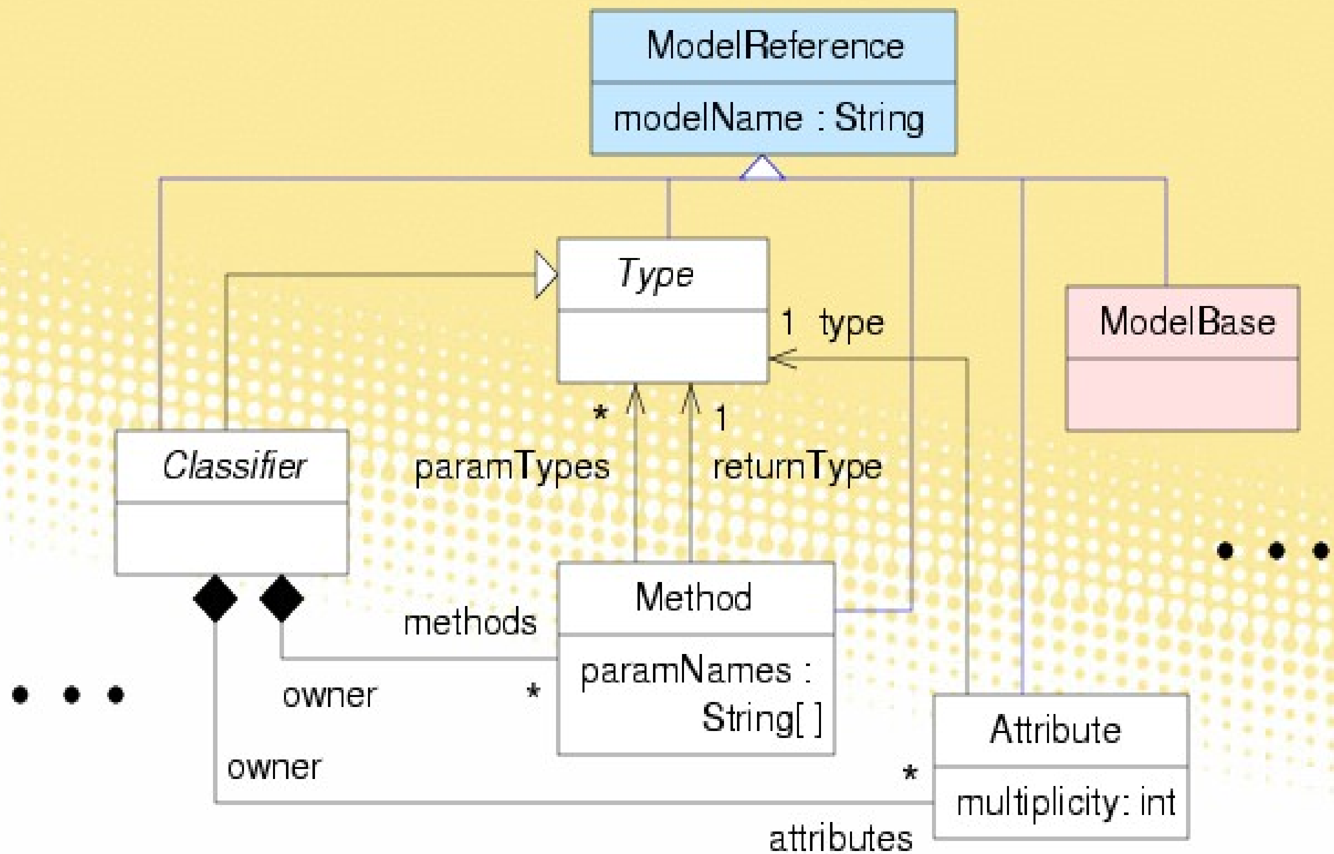
- Limitation of the pre/post specification
  - Verify that the *execution* of the transformation is correct
  - Need a tool that can realized both execution and verification
  - Strong restriction on the way to obtain the models
    - Notably no possibility to modify manually the models
- Other solution
  - Defining an OCL invariant for each of the 3 sets
  - Problem
    - Need for the evolution constraint set to define invariants applying both on source and target models
    - Not possible because of the OCL single expression context

Concatenation of models

- To overpass the single OCL context limitation
  - Concatenation of both source and target models into a third global one
    - All 3 models conform to the same meta-model
  - Express invariants and constraints on this global model
  - Need to know if an element of the global model comes from the source or the target model
- Technical solution
  - On meta-model: add a ModelReference super-class
  - Allow each element to be tagged: "target" or "source"
  - A tool automatically modifies the meta-model, concatenates the models and tags their elements

Concatenation of models

- Example: our class diagram meta-model
  - Addition of the ModelReference super-class



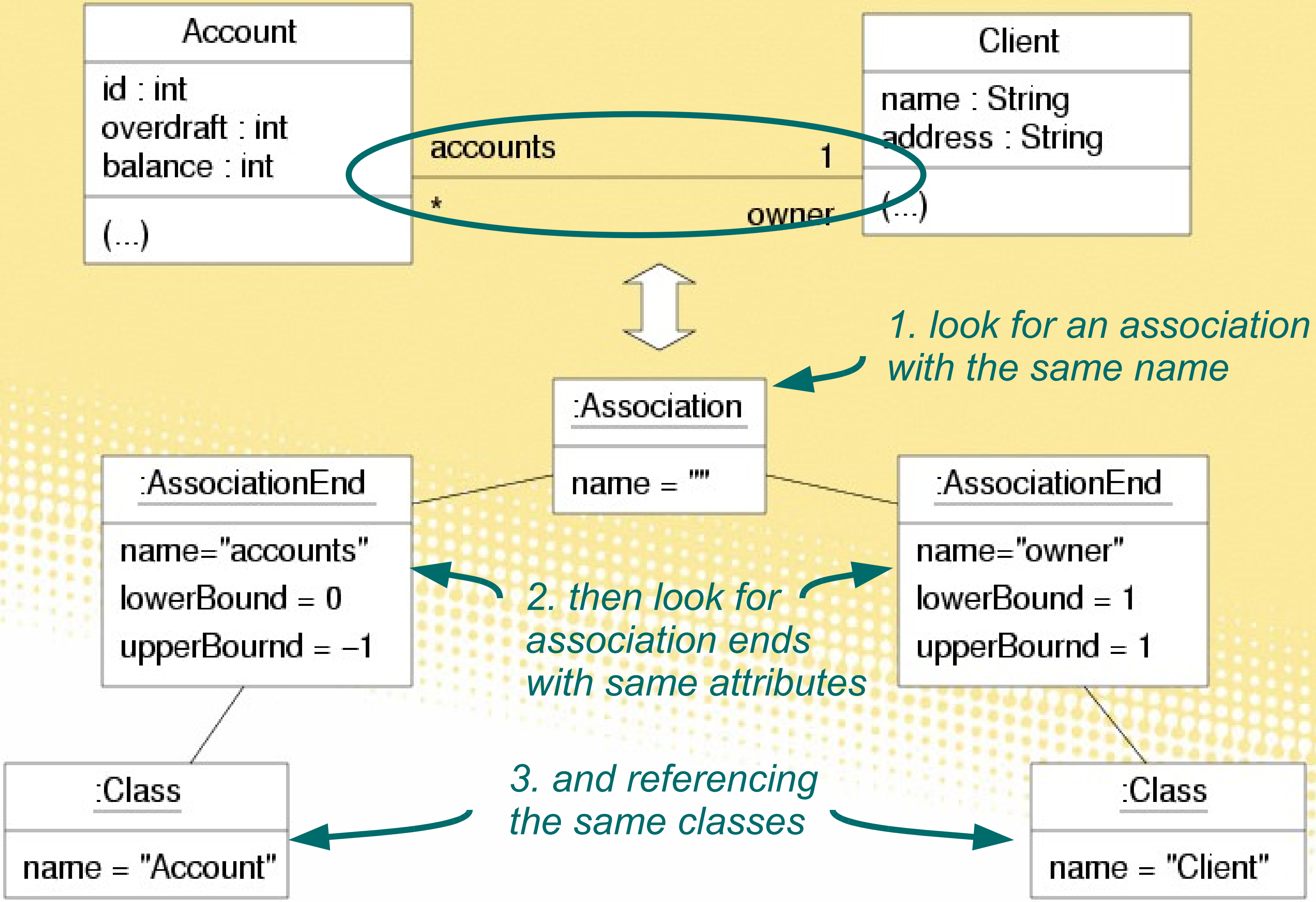
Mapping between elements

- Definition of evolution constraints
  - Constraints between the contents of some target elements and the contents of some source elements
  - Often based on the necessity to get on the source side the *mapped* element of a target element
    - Have the same type
    - Have common values or characteristics
  - Example with classes for our contract
    - "Account" class in the target model must have the same methods of the "Account" class in the source
    - Need to get the mapped class of "Account" on source
      - Simply look for a class with the same name
      - It is enough because of the unicity of type names

Mapping between elements

- Other example
  - Unmodification of associations
    - Simply need to verify that an association has a mapping on the other side with same values
  - More complex to express than for classes
    - Not unicity of association names
    - Need also to check the mappings of their association ends
      - Name, bounds and associated classes
      - Need then to check the mapping of classes
  - Transitive mapping checks on associated elements

Example: mapping for associations

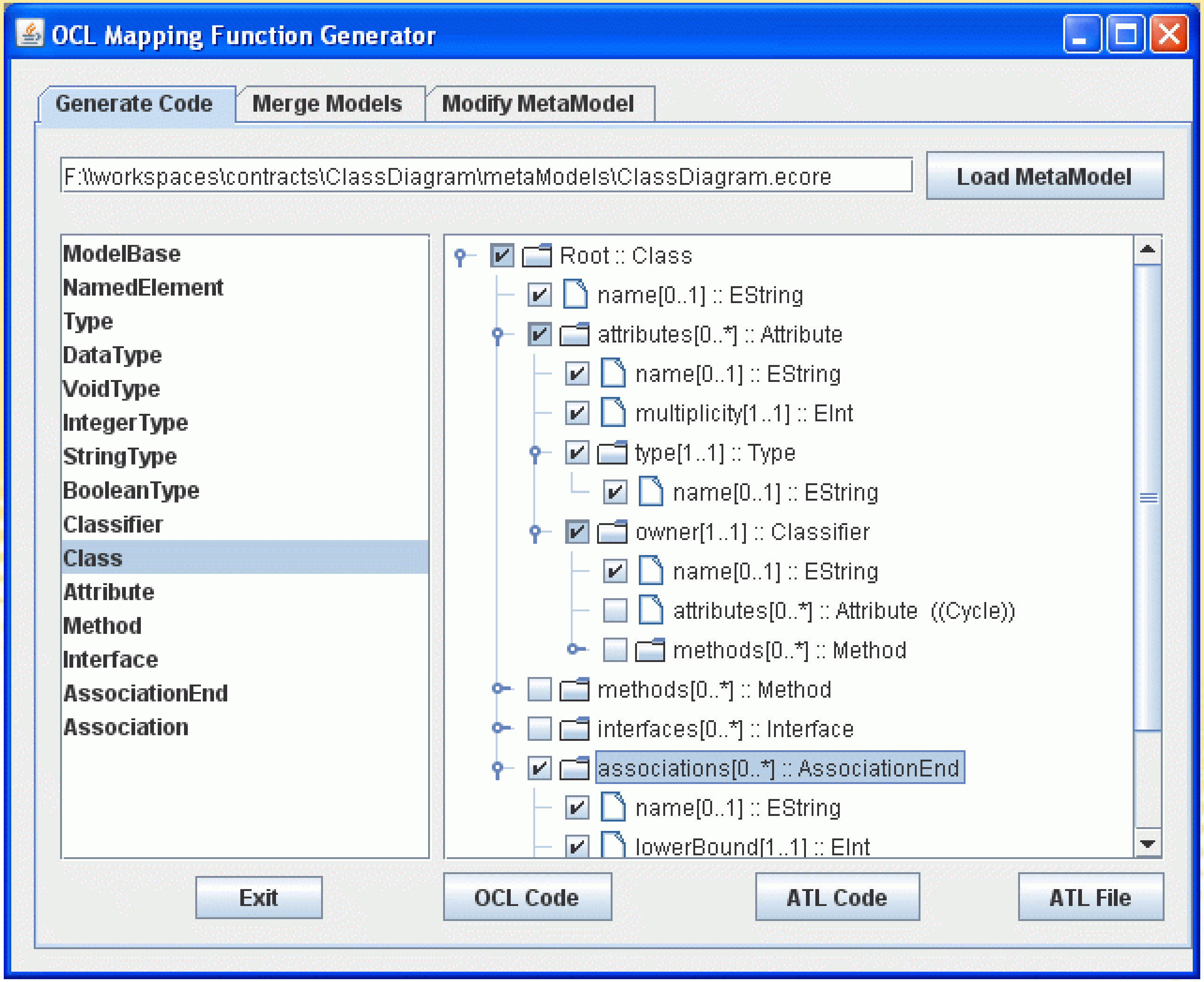


LIUPPA Laboratory – Self-* Team

Mapping functions

- Mapping functions
  - Defined through a set of OCL helpers (def:)
- Example: mapping functions for the Class element
  - **context** Class **def**: classMapping(cl : Class) : Boolean = self.name = cl.name **and** self.sameAttributes(cl)
  - **context** Class **def**: hasMappingClass(mb:ModelBase) : Boolean = mb.allClasses -> exists(cl | self.classMapping(cl))
  - **context** Class **def**: getMappedClass(mb:ModelBase) : Class = mb.allClasses -> any (cl | self.classMapping(cl))
- A tool allows the automatic generation of all required mapping functions in a contract

Tool: Mapping Function Generator



Contract example: evolution constraints

- Verification of unmodification of method sets
  - For each class on target side, check that it gets a mapped class on the source side
    - If not, the contract is not respected
  - For each of the target class and its mapped source class
    - Get its full set of methods: directly implemented or through its interfaces
  - Compare the contents of these sets
    - If not the same, the contract is not respected
- Based on mapping functions applied on
  - Classes, attributes, methods, set of attributes, set of methods ....

Contract example: evolution constraints

- Contract invariant for evolution constraints
  - All constraints expressed on the global model
  - First, get the model base element for each model
    - **context** ModelBase **def**: sourceModel : ModelBase = ModelBase.allInstances() -> any (modelName = ’source’)
    - **context** ModelBase **def**: targetModel : ModelBase = ModelBase.allInstances() -> any (modelName = ’target’)
  - Then, apply an invariant on these models
    - **context** ModelBase **inv** checkInterfaceContract: targetModel.sameClasses(sourceModel)

Contract example: evolution constraints

- **context** ModelBase **def**: sameClasses(mb : ModelBase) : Boolean =

```
self.allClasses -> size() = mb.allClasses -> size() and
self.allClasses -> forAll( c |
    if c.hasMappingClass(mb)
    then
        let myMethods : Set(Method) = c.interfaces -> collect(i |
                                  i.methods) -> union(c.methods) -> flatten() in
        let eqClass : Class = c.getMappedClass(mb) in
        let eqClassMethods : Set(Method) = eqClass.interfaces ->
            collect(i | i.methods) -> union(eqClass.methods) -> flatten() in
        c.sameMethodSet(myMethods, eqClassMethods)
    else
        false
    endif)
```

Conclusion

- Definition of model transformation contracts
  - Using only full standard OCL
  - Show that the intuitive pre/post specification is too restrictive
- Contracts = 3 sets of OCL invariants
  - Constraints on source model
  - Constraints on target model
  - Constraints on element evolution between source and target
    - Require a model concatenation "trick" to overpass the OCL single expression context
    - Strong need of a multi-context feature in OCL

Conclusion

- Mapping functions
  - One-to-one mapping between elements of the same type
    - Automatically generated thanks to our tool
  - "Simple" mappings but two major interests
    - Help in structuring and defining the contract
      - Interface contract example: 17 lines of OCL written by hand and ~35 lines for mapping functions
    - Checking unmodification parts of a model is only composed of mapping functions
      - Unmodification contracts are fully automatically generated
      - For our example: ~50 lines of OCL

Perspectives and resources

- Currently, restriction to endogenous context
  - Extension of our approach to exogenous context
    - Different source and target meta-models
    - Definition of other mapping functions in this context
  - Problem: still the single OCL expression context
    - Solution: concatenation of meta-models as for models
- Resources
  - Prototypes of our tools and full contract examples
    - For the Eclipse/EMF platform
    - http://web.univ-pau.fr/~ecariou/contracts/