A Classification of Generalisation Operators Formalised in OCL

Theodor Foerster, Javier Morales & Jantien Stoter

GI-days 2008, Muenster, Germany

17 June 2008

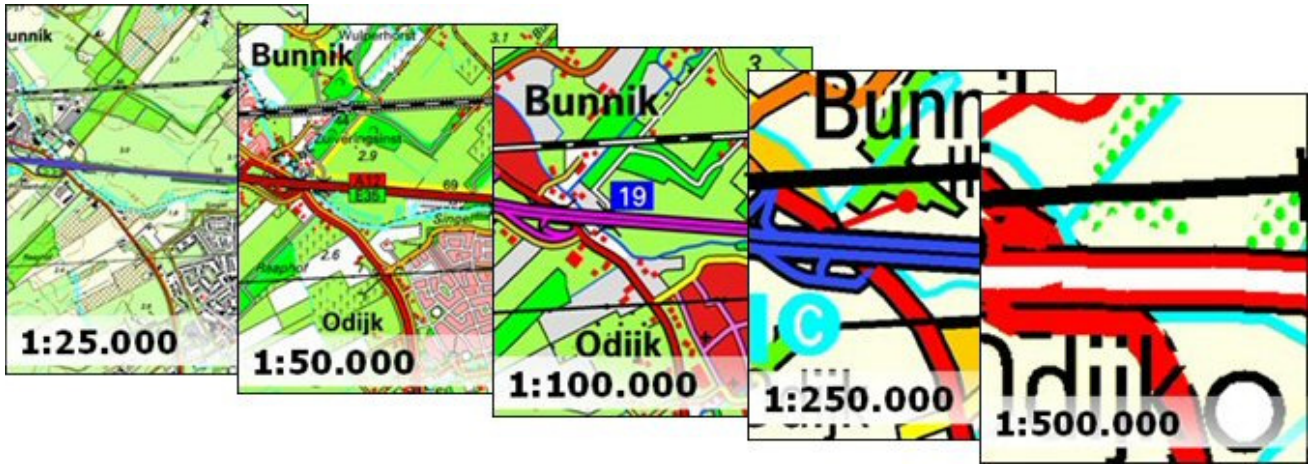
Outline

- Introduction
- The Object Constraint Language
- A Model for Generalization Operators
- Generalization Operators in OCL
- Outlook & Conclusion

Introduction

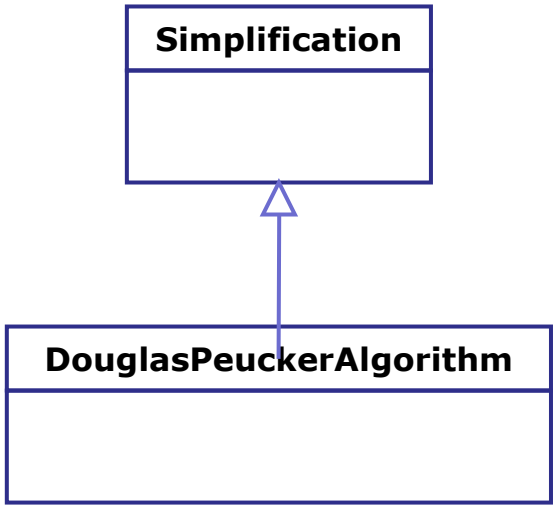
- Generalization
  - Data ➔ Information (subjective to the cartographers knowledge)
  - Dominant type of geospatial processes

- Generalization process consists of different generalization operators



Generalization operators

- Provide an abstract description of generalization functionality
  - Important to abstract human knowledge
  - Allows comparing generalization functionality
- An operator is implemented by different algorithms

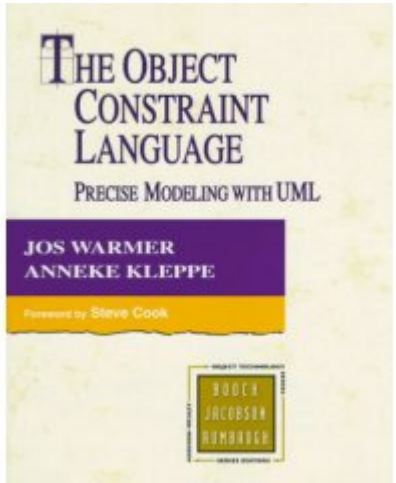


Non-formalised classifications

McMaster & Shea		Cecconi et al. (Agent)		Liu et al.
spatial transformations	Simplification	*<unspecified>*	Thematic selection	Simplification
	Amalgamation		Thematic aggregation	Merge
	Refinement	*individual objects*	Weeding	Amalgamation
	Displacement		Unrestricted simplification	Aggregation
	Smoothing		Enlargement	Classification
	Merging		Exaggeration	Selection
	Exaggeration		Fractalization	
	Aggregation		Smoothing	
	Collapse		Rectification	
	Enhancement	*individual or groups of objects*	Selection	
attribute transformations	Symbolization		Elimination	
	Classification		Displacement	
		groups of objects	Amalgamation	
			Combine	
			Typification	

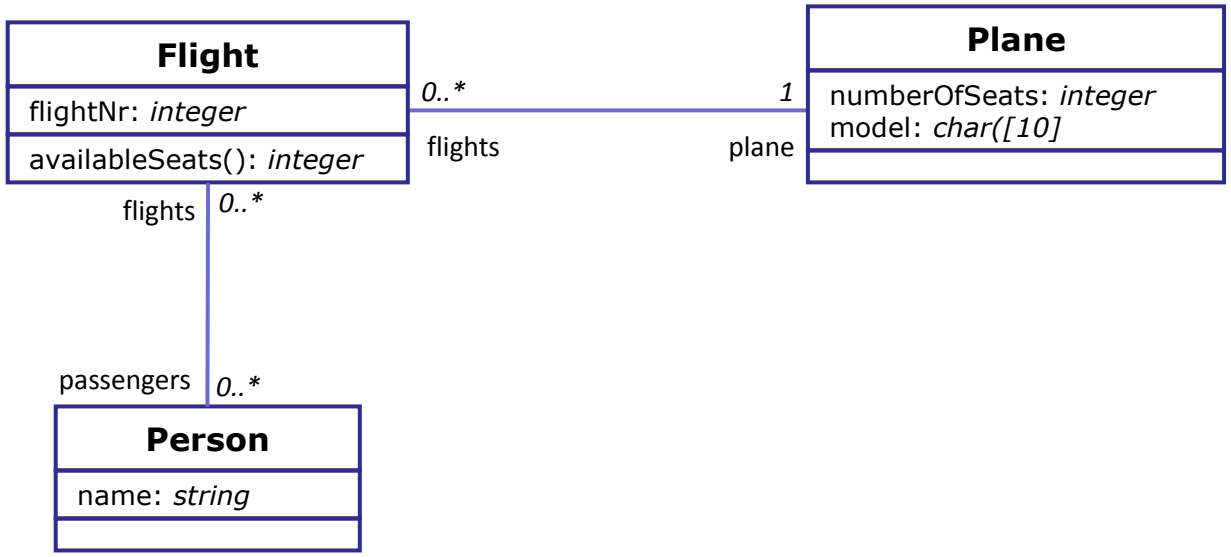
Object Constraint Language (OCL)

- Extension of UML2 for semantic modeling
  - Model Driven Architecture (MDA)
  - Maturity Levels
- Set-oriented approach to query data
  - Example: Airport->select("AMS")
- Defining consistency
  - Invariants
- Defining behaviour (methods)
  - Pre & Post conditions
  - Body expression



Warmer & Kleppe (2003): The Object Constraint Language.

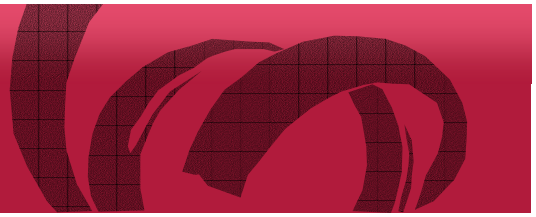
OCL an example




```
context Flight
   Inv: passengers->size() <= plane.numberOfSeats
```

```
context Flight::availableSeats(): integer
   body: plane.numberOfSeats - passengers->size
```

Motivation



- Existing classifications are not formalized
  - Formalization supports meaningful & automated integration
  - Extension of the theory of Generalization (comparable to 9-intersection model or Tomlin's map algebra)
- Application of OCL
  - Operator classification is based on models described in UML
  - Testing of formalization based on existing UML-models (topographic data)

A classification of generalization operators

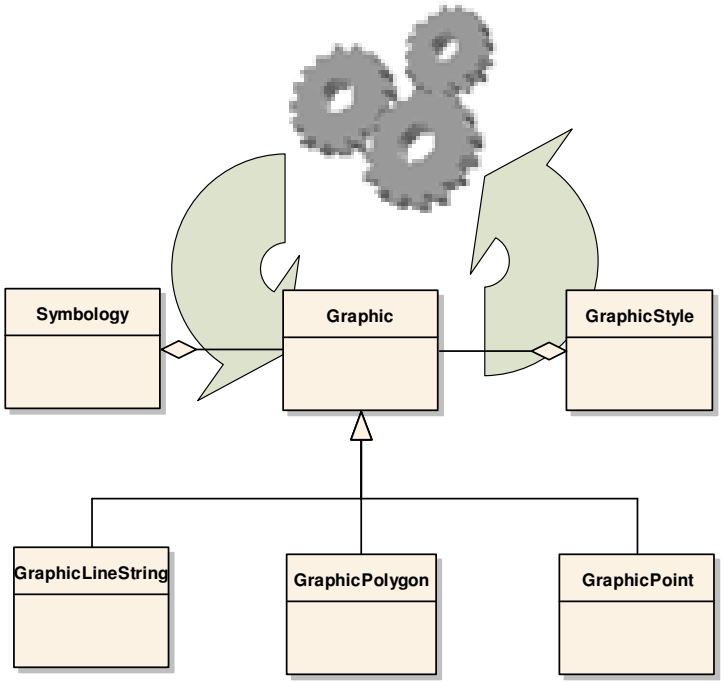
- Identified operators are based on literature

Model generalisation	Class Selection
	Reclassification
	Collapse
	Combine
	Simplification
	Amalgamation
Cartographic Generalisation	Enhancement
	Displacement
	Elimination
	Typification
	Enlargement
	Amalgamation

Foerster et al. (2007)

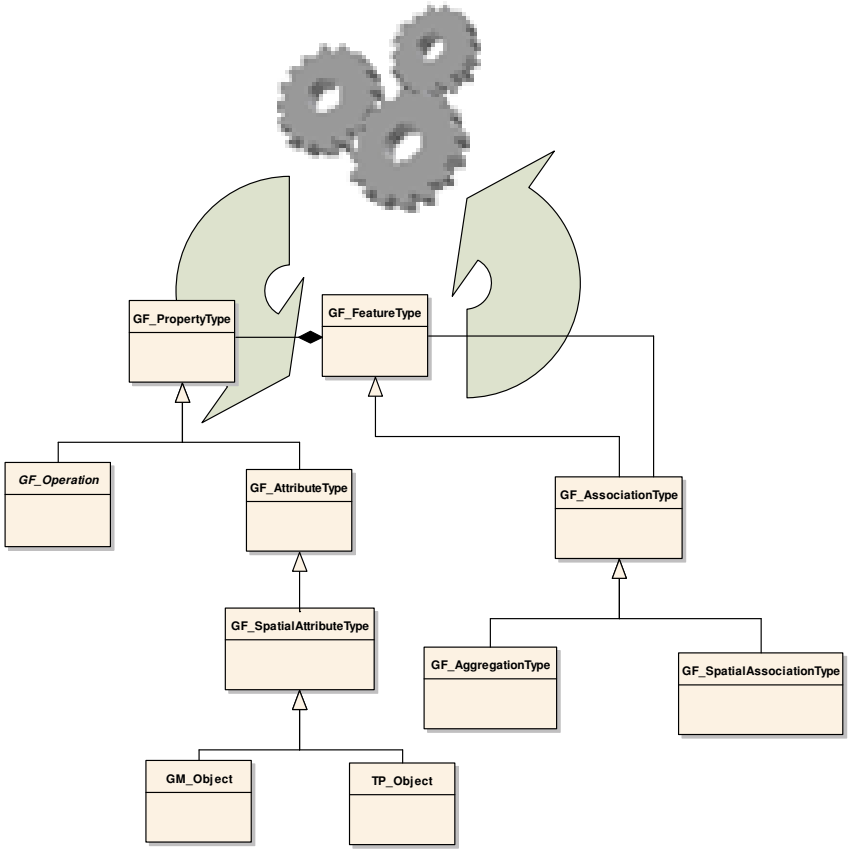
A classification of generalization operators

Cartographic Generalization operators



OGC Go-1 Application Objects

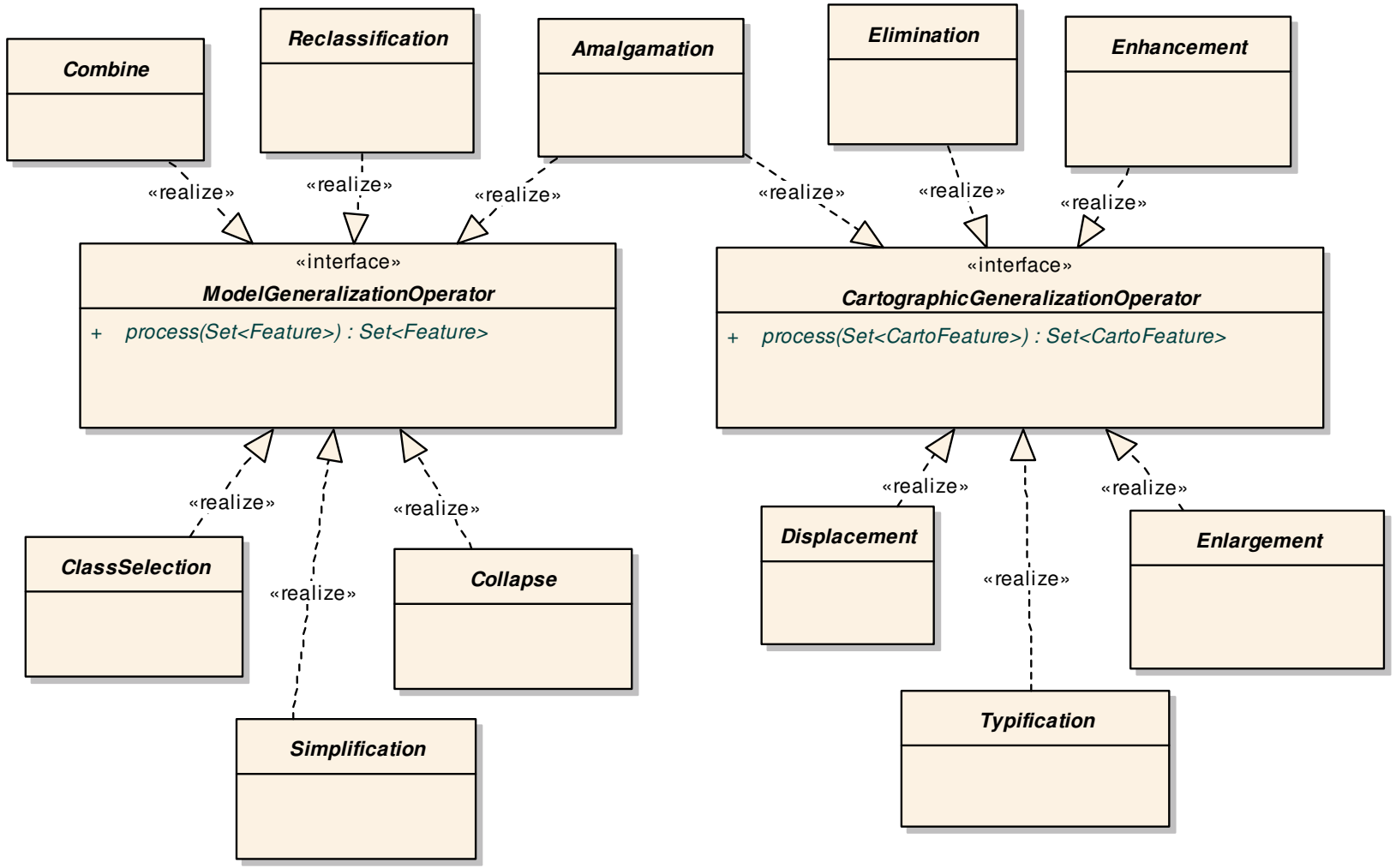
Model Generalization operators



ISO 19109

Classification is based on standardized data models

UML model for generalization operators



- Interfaces vs. abstract classes
  - Abstract classes allow to implement/specify some functionality

Formalized Operator - Simplification

- Mainly used for data reduction
- Deletes aspects of a feature

OCL snippet:

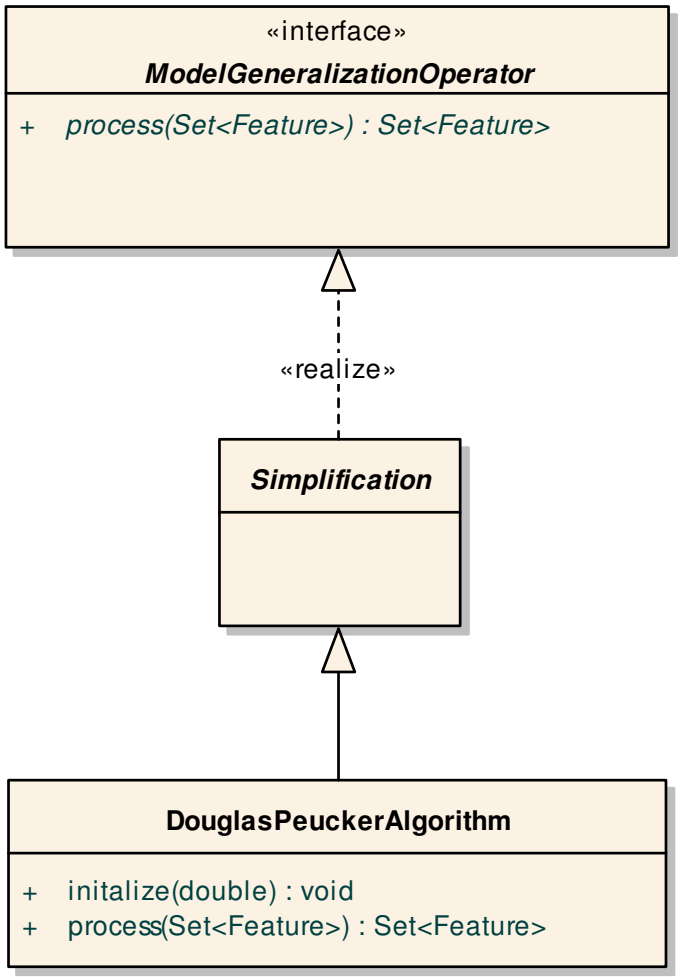
```
Context Simplification::process(fc: Set<Feature>) : Set<Feature>

   pre: fc->    forAll(Featuref|f.geometry.dimension>=1)
        fc->    forAll(Feature f|f.addAllowed = false)
        fc->    forAll(Feature f|f.geometry.->
                forAll(Coordinate c|c.changeAllowed = false))

   post: result->      forAll(Feature f|f.geometry.nGeometry() <

        fc.getByFid(f.fid).geometry.nGeometry())
```

Modeling Operators vs. Algorithms



Simplification Algorithm in OCL

- Douglas-Peucker Algorithm
- Modelling issues
  - Passing individual parameters initialize(double)
  - Functionality is defined in private simplifyGeom() to handle iterative approach of DP Algorithm

OCL snippet:

```
Context DouglasPeuckerAlgorithm
   def: simplifyGeom(geom:Geometry) Set(coords)
   body:
        if(geom.size > 2)
                let cand : Geometry = geom.coordinates-
                >sortBy(c.distance(Line(geom.first,geom.last)))->at(1)
                if(cand.distance(Line(geom.first,geom.last))>
                        dpTolerance
                        let coords: Geometry
                        add(cand)
                        coords->prepend(simplifyGeom(geom(first,cand))
                        coords->append(simplifyGeom(geom(cand,last)))
                        result = coords
                endif
        endif
```

Douglas-Peucker Algorithm

1. check that the number of points **n > 2**
2. determine a tolerance **t**
3. find the point whose perpendicular distance to the segment $\overline{\mathbf{geom_{first}}\,\mathbf{geom_{last}}}$ is the greatest
   - Call this point $\mathbf{geom_{cand}}$
   - Call this distance $\mathbf{dist_{cand}}$
4. if $\mathbf{dist_{cand}} < \mathbf{t}$, then
   discard the points $\mathbf{P_{first+1}}..\mathbf{P_{last-1}}$
   else
   - Simplify the array of points $\mathbf{geom_{first}}..\mathbf{geom_{cand}}$ using **t**
   - Simplify the array of points $\mathbf{geom_{cand}}..\mathbf{geom_{last}}$ using **t**

Outlook

- Operator description in other languages
  - Web Services: WSML, OWL
  - Knowledge modeling: Haskell
- Modeling dependencies between operators
- Linking operators with cartographic constraints
- Link the described operators to topographic data models

Conclusion

- Model of operators extensible
  - Interface & abstract classes vs. Category & operator
- OCL sufficient to model classification
- OCL is lacking of tool support
- Basis for further formalization applied in other applications
  - Web Services & data models

Thanks for your attention!

Questions?

Email: {morales, foerster}@itc.nl