Advanced ontology visualization with OWLGrEd

Jānis Bārzdiņš, *Kārlis Čerāns*,
Renārs Liepiņš, Artūrs Sproģis

Ontology visualization task

Ontologies represent knowledge

Knowledge – not only for computers, but also for people

How to create, share, learn ontologies?

Ontology visualization task: visual rendering, visual editing

Tools exist for textual (e.g. Protege) and graphical ontology rendering and editing

OWLGrEd editor: compact graphical + textual notation, based on UML class diagrams and OWL Manchester syntax

OWLGrEd: Main concepts

UML Class diagram notation:

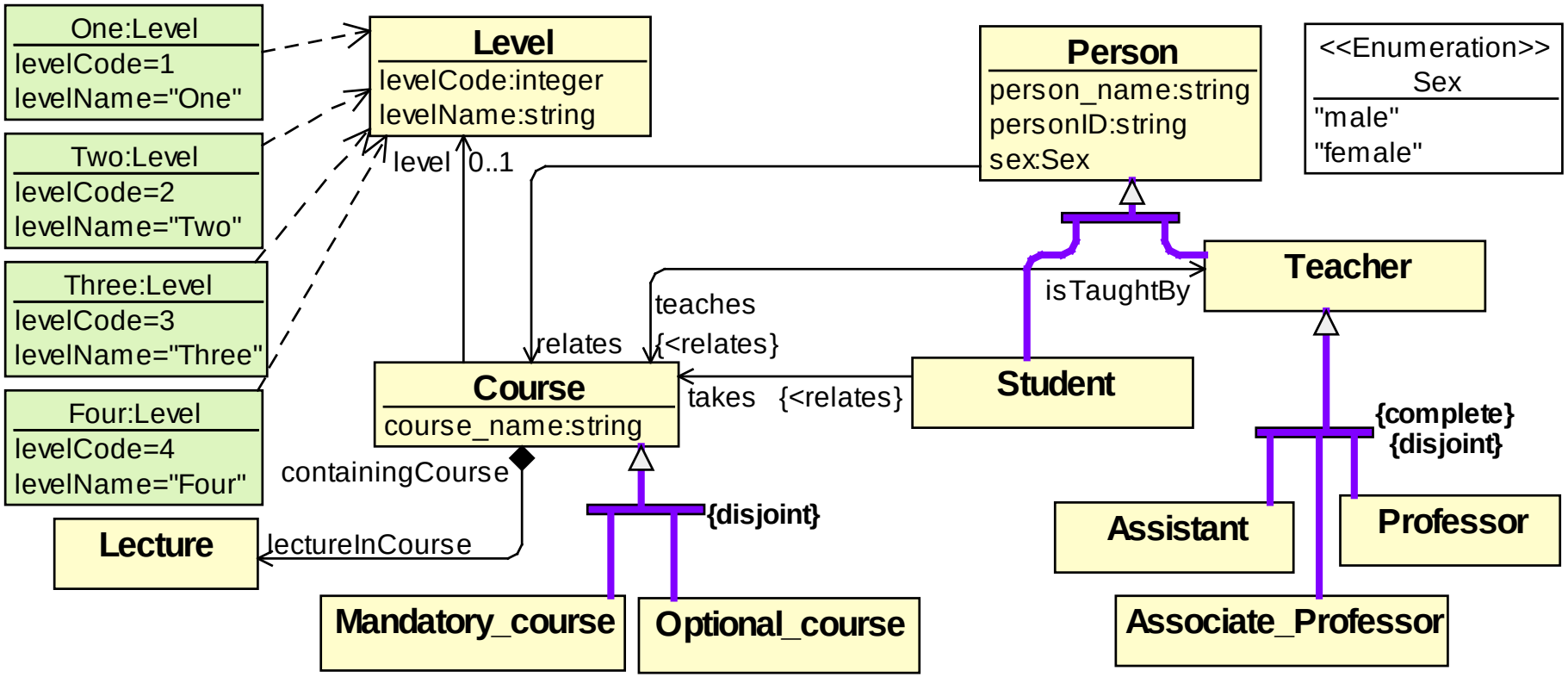
convenient for class-based modeling,
apply this to OWL 2.0 ontology modeling

- UML Package <-> OWL Ontology
- UML Class <-> OWL Class
- UML Association End <-> OWL Object Property
- UML Attribute <-> OWL Data Property
- UML generalization <-> OWL SubClassOf axiom

OWL features without direct UML Class diagram counterpart:

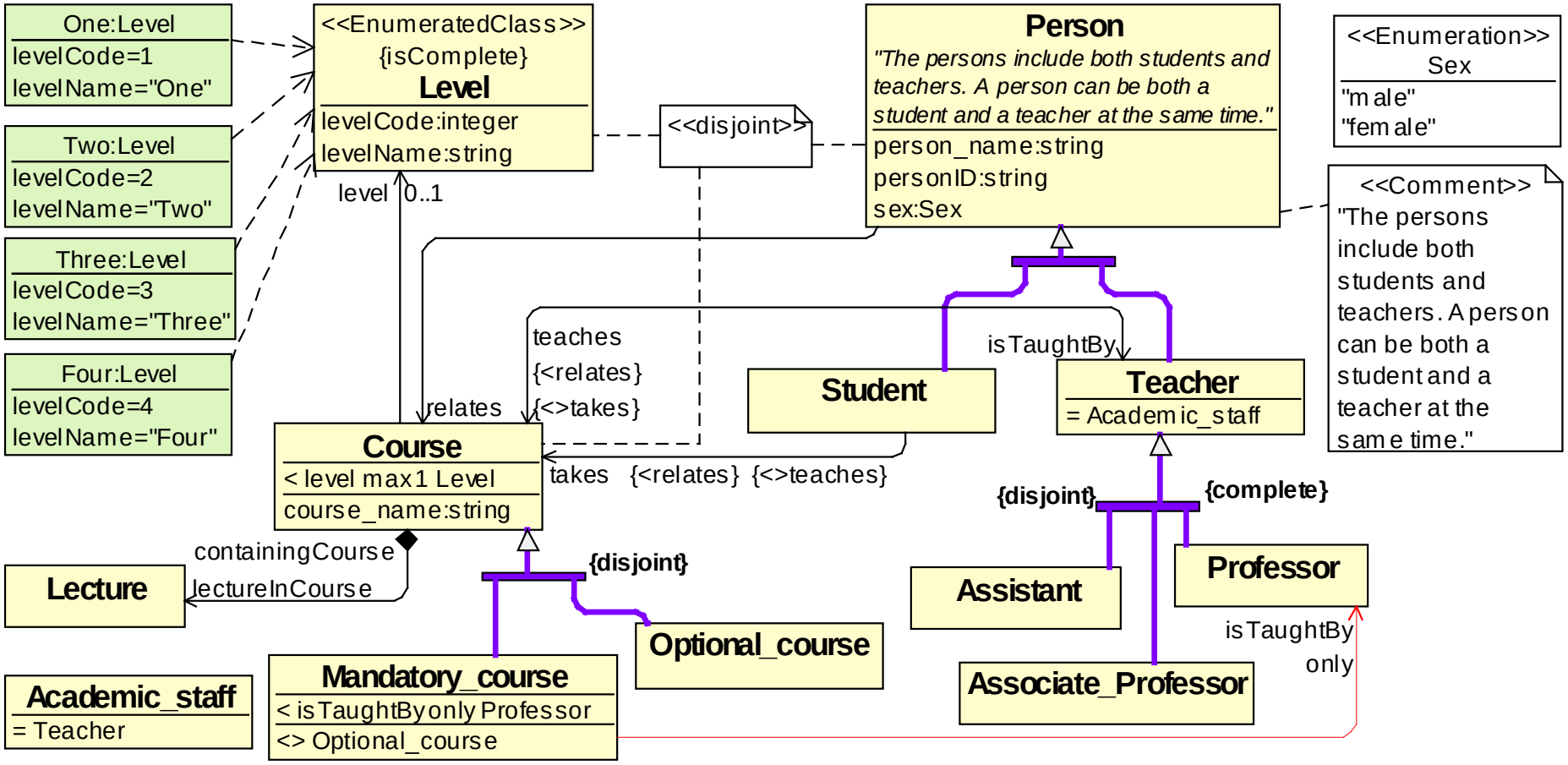
- equivalent, disjoint classes, some OWL class expressions:
  use custom / adopted graphical syntax
- OWL class expressions: use textual Manchester syntax

Mini-University ontology in OWLGrEd



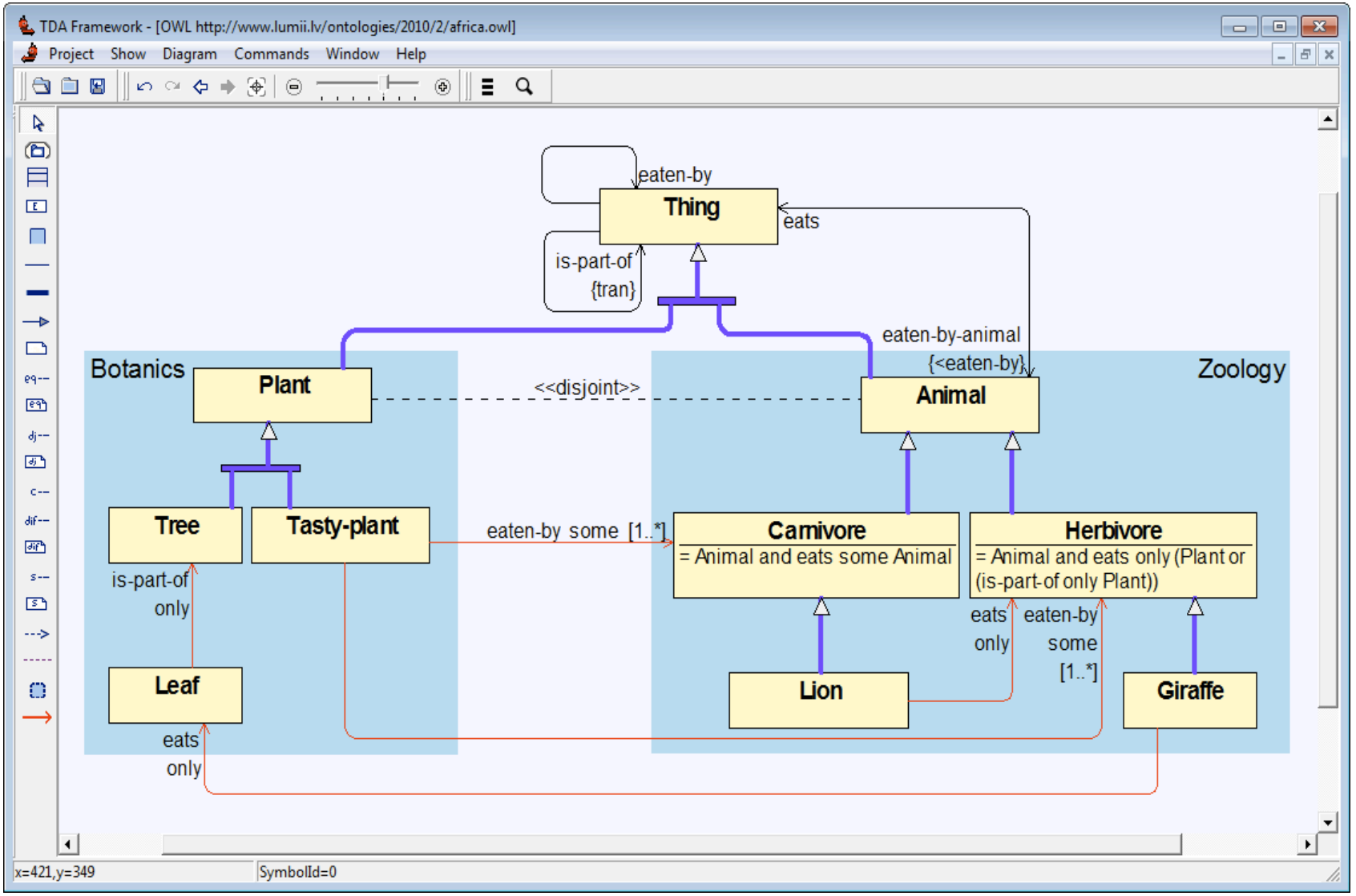
- classes, properties, individuals
- class as single domain / range for a property
- some cardinality restrictions
- generalization, generalization sets, complete, disjoint
- enumerations – simple data ranges
- composition – only visual representation

Extended UML notation



- Equivalent, disjoint classes: note with connectors, binary connectors, textual form
- Single logical meaning – different graphical presentations
- Class expressions in Manchester notation: <, = and <> compartments.
- Anonymous class depicted visually, if used as a domain/range for a property.
- Graphical restriction forms: some/only, cardinality restrictions
- Enumerated Class

African Wildlife ontology in the editor



Note: free comments (Botanics, Zoology) used for extra conceptualization

OWLGrEd: usage patterns

Ontology import/export: OWLGrEd ←→ Protege 4.1.

Uses Protege OWLGrEd plugin

Usage pattern 1: Ontology visualization and editing

- Create/import OWL 2.0 ontology using Protege 4.1
- Export to OWLGrEd / TDA (exporting options available)
- Customize ontology visualization (automatic re-layouting, manual layout and appearance customization)
- Ontology editing facilities available

Usage pattern 2: Visual ontology creation

- Create ontology using OWLGrEd visual ontology editing facilities
- Export the created ontology to Protege for interoperability with reasoners and/or other ontology management tools

OWLGrEd: state of the art

Stable release: http://owlgred.lumii.lv

- standalone OWLGrEd tool
- Protege plugin for interoperability

News:

Works with Protege 4.1.

Full (almost) support of OWL 2.0

Supports: data ranges, keys, property chains, ontology imports, annotations (mostly)

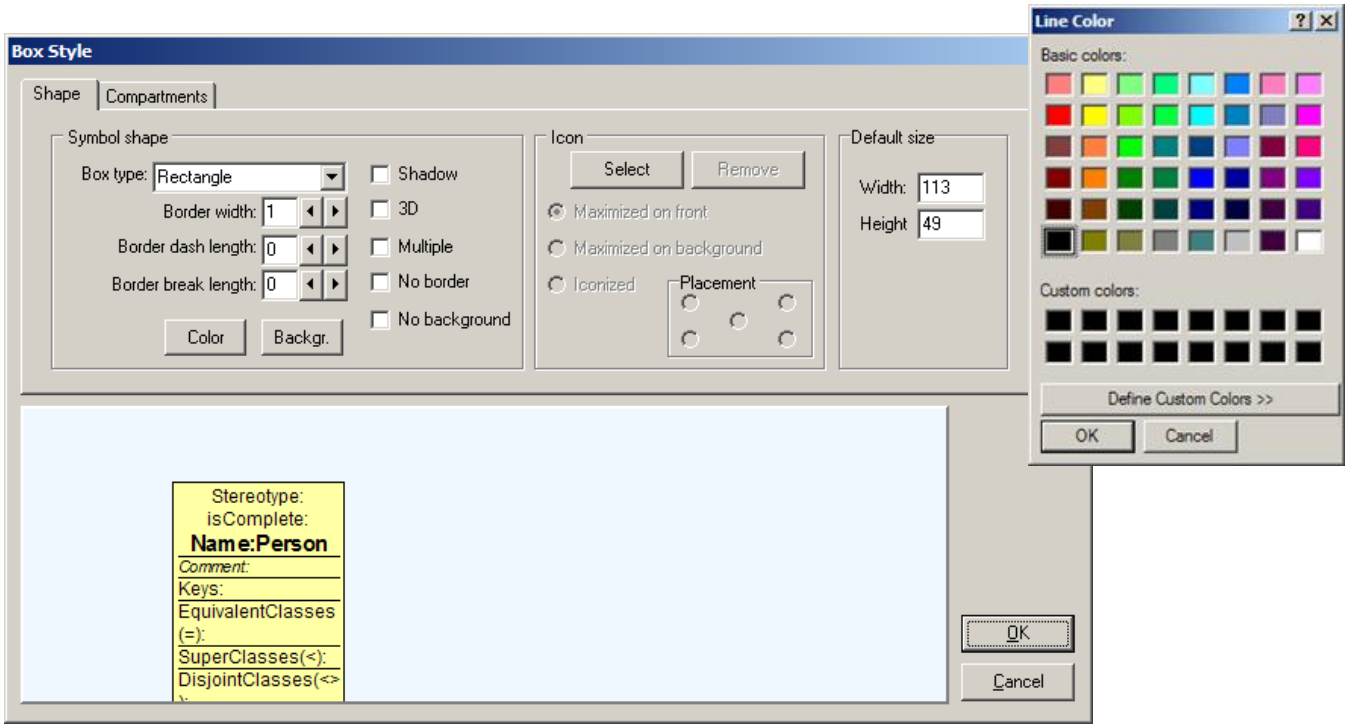
No graphical representation: some annotations, e.g. axiom annotations.

Built-in advanced modeling constructs: composition, free comment (no representation in OWL).

OWLGrEd: Basic visualization

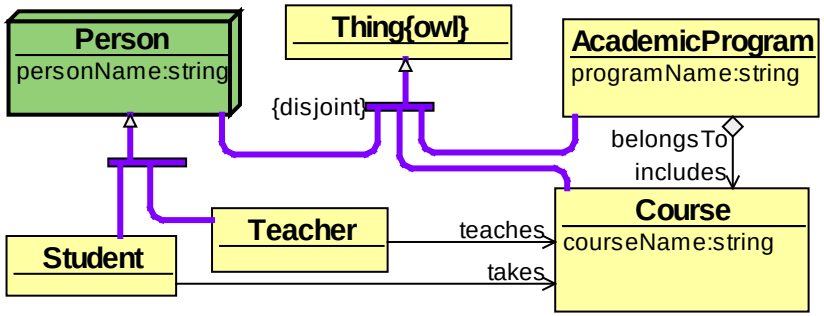
- Local visual style changes for any item (color, shape, text font, etc.)
- Globally setting custom styles for certain element types (e.g. all classes, all object properties, all notes)

No effects outside OWLGrEd

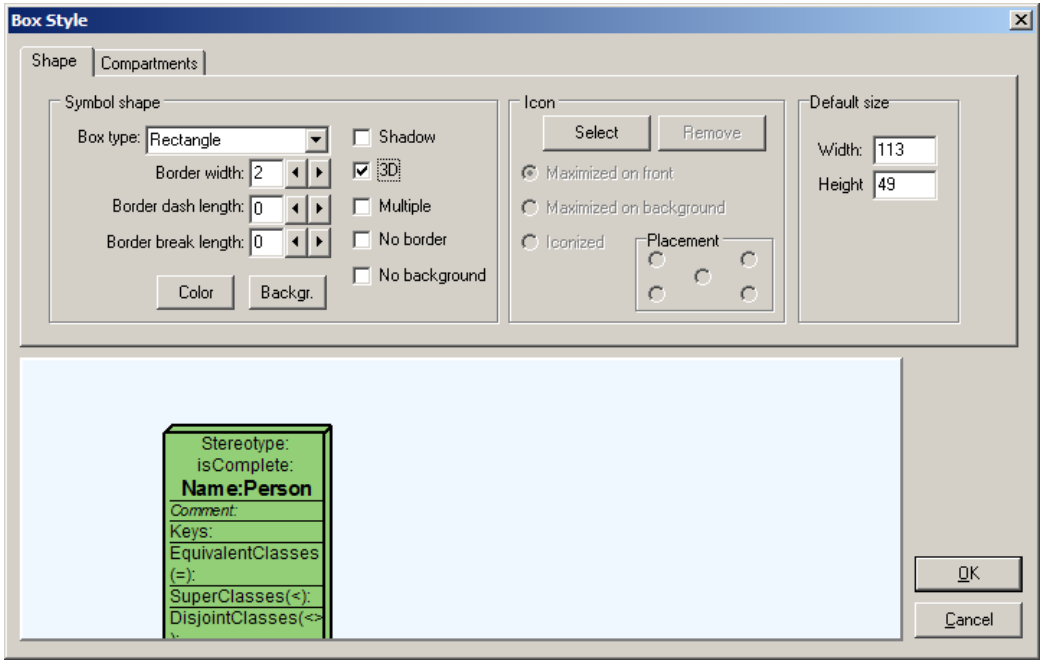


Visualization annotations: a naïve way

- Introduce annotation property for visual style, e.g. Declaration(AnnotationProperty (og:ClassDisplayStyle))
- In ontology export, annotate the user ontology items (e.g. classes, properties, individuals) that have specific style, e.g. AnnotationAssertion(og:ClassDisplayStyle :Person "bkgColor=green, borderWidth=2, 3D=true")
- In ontology import, recognize the og:ClassDisplayStyle-annotations to set the custom style of the class box.

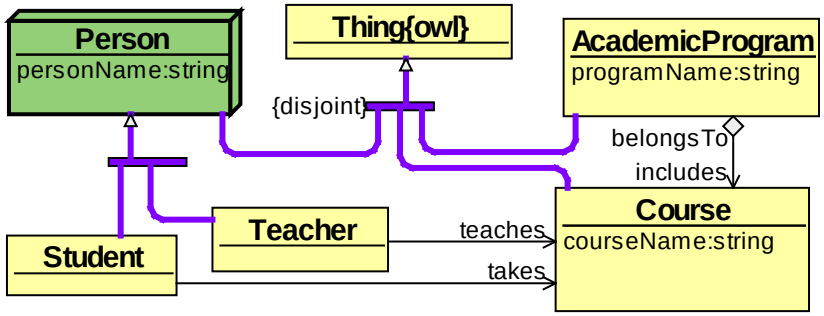


- OWLGrEd++: group visual styles and attach to user-defined annotation properties



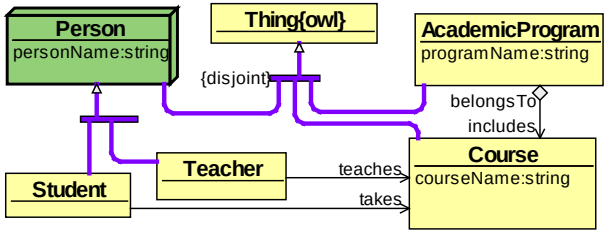
Visual annotation framework idea

- Suppose there is a built-in annotation property in OWLGrEd for visual style, e.g. Declaration(AnnotationProperty (og:ClassDisplayStyle))
- Let the user (e.g. a power user) introduce a domain-specific annotation property Declaration(AnnotationProperty (user:ImportantClass)) ..
- .. and annotate the user annotation by the visual annotation: AnnotationAssertion(og:ClassDisplayStyle user:ImportantClass "bkgColor=green, borderWidth=2, 3D=true")
  (These definitions are stored in visual profile ontology; handled by OWLGrEd in a special way)
- In ontology import, set the custom style of the class box whenever the class has been marked by the user annotation property: AnnotationAssertion(user:ImportantClass :Person "true")
- A custom visual specification language has been created!



Framework for «Annotation Visualization»

- Let the user (e.g. a power user) introduce a domain-specific annotation property Declaration(AnnotationProperty (user:ImportantClass)) ..
- Annotate the user annotation by the visual annotation: AA(og:ClassDisplayStyle user:ImportantClass "bkgColor=green, borderWidth=2, 3D=true")
- Set the custom style whenever: AA(user:ImportantClass :Person "true")



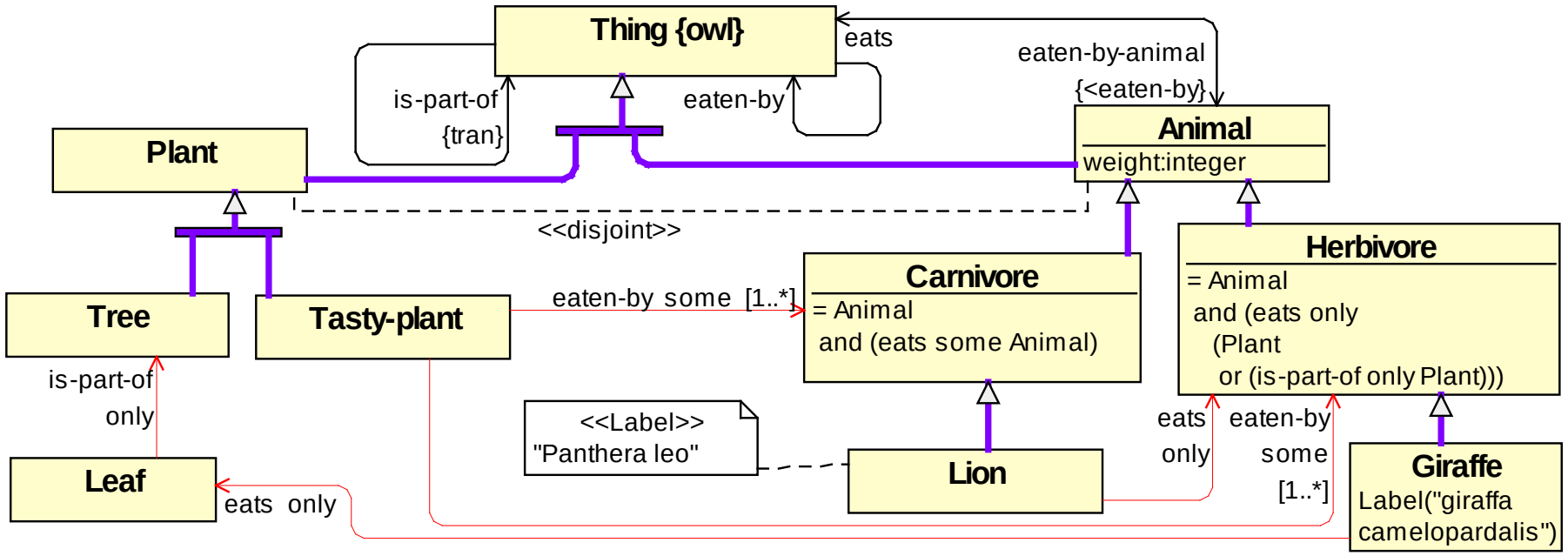
Visual annotations to user annotation properties– much more powerful concept, allows specifying visualizations for any annotation properties:

- Style annotations (the annotation properties created to determine visual style)
- Value annotations («normal» annotation properties, carrying a meaningful annotation value, possibly to be displayed)

Visual settings – where (e.g. inside the box/in outside note) and how (e.g. the field style) to display the annotation value.

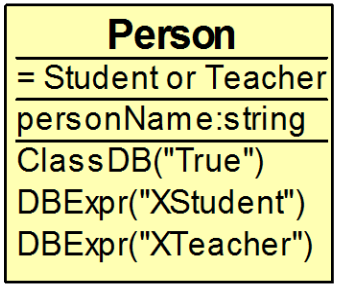
Annotation value entry settings: e.g. presence of language/datatype fields, placement of the field on property sheet, supporting event procedures.

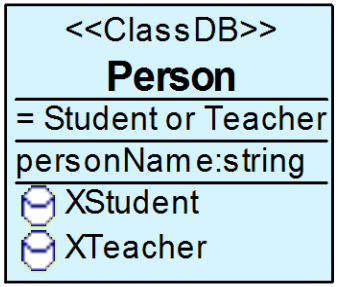
Example: annotation placement inside/outside



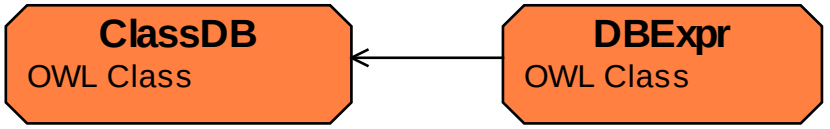
Example: Database expression specification

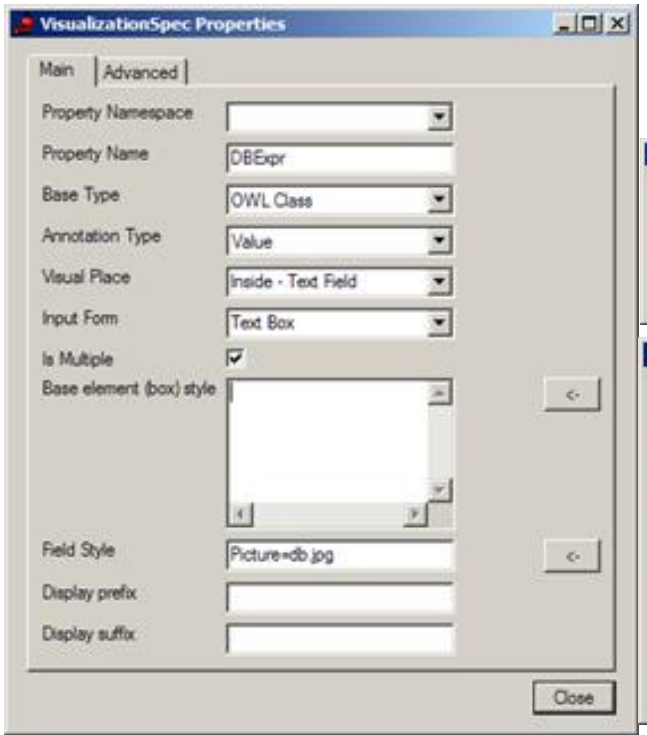
- **AA(** A(og:inputForm og:ListItem)
  A(og:displayElemStyle “bkgColor=’blue’”)
  og:aClassShowMode :ClassDB og:Style)
- **AA(** A(og:aDependency :ClassDB)
  A(og:displayFieldStyle “picture=’db.jpg’”)
  og:aClassShowMode :DBExpr og:ValueInside)

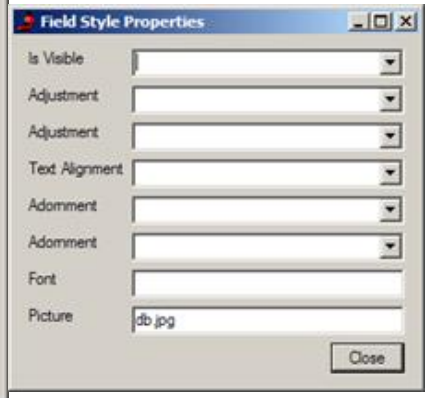




Specification in OWLGrEd: annotation profile diagram, to be used by user ontology diagram.





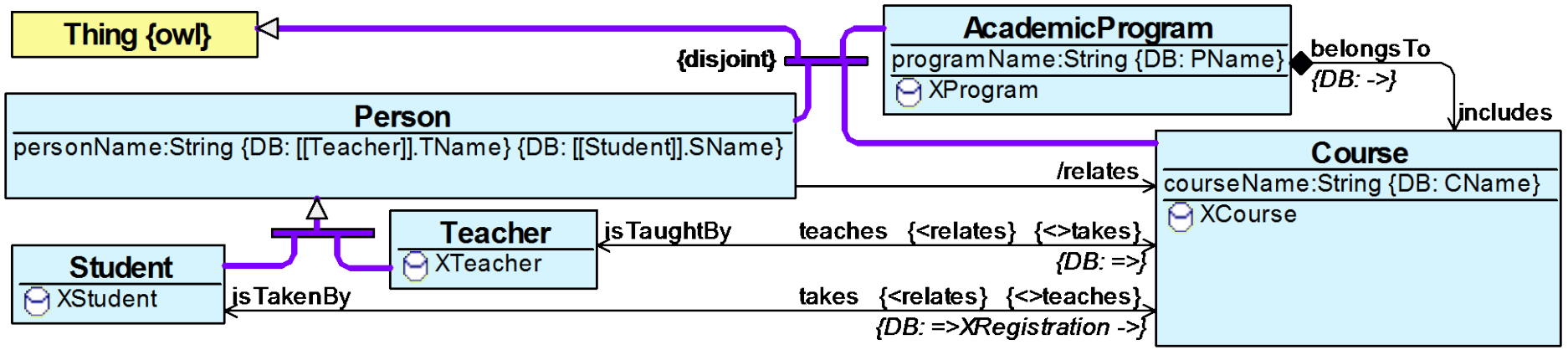


UML stereotype coverage:

- Style specification
- Dependent values («tagged values» in UML).

Only visualization aspect here (the OWL format allows attaching any annotations also without any «stereo-styles»)

Example: UML Composition and derived union



AA(A(og:inputForm og:CheckBox)
A(og:displayElemStyle “lineStart=’diamond’”)
og:aObjectPropertyShowMode user:isComposition og:Style)

AA(A(A(og:compID “name”) og:displayValuePrefix “/”)
A(og:inputForm og:CheckBox)
A(og:displayStyle “isVisible=false”)
og:aObjectPropertyShowMode user:isDerivedUnion og:Style)

AA(user:isComposition :includes “True”)

AA(user:isDerivedUnion :relates “True”)

The two user annotations allow to obtain typical UML visualizations for UML composition and derived union constructs for object properties (for data property annotation visualization similar constructions are used).

Conclusions

- Work in progress (full annotation visualization ontology, implementation in the editor)
- Annotating the annotation properties – a powerful principle for defining high-level ontology visualization constructs
- UML constructs – composition, property derived unions – special examples; UML stereotype functionality covered and extended
- Tool building platform – can discuss graphical extensions to OWLGrEd that capture «logical» meaning
- Meanwhile: http://owlgred.lumii.lv

http://owlgred.lumii.lv/

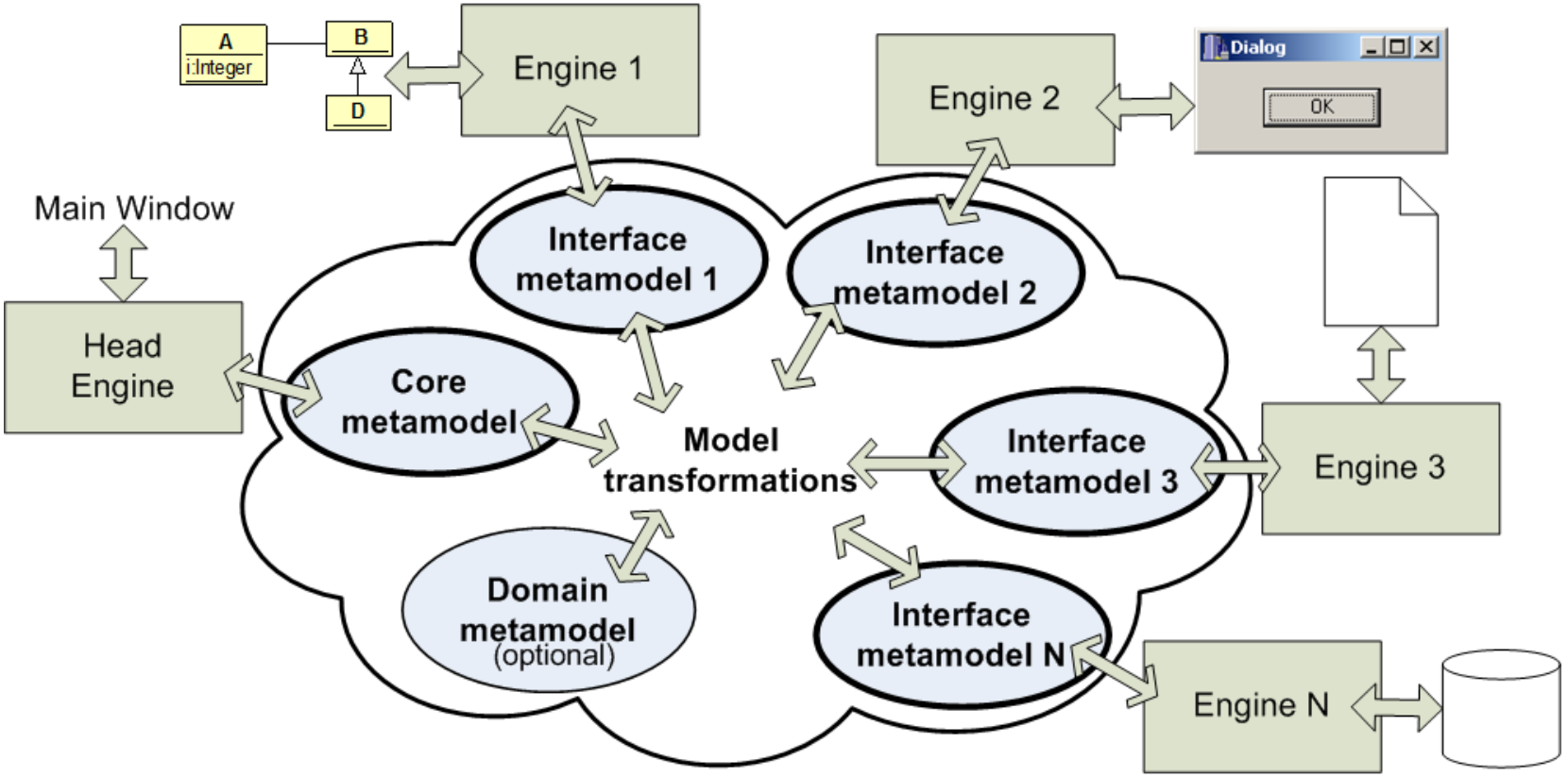
Thank you!

The OWLGrEd development team:

Jānis Bārzdiņš, Kārlis Čerāns,
Renārs Liepiņš, Artūrs Sproģis

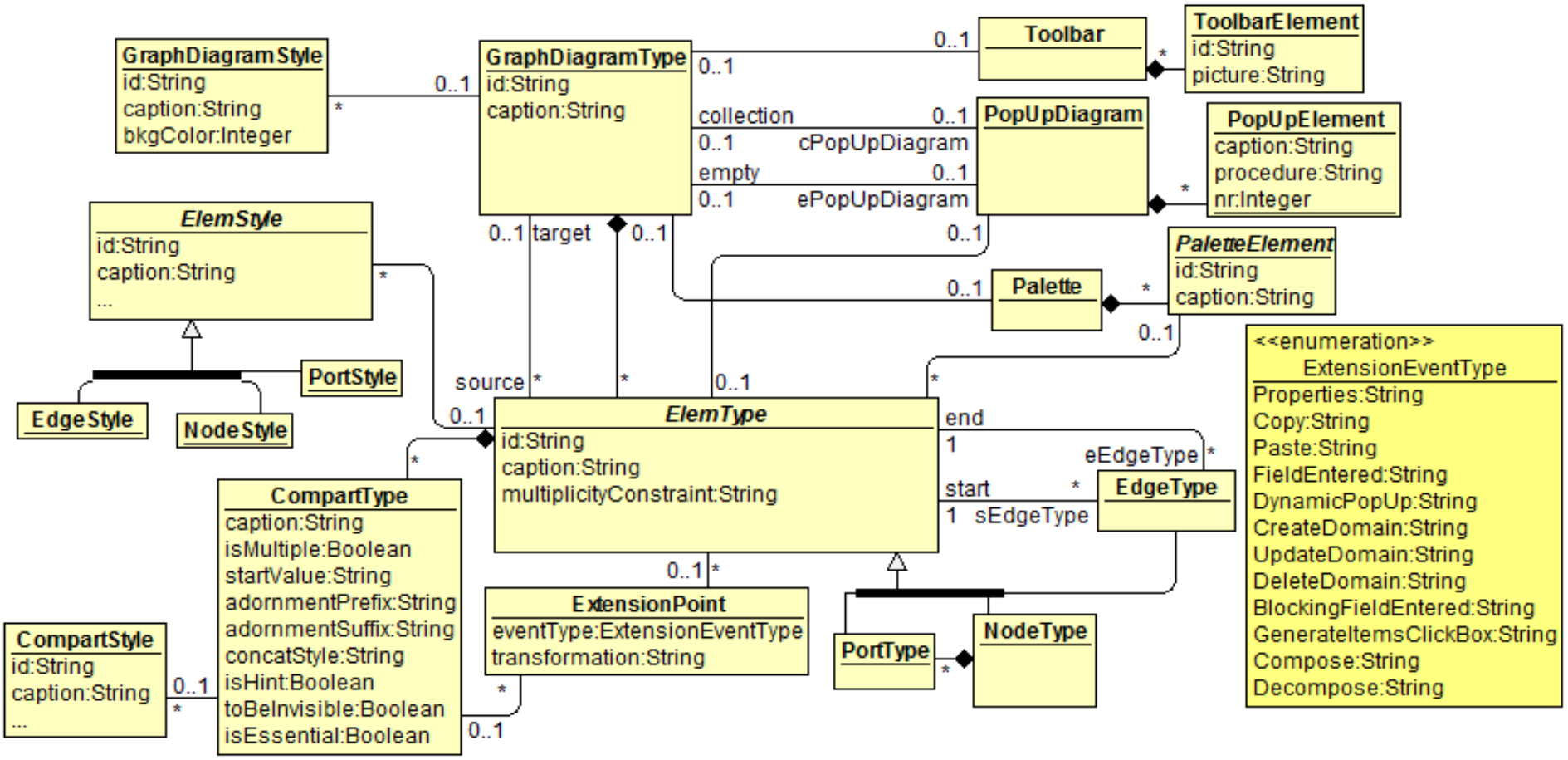
Institute of Mathematics and Computer Science,
University of Latvia

Implementation: Transformation Driven Architecture



- MDA approach. Separation of logical and presentation activities.
- Development of universally re-usable user interface engines; the graphical diagramming engine employs advanced layouting facilities.
- Logical activities based on metamodels and model transformations.
- User actions recorded as *events.* Transformations create *commands* for engines.
- Events and commands stored in the repository (as other metamodel classes).

Implementation: Tool Definition Framework



A framework for defining graphical domain specific tools, built on top of TDA, uses Graph Diagramming Engine, User Dialogue (Form) Engine

Implementation of a concrete tool (e.g. OWLGrEd): instance of TDMM + supporting model transformations for advanced behaviour; visual configurator available

Successful applications: OWLGrEd; several DST; UML class, activity diagram editors