

GraphRep

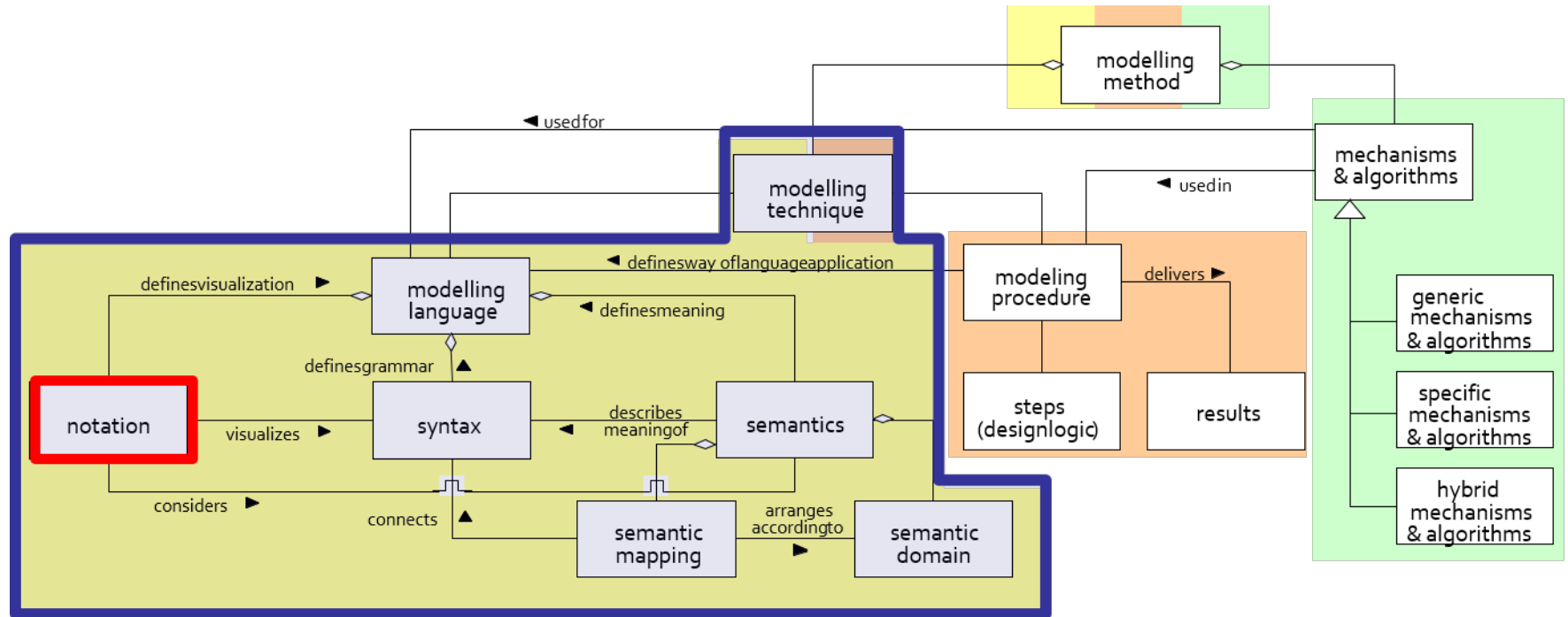
Overview & Goals

1. Application Scenario Notation
2. Modeling Language Notation Specification in ADOxx
3. Exercise Application: Notation for 'Heliport'
4. Lessons Learned

Overview & Goals

1. Application Scenario Notation
2. Modeling Language Notation Specification in ADOxx
3. Exercise Application: Notation for 'Heliport'
4. Lessons Learned

Generic Modeling Method Framework

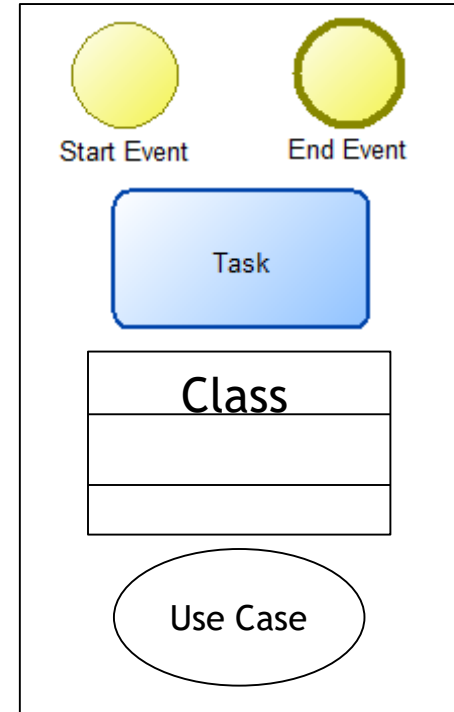
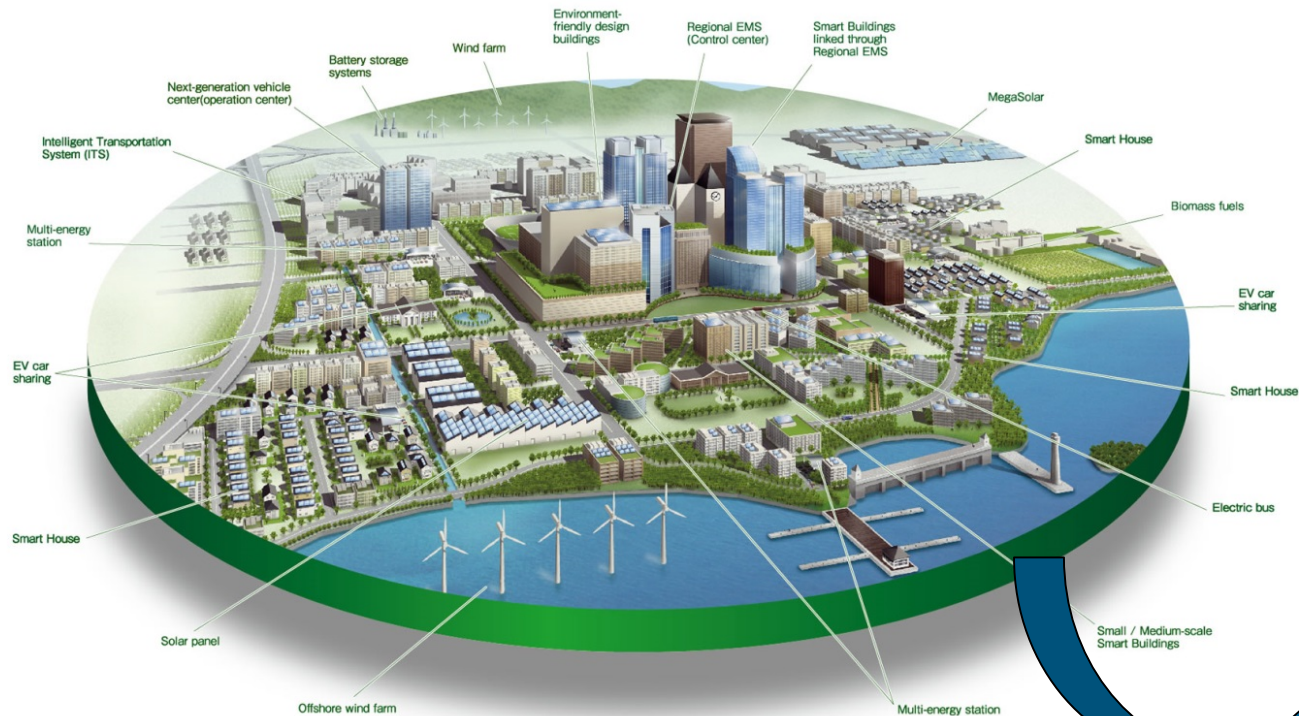


Reference: Karagiannis, D., Kühn, H.: „Metamodelling Platforms“. In Bauknecht, K., Min Tjoa, A., Quirchmayer, G. (Eds.): Proceedings of the Third International Conference EC-Web 2002 – Dexa 2002, Aix-en-Provence, France, September 2002, LNCS 2455, Springer, Berlin/Heidelberg, p. 182 ff.

1. Application Scenario: Modeling Language Notation for Smart Cities

Requirements for Specificity in Notation

How to capture aspects of Smart Cities in standard modeling languages?




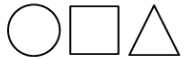




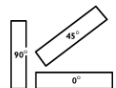
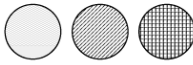
Source: http://www.districtoffuture.eu/uploads/imagenes/imagenes_meetinpoint_smart-city_2b637ab6.jpg



2. Modeling Language Notation Specification in ADOxx

- Static Notation:
 - Semiotic Clarity
 - Perceptual Discriminability
 - Semantic Transparency
 - Complexity Management
 - Cognitive Integration
 - Visual Expressiveness
 - Dual Coding
 - Graphic Economy
 - Cognitive Fitness
- Dynamic Notation:
 - Notation changes (e.g. attribute change)

Eight visual variables that can be used to graphically encode information

PLANAR VARIABLES	RETINAL VARIABLES		
Horizontal Position 	Shape 	Size 	Colour 
Vertical Position 	Brightness 	Orientation 	Texture 

Moody (2009) The Physics of Notation, IEEE Transactions on Software Engineering Vol. 35 No. 6.
<http://doi.ieeecomputersociety.org/cms/Computer.org/dl/trans/ts/2009/06/figures/tts20090607567.gif>

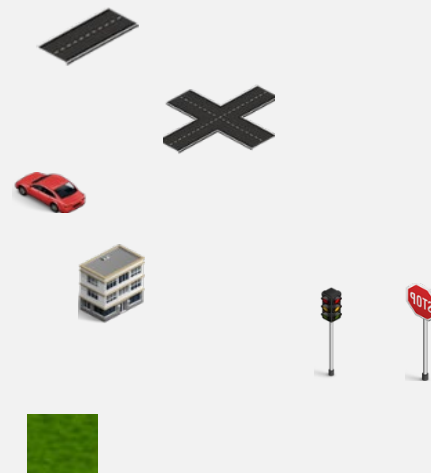
1. Application Scenario: Modeling Language Notation for Smart Cities

Requirements for Specificity in Notation

- What are the Smart City aspects we are interested in?
- How can we map these aspects to modeling classes, relation classes, and attributes?
- What is a proper visualization of the relevant aspects?

Requirements for new modeling concepts:

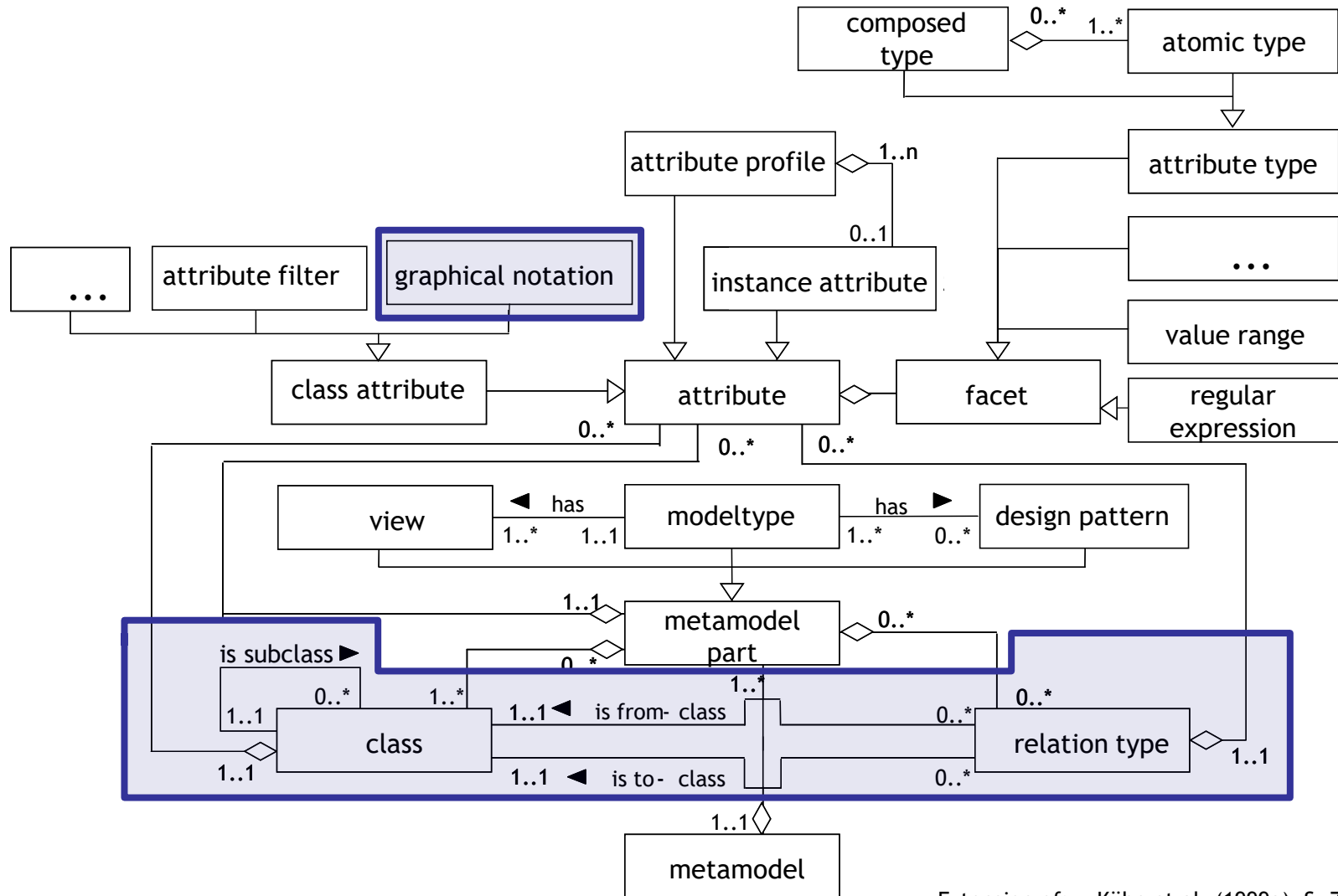
- Streets
- Crossings
- Cars
- Buildings
- Traffic Lights and Traffic Signs
- Green areas



Overview & Goals

1. Application Scenario Notation
- 2. Modeling Language Notation Specification in ADOxx**
3. Exercise Application: Notation for 'Heliport'
4. Lessons Learned

Meta Model of Meta Modeling Language



Extension of: Kühn et al. (1999a), S. 79

2. Modeling Language Notation Specification in ADOxx

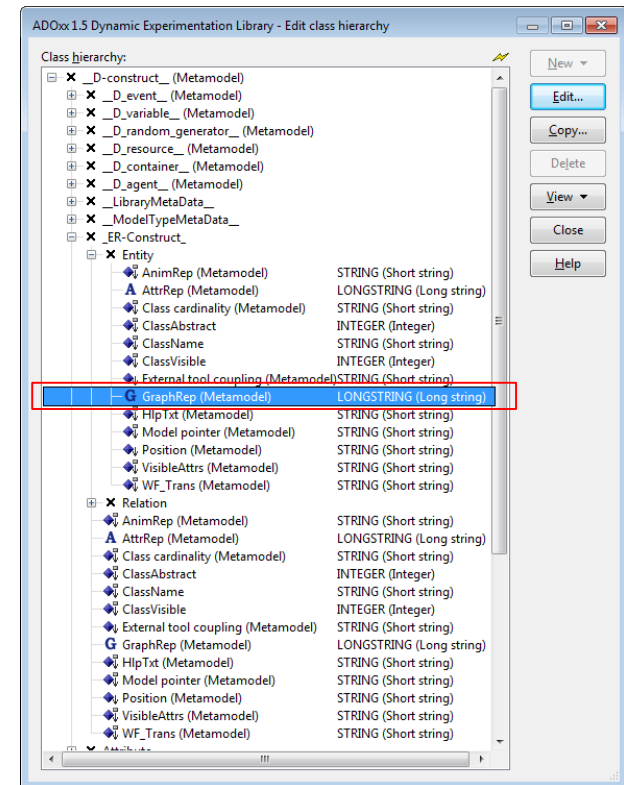
- Photoshop and Paint etc. allow you to create vector as well as pixel graphics
- With standards like e.g. SVG you can “programmatically” create pictures

→ Why is GraphRep necessary?

- GraphRep allows to **change the notation** (=visual appearance) **depending** on the objects state e.g. attributes of object (this is called **dynamic notation**)
- A static notation is a notation that remains the same independently of the objects state (this is called static notation)

2. Modeling Language Notation Specification in ADOxx

- The **graphical visualization** of modeling **classes** and **relation** classes in ADOxx is realized by a special kind of class attribute
 - **GraphRep** (inherited from a generic super class)
- Meta modelers can edit it's value to realize a specialized visualization
- The 'GraphRep' attribute value must follow a specific syntax in order for the platform to parse and draw the graphics
 - Supporting basic **geometrical objects** (rectangles, circles, lines...)
 - Compositions of **basic geometrical objects** (Compounds ..)
 - Integration of **graphics files** into the **visualization** (Icons ..)
- It is interpreted like a procedural program, drawing the shapes in the specified order.



2. Modeling Language Notation Specification in ADOxx

> GraphRep Elements

■ Graph Elements

```
Edge | Start | Middle | End |  
Pen | Fill | Shadow | Stretch | Map | Font |  
ClipRect | ClipRoundRect | ClipPoly | ClipEllipse | ClipOff |  
Point | Line | PolyLine | Arc | Bezier | Curve |  
Rectangle | RoundRect | Polygon | Ellipse | Pie |  
BeginPath | MoveTo | LineTo | BezierTo |  
EndPath | DrawPath |  
Compound | Bitmap | GradientRect | GradientTri |  
Text | Attr | Hotspot |  
Set | Aval | Table | TextBox | AttrBox | BitmapInfo |  
IfStatement | WhileStatement |  
ForNumStatement | ForTokenStatement | Execute.
```

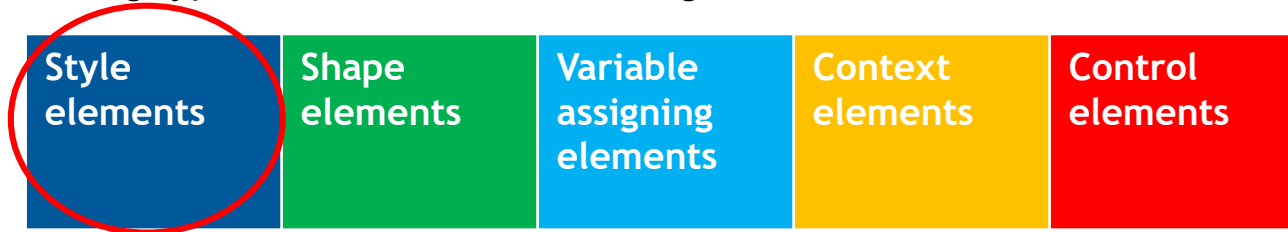
■ GRAPHREP

- The GraphRep definition must start with command GRAPHREP to be valid.
- Syntax Description: <https://www.adoxx.org/live/adoxx-notation-language-graphrep>

2. Modeling Language Notation Specification in ADOxx

> GraphRep Basics

- Class attribute GRAPHREP is of **type long string**, hence the attribute value is a **text that is interpreted as a script by the GRAPHREP interpreter**.
- The following types of elements are distinguished:



- The representation characteristic for following shape elements is modified **by style elements**:
 - **PEN** sets the characteristics of the outline pen for shape elements.
 - **FILL** sets the characteristics of the fill-in brush for shape elements.
 - **SHADOW** switches the additional shadow of shape elements on or off
 - **FONT** sets the font for displayed texts and attribute values.

2. Modeling Language Notation Specification in ADOxx

> GraphRep Basics

- **PEN** determines in which manner the **lines and curves** are drawn, i.e. how strong, in which color and in which style (e.g. dashed line, solid line ..).
 - For shape elements which can be filled, only the outline of the shape is influenced by the current pen.
 - The filling of shapes is controlled by the current fill-in brush, which can be set with **FILL**.



- Shape elements which can not be filled are **POINT**, **LINE**, **POLYLINE**, **ARC** and **CURVE**.
- Fillable elements are **RECTANGLE**, **POLYGON**, **ELLIPSE**, **PIE** and **COMPOUND**.
- For shape elements coordinates (positions) have to be specified. Coordinates here are **relative to the position of the particular object**, i.e. they are added to the object's position.

2. Modeling Language Notation Specification in ADOxx

> GraphRep Commands

- **POINT**
 - Draws a point.
- **LINE / POLYLINE**
 - Draws a single line (**LINE**) or several lines (**POLYLINE**).
- **CURVE / ARC**
 - Draws a curve according to a mathematical function or an arc.
- **POLYGON**
 - Draws a polygon consisting of several straight lines where each corner is defined as a single point.
- **RECTANGLE / ROUNDRECT / ELLIPSE / PIE**
 - A rectangle, rectangle with rounded edges, an ellipse or a segment of an ellipse.
- **COMPOUND**
 - A composite filled Form (from **LINE**, **POLYLINE** und **CURVE**-Elements).

Static Notation Example

GRAPHREP representation
characteristic

SHADOW off
PEN style:dash
FILL color:red
RECTANGLE w:3cm h:1cm



shape

Style elements

Shape elements

Variable
assigning
elements

Context
elements

Control
elements

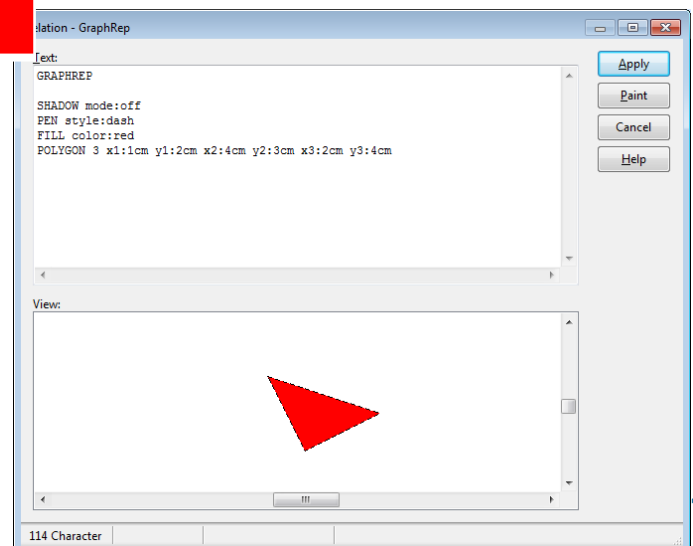
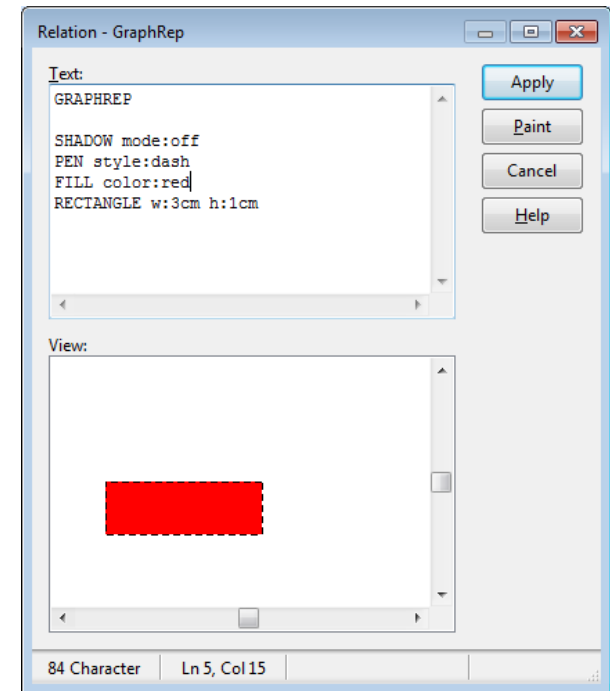
GRAPHREP

representation
characteristic

SHADOW off
PEN style:dash
FILL color:red
POLYGON 3 x1:1cm y1:2cm
x2:4cm y2:3cm x3:2cm
y3:4cm

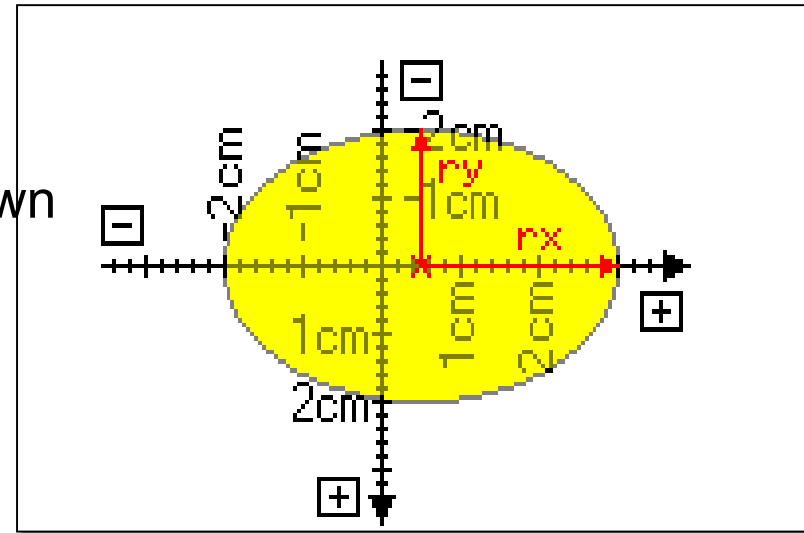


shape



2. Modeling Language Notation Specification in ADOxx > GraphRep Coordinates

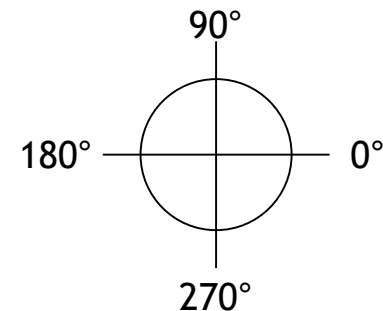
- A **coordinate plane** is used to provide an exact positioning of the GraphRep elements. It is composed of:
 - The null coordinate is in the middle
 - Positive values go to the right and down
 - Negative values go to the left and up



ELLIPSE x:0.5cm y:0cm rx:2.5cm ry:2cm

Hint:

- It is required to specify the Unit (cm or pt). Units in pixels are not possible.
- The direction of rotation progresses counter-clockwise!



Examples

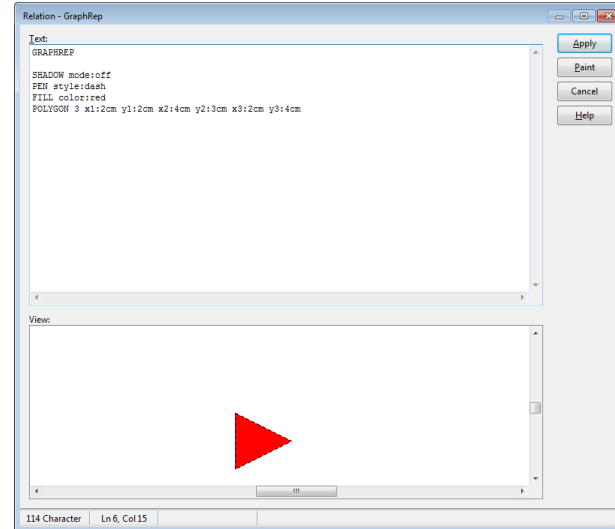
GRAPHREP

SHADOW off

PEN style:dash

FILL color:red

POLYGON 3 x1:2cm y1:2cm
x2:4cm y2:3cm
x3:2cm y3:4cm



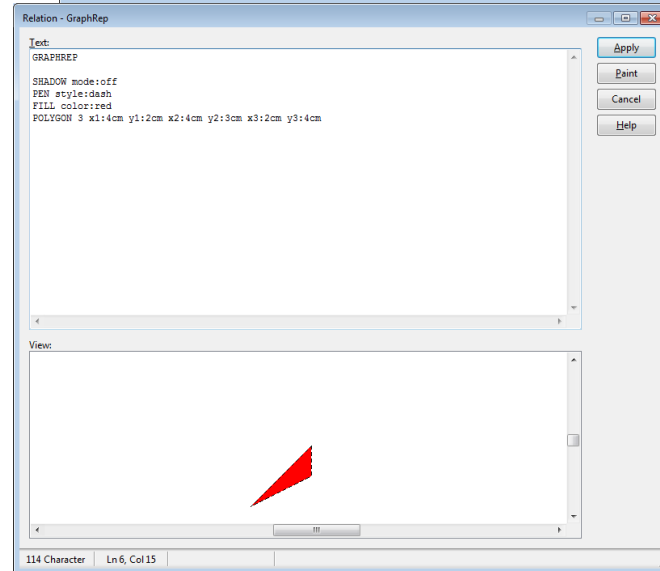
GRAPHREP

SHADOW off

PEN style:dash

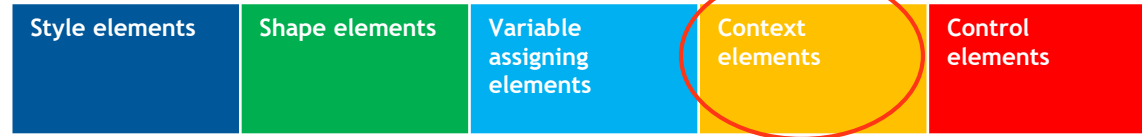
FILL color:red

POLYGON 3 x1:4cm y1:2cm
x2:4cm y2:3cm
x3:2cm y3:4cm



2. Modeling Language Notation Specification in ADOxx

> GraphRep for Relation Classes



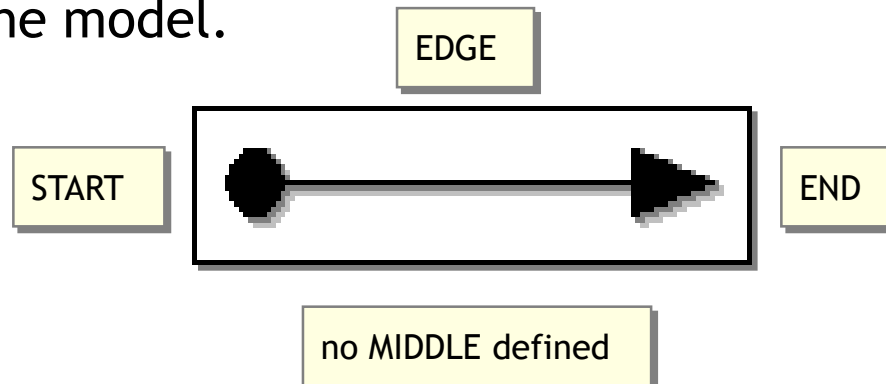
- **Context elements just exist for relations.** They specify whether the starting, the middle or the endpoint of the relation is being defined. Keyword "**START**" defines that the following description refers to the start point of the relation until the next context element **START/MIDDLE/END** is specified. A fourth context element (**EDGE**) triggers the drawing of a relation's edge. This is the line from the starting point via possible bendpoints to the end point of a relation.
- For relations the starting, the middle and the end (point) can be defined. Positions then refer to one of these three points. However, the coordinate system is rotated depending on the direction of the relation instance. On defining a relation's GraphRep, you have to regard the relation as going horizontally from the left to the right. The coordinate system's origin then is the point of the relation for which the graphical representation currently is being defined, i.e. start, middle or end point.

20

2. Modeling Language Notation Specification in ADOxx

> GraphRep for Relation Classes I

- The same commands from normal classes can be used for relation classes as well. In **addition** the following **keywords** are available:
- **EDGE**
 - Defines the representation of the relation edge (line). This command also "draws the line", allowing to put it in front or behind other things.
- **START / MIDDLE / END**
 - This command defines the representation of the important edge parts. If **MIDDLE** is defined, then the middle of the edge can be moved in the model.



2. Modeling Language Notation Specification in ADOxx

> GraphRep for Relation Classes II

```
GRAPHREP rounded:0.05cm  
SHADOW off  
PEN color:red w:0.02cm color:$727272
```

Applies to both

EDGE

GRAPHREP OF EDGE

```
END  
FILL color:red  
POLYGON 3 x1:-0.2cm y1:0.11cm x2:0cm y2:0cm  
x3:-0.2cm y3:-0.11cm
```

GRAPHREP END



2. Modeling Language Notation Specification in ADOxx

> GraphRep for Relation Classes III

```
GRAPHREP rounded:0.05cm  
SHADOW mode:off  
PEN color:red w:0.02cm color:$727272 style:dash
```

EDGE

GRAPHREP OF EDGE

START

FILL color:red

POLYGON 3 x1:-0.2cm y1:0.11cm x2:0cm y2:0cm

x3:-0.2cm y3:-0.11cm

GRAPHREP START

END

FILL color:red

POLYGON 3 x1:-0.2cm y1:0.11cm x2:0cm y2:0cm

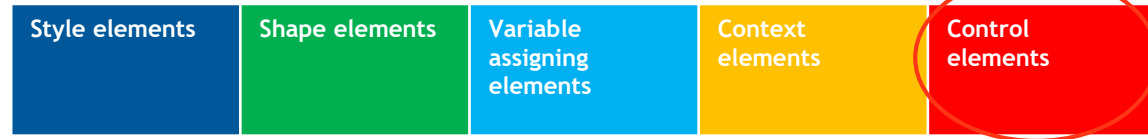
x3:-0.2cm y3:-0.11cm

GRAPHREP END



2. Modeling Language Notation Specification in ADOxx

> GraphRep Commands



- **ATTR**
 - Shows an attribute value on the drawing area (e.g. object name).
- **TEXT**
 - Allows to show a specific text on the drawing area (Letters, Symbols ...).
- **BITMAP**
 - Allows to embed a picture.
- **TABLE**
 - Creates a table for structuring the attribute representation of an object.
- **HOTSPOT**
 - Allows to access some attributes special function on the drawing area, e.g. follow an InterRef or execute a Programcall.

Example

GRAPHREP

SHADOW off

PEN style:dash

FILL color:red

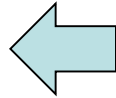
POLYGON 3 x1:1cm y1:2cm

x2:4cm y2:3cm

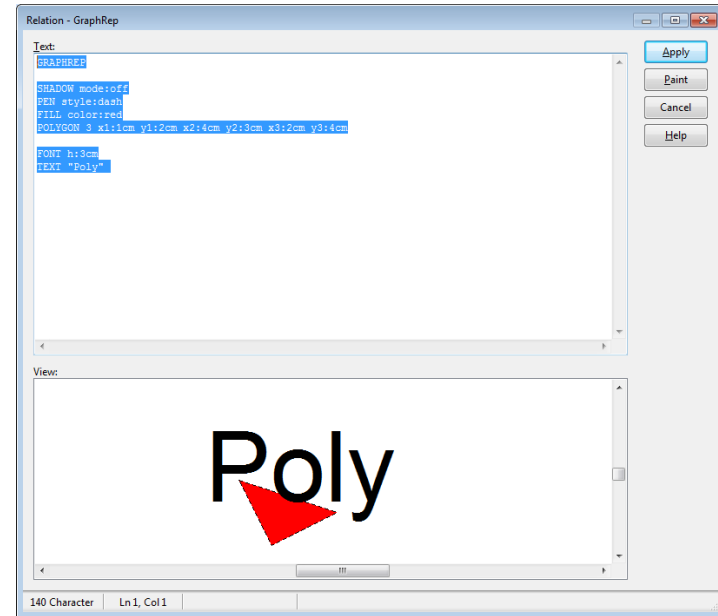
x3:2cm y3:4cm

FONT h:3cm

TEXT "Poly"



text



FONT

- Defines the font style/color for drawn text.

2. Modeling Language Notation Specification in ADOxx

> GraphRep Structural Commands



■ SET

- Sets a variable with a constant or the result of an expression, which in turn can contain variables.

■ AVAL

- Sets variables with the values from an attribute of the instantiated object.

■ IF / ELSIF / ELSE

- Allows to change the representation based on values.

■ BITMAPINFO

- Reads the height and width of a bitmap file, allowing to properly represent it.

■ TEXTBOX / ATTRBOX

- Similar to TEXT and ATTR. However instead of drawing the values it sets specific variables with the rectangle area they would need.

Examples

GRAPHREP

```
SHADOW off
PEN style:dash
FILL color:red
POLYGON 3 x1:4cm y1:2cm x2:4cm
y2:3cm x3:2cm y3:4cm
```

AVAL var:"Name"

```
IF (LEN(var)>5) {
  TEXT "<too long Name>"
}
ELSE {
  TEXT (var)
}
```

<too long Name>

Name

Var is not shown in the GraphRep Editor!

Examples

GRAPHREP

SHADOW off

PEN style:dash

FILL color:red

POLYGON 3 x1:4cm y1:2cm

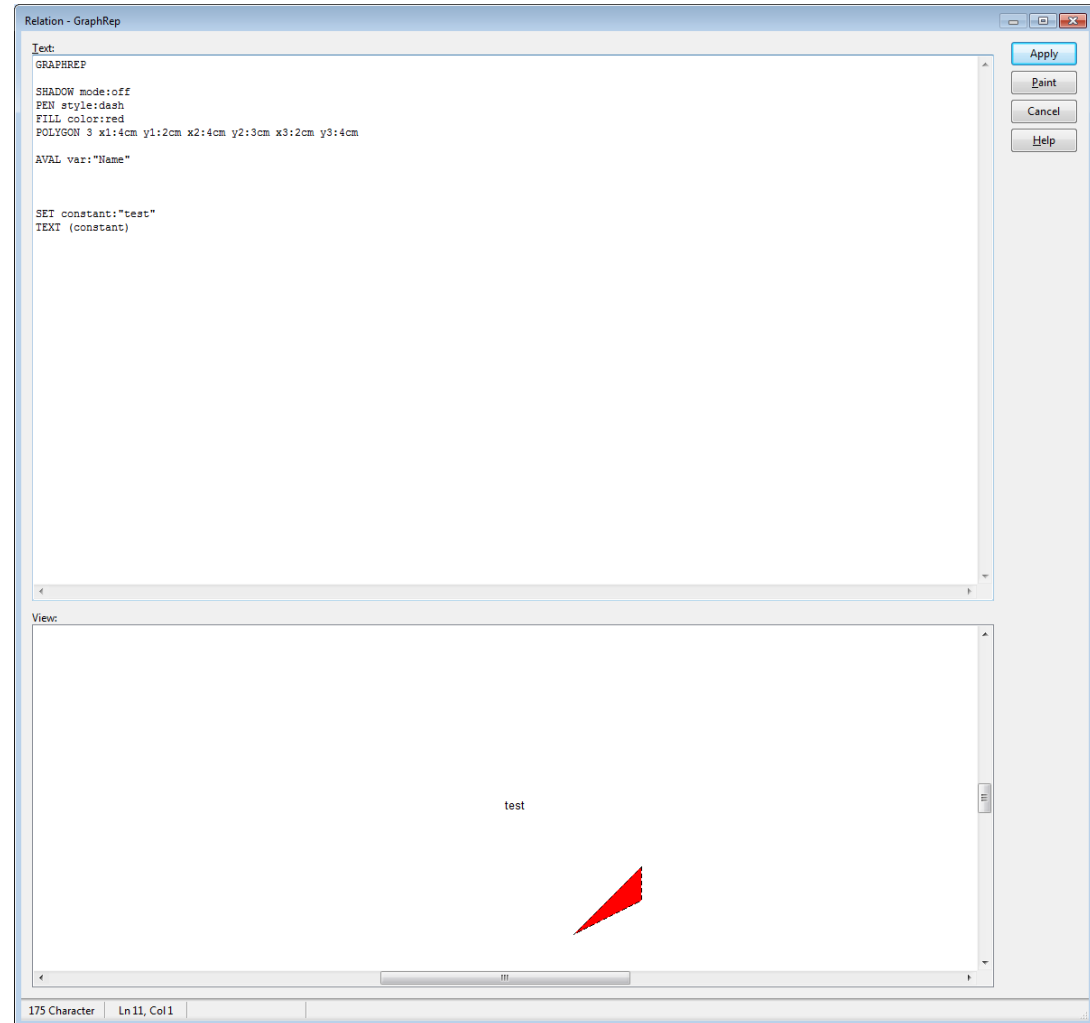
x2:4cm y2:3cm

x3:2cm y3:4cm

AVAL var:"Name"

SET constant:"test"

TEXT (constant)



Small Exercise

Create a Rectangle

It should be green if the name attribute has the value „Name“ and red in all other cases

```
GRAPHREP
```

```
SHADOW off
```

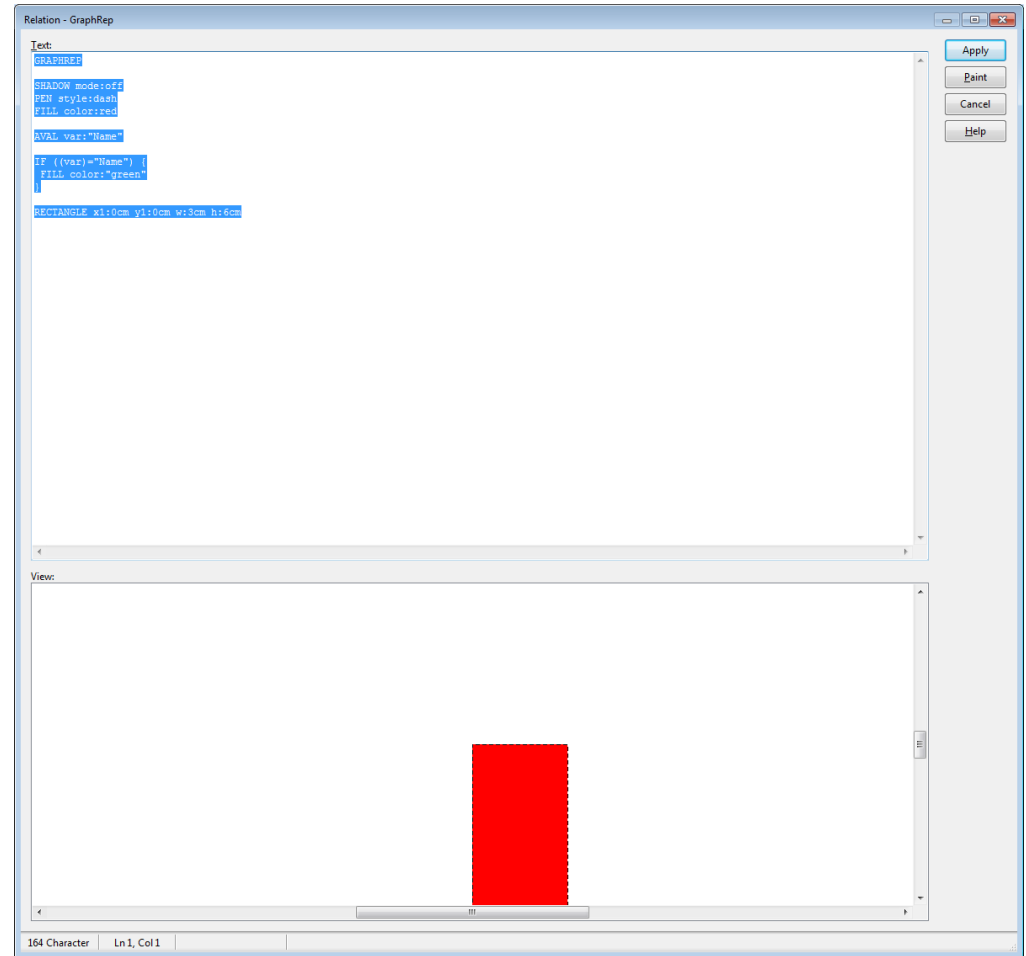
```
PEN style:dash
```

```
FILL color:red
```

```
AVAL var:"Name"
```

```
IF (var="Name") {  
  FILL color:"green"  
}
```

```
RECTANGLE w:3cm h:6cm
```



Small Exercise II

Create a Rectangle

It should be yellow if the name attribute has the value „5“ and red in all other cases

GRAPHREP

SHADOW mode:off

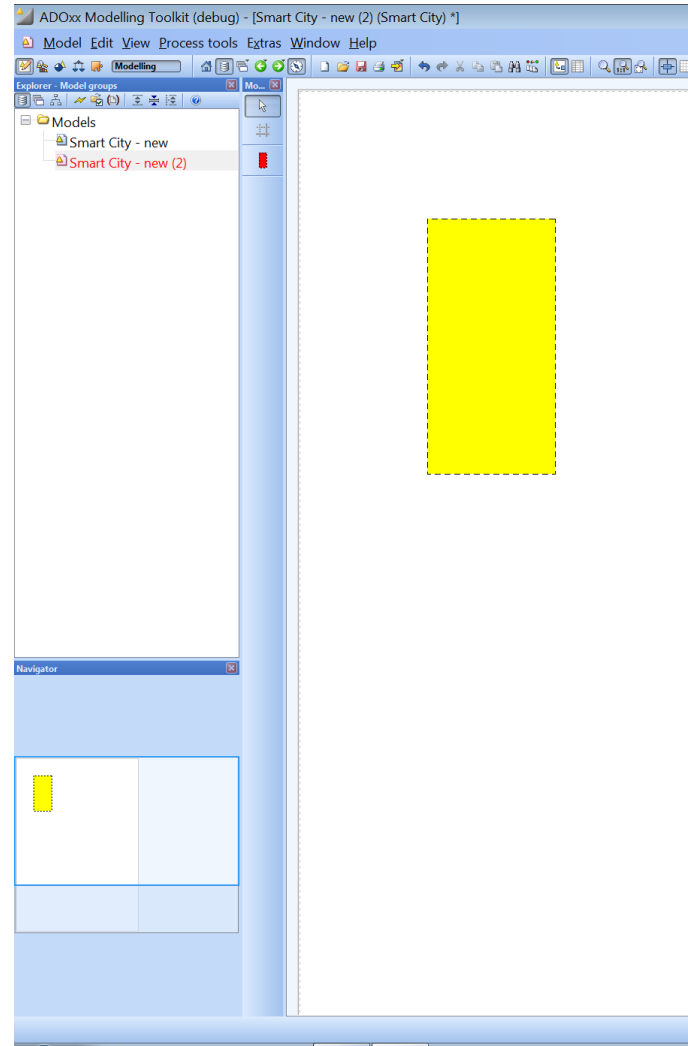
PEN style:dash

FILL color:red

AVAL var:"Name"

```
IF ((VAL var)=5) {  
  FILL color:"yellow"  
}
```

RECTANGLE w:3cm h:6cm



2. Modeling Language Notation Specification in ADOxx > GraphRep Example

GRAPHREP

#Container Rectangle

RECTANGLE x:-1.5cm y:-0.5cm w:3cm h:1cm

#Arrow Lines

PEN style:dash

LINE x1:-0.8cm x2:0.8cm y1:-0.2cm y2:-0.2cm

LINE x1:-0.8cm x2:0.8cm y1:0.2cm y2:0.2cm

#Arrow Ends

PEN style:solid

LINE x1:-0.8cm x2:-0.6cm y1:-0.2cm y2:-0.1cm

LINE x1:-0.8cm x2:-0.6cm y1:-0.2cm y2:-0.3cm

LINE x1:0.8cm x2:0.6cm y1:0.2cm y2:0.1cm

LINE x1:0.8cm x2:0.6cm y1:0.2cm y2:0.3cm

#Right actor

ELLIPSE x:1.1cm y:-0.2cm rx:0.15cm ry:0.15cm

LINE x1:1.1cm x2:1.1cm y1:-0.05cm y2:0.2cm

LINE x1:1.1cm x2:0.95cm y1:0.2cm y2:.3cm

LINE x1:1.1cm x2:1.25cm y1:0.2cm y2:.3cm

LINE x1:0.95cm x2:1.25cm

#Left actor

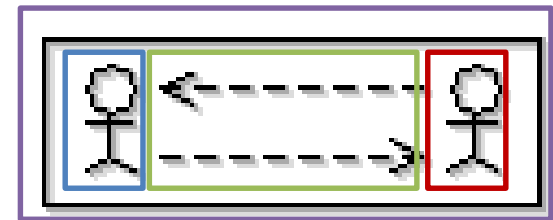
ELLIPSE x:-1.1cm y:-0.2cm rx:0.15cm ry:0.15cm

LINE x1:-1.1cm x2:-1.1cm y1:-0.05cm y2:0.2cm

LINE x1:-1.1cm x2:-0.95cm y1:0.2cm y2:.3cm

LINE x1:-1.1cm x2:-1.25cm y1:0.2cm y2:.3cm

LINE x1:-0.95cm x2:-1.25cm






2. Modeling Language Notation Specification in ADOxx > GraphRep Repository

- More free available GraphRep examples can be found at the **ADOxx GraphRep Repository**
 - <https://www.adoxx.org/live/adoxx-graphrep-repository-wiki/-/wiki/GRAPHREP+Repository/FrontPage>

ADOxx GraphRep Repository

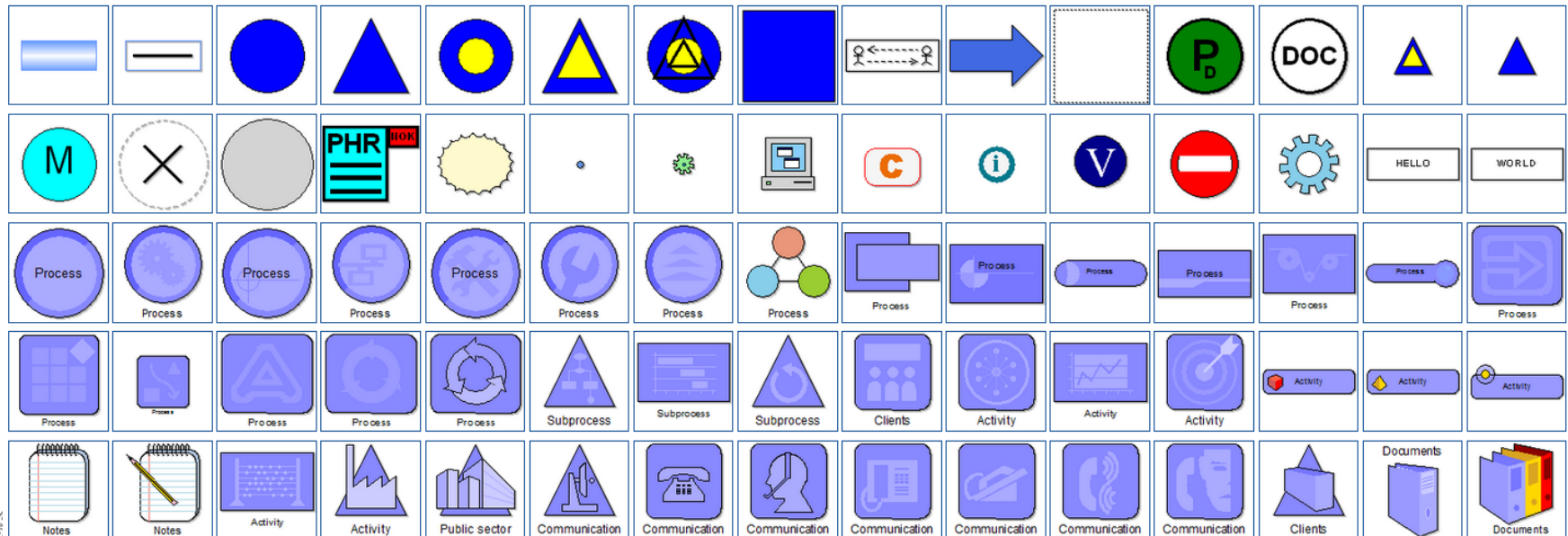
(Redirected from FrontPage)

Tags:  graphrep

 Details  Print

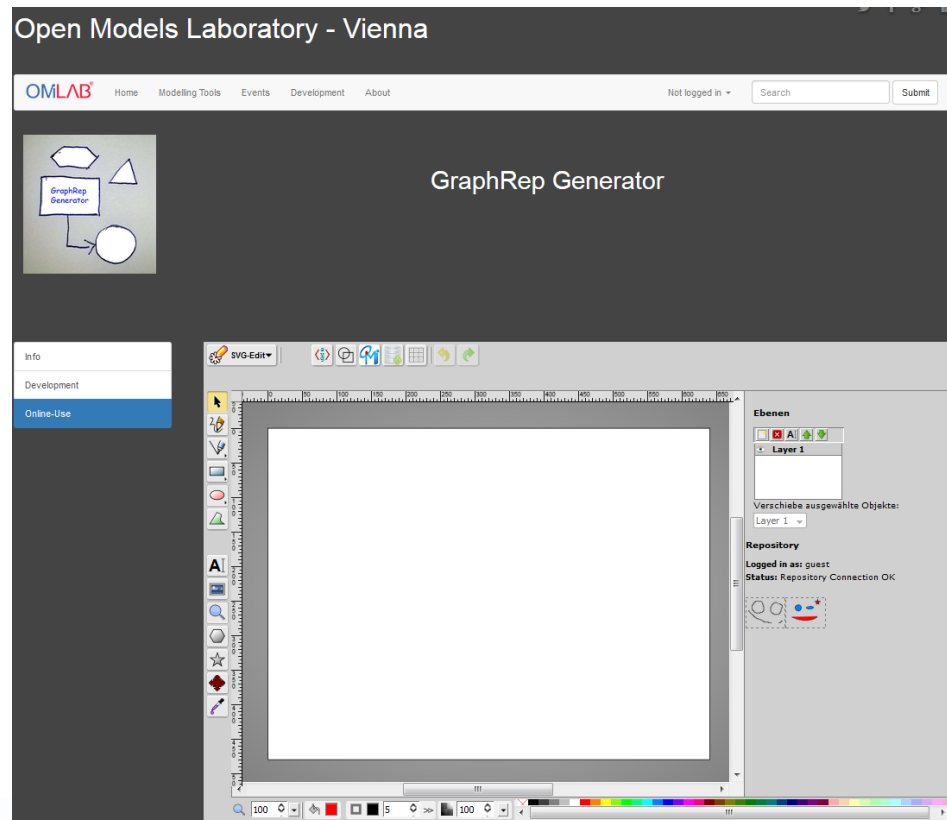
The ADOxx GraphRep repository collects implementation of graphical representation from different scenarios and projects and provides them to the community. As a community member, feel free to add, revise, use, modify, comment and rate the GraphReps available in the repository.

CLASSES



2. Modeling Language Notation Specification in ADOxx > GraphRepGenerator

- Implementation of GraphRep code can be supported by the **GraphRepGenerator**
- <http://vienna-omilab.dke.univie.ac.at/GraphRepGenerator/editor/svg-editor.html>



Excuse: Use Font Symbols

- You can use font symbols for your notation
- Example:

GRAPHREP

```
FONT "Webdings" h:28pt style:bold color:black  
TEXT "u"
```



- Only use default fonts which are installed on all machines!

Overview & Goals

1. Application Scenario Notation
2. Modeling Language Notation Specification in ADOxx
- 3. Exercise Application: Notation for 'Heliport'**
4. Lessons Learned

3. Exercise Application: Static Heliport notation

Use the “ADOxx 1.5 Experimentation Library”

- I. Create a new class called „Heliport“
- II. Create a graphical visualization for Heliport
 - a. Manual GraphRep specification
 - b. GraphRepGenerator utilization
- III. Add the MyHeliport class to a ‘Smart City’ model type
- IV. Create Smart City models and use the new ‘Heliport’ class



Overview & Goals

1. Application Scenario Notation
2. Modeling Language Notation Specification in ADOxx
3. Exercise Application: Notation for 'Heliport'
4. **Lessons Learned**

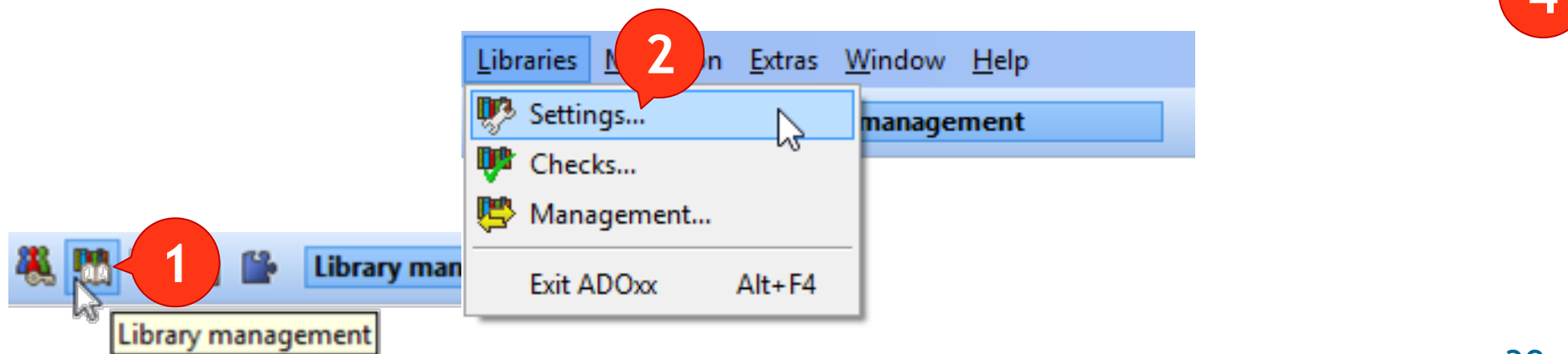
4. Lessons learned

- Domain-specific requirements also affect the graphical visualization of modeling classes and relation classes
- The notation in ADOxx is specified in the GraphRep attribute
- How to create a notation for Modeling Classes
- How to create graphical instances of Modeling Classes in the Modeling Toolkit

Solution Exercise Application: Static Heliport notation

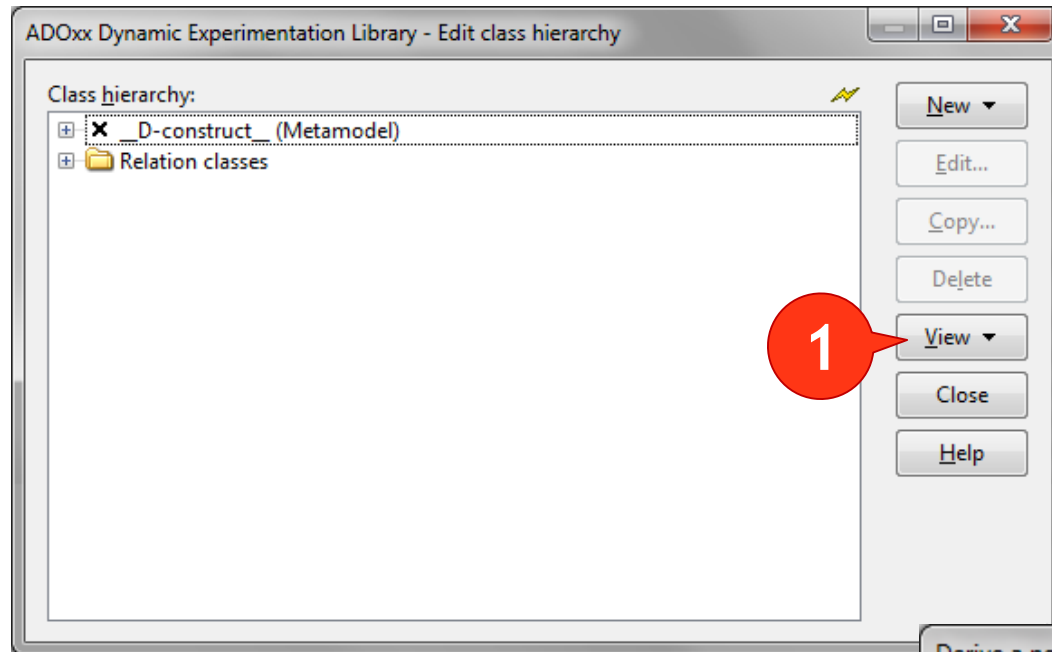
I. Create a new modeling class called „Heliport“

1. Start the Development Toolkit
2. Click on „Library Management“
3. Click on „Settings“
4. Click on a library, e.g., „ADOxx Experimentation Library“
5. Select the „Dynamic Experimentation Library“
6. Click on „Class hierarchy“

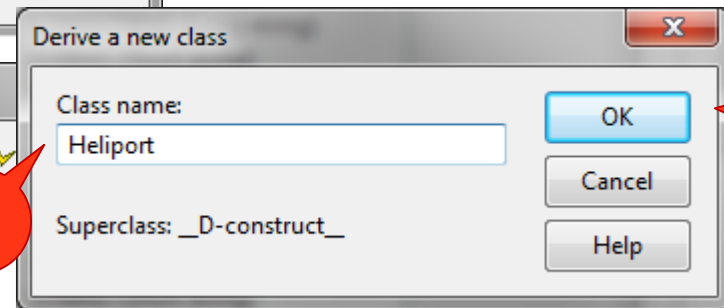
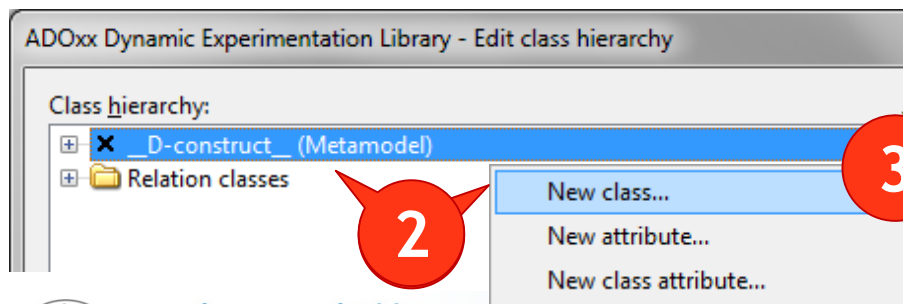


Solution Exercise Application: Static Heliport notation

I. Create a new modeling class called „Heliport“

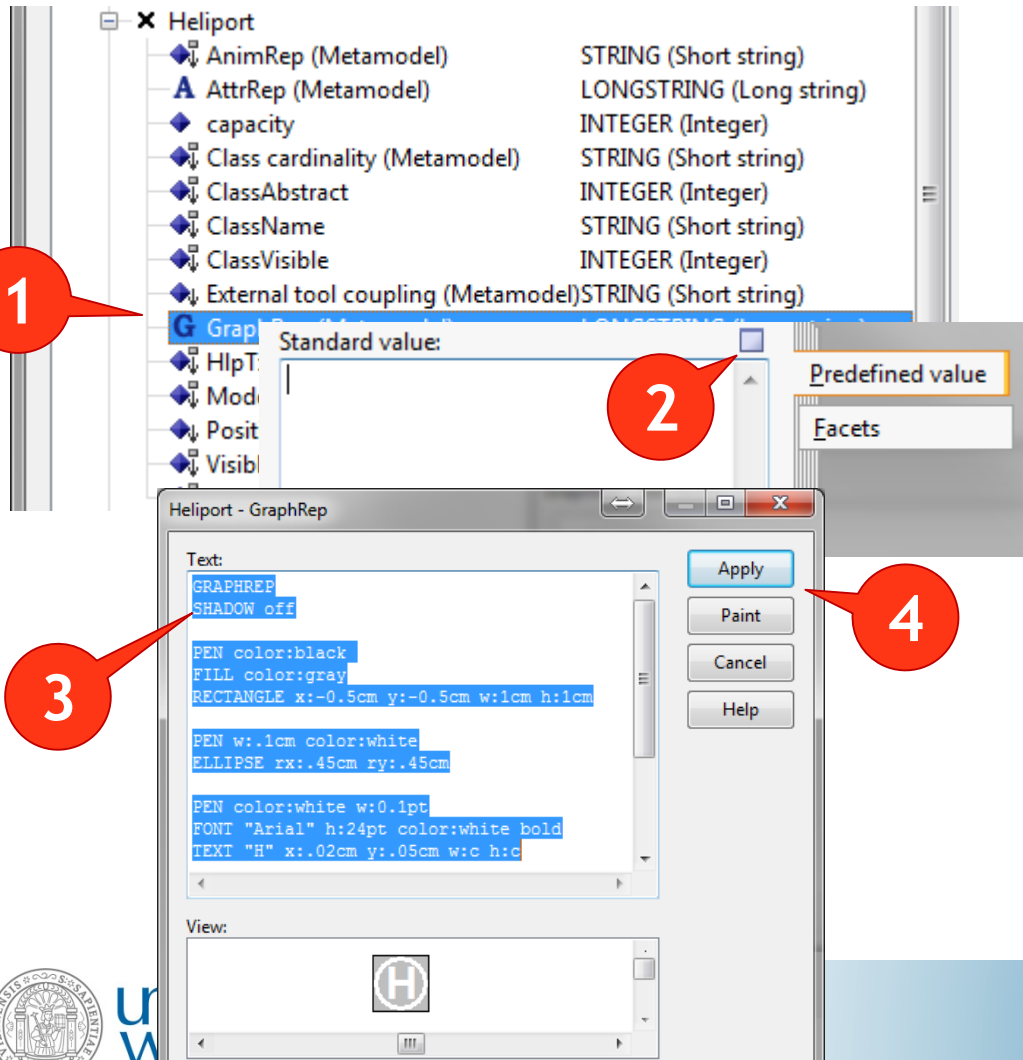


1. Adjust the View
 - Enable “Metamodel” and “Class hierarchy”
2. Right-click on “_D_Construct_ -> “New class...”
3. Enter “Heliport” as name for the new class
4. Click on “Ok”



Solution Exercise Application: Static Heliport notation

Ila. Create a graphical visualization for 'Heliport'



1. Double-click on the "GraphRep" attribute of the class "Heliport"
2. Click on "Dialog"
3. Copy this code into the „Text“ area:

```
GRAPHREP
```

```
SHADOW off
```

```
PEN color:black
```

```
FILL color:gray
```

```
RECTANGLE x:-0.5cm y:-0.5cm  
w:1cm h:1cm
```

```
PEN w:.1cm color:white
```

```
ELLIPSE rx:.45cm ry:.45cm
```

```
PEN color:white w:0.1pt
```

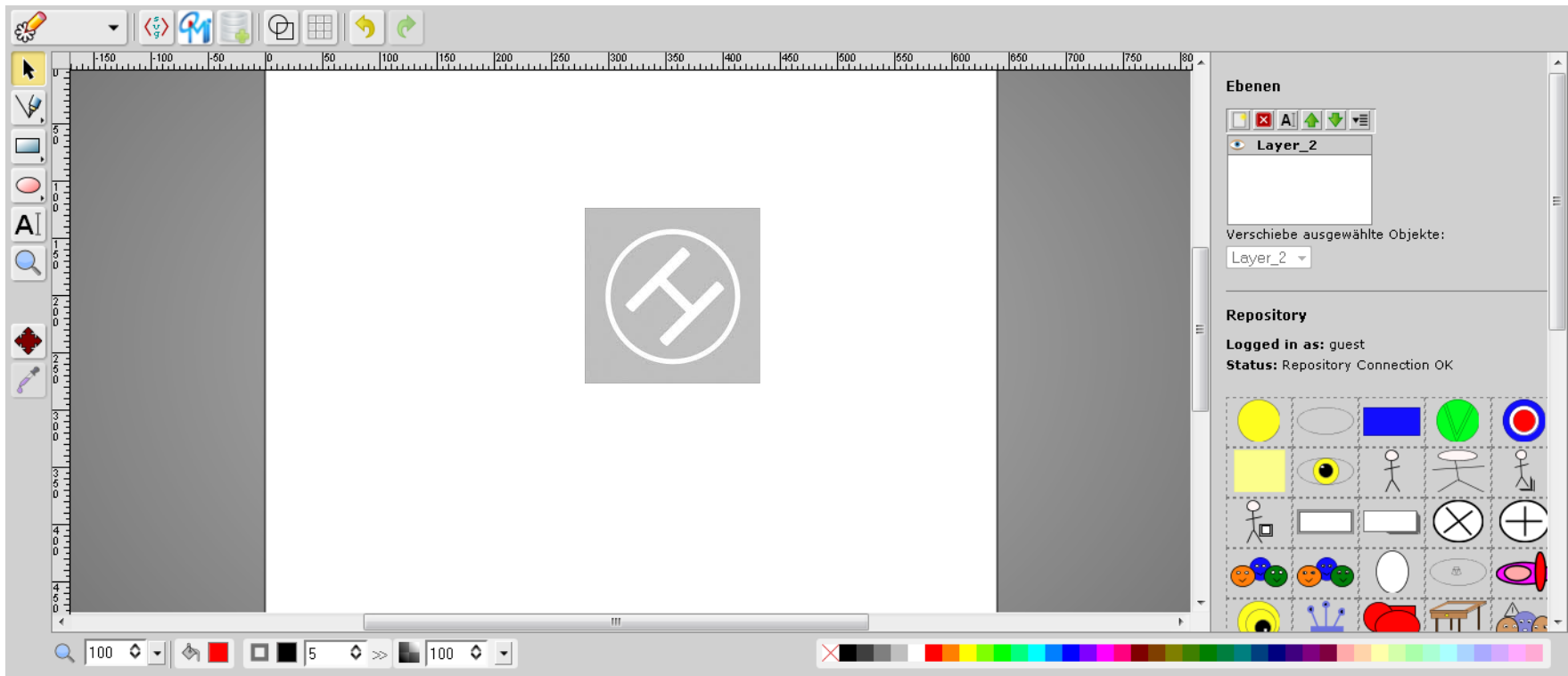
```
FONT "Arial" h:24pt color:white  
bold
```

```
TEXT "H" x:.02cm y:.05cm w:c h:c
```

4. Press "Apply", "Close" and "Save changes!"

Solution Exercise Application: Static Heliport notation

IIb. Create a graphical visualization for 'Heliport' with the **GraphRep Generator**



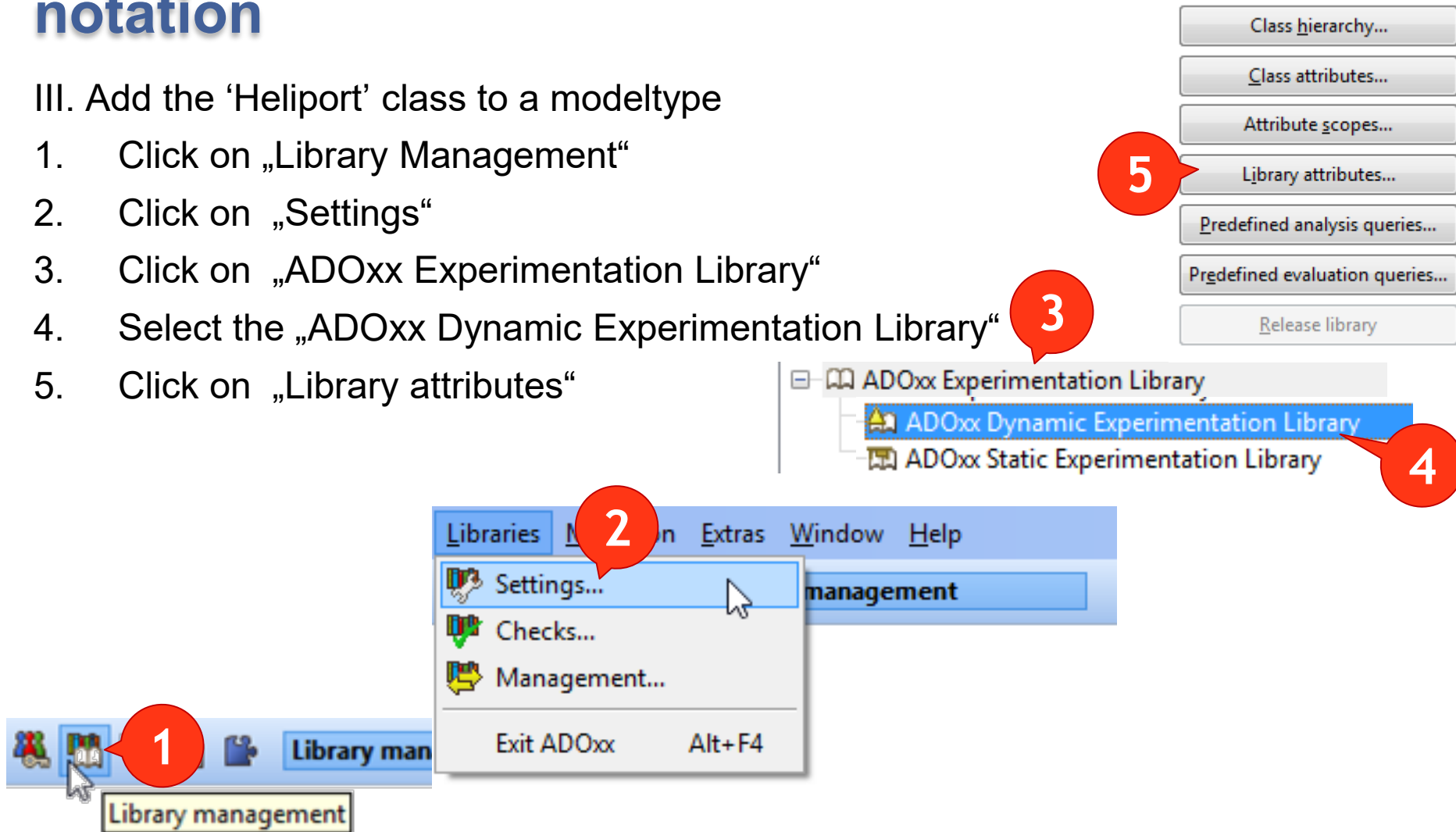
<http://vienna-omilab.dke.univie.ac.at/GraphRepGenerator/editor/svg-editor.html>

42

Solution Exercise Application: Static Heliport notation

III. Add the 'Heliport' class to a modeltype

1. Click on „Library Management“
2. Click on „Settings“
3. Click on „ADOxx Experimentation Library“
4. Select the „ADOxx Dynamic Experimentation Library“
5. Click on „Library attributes“



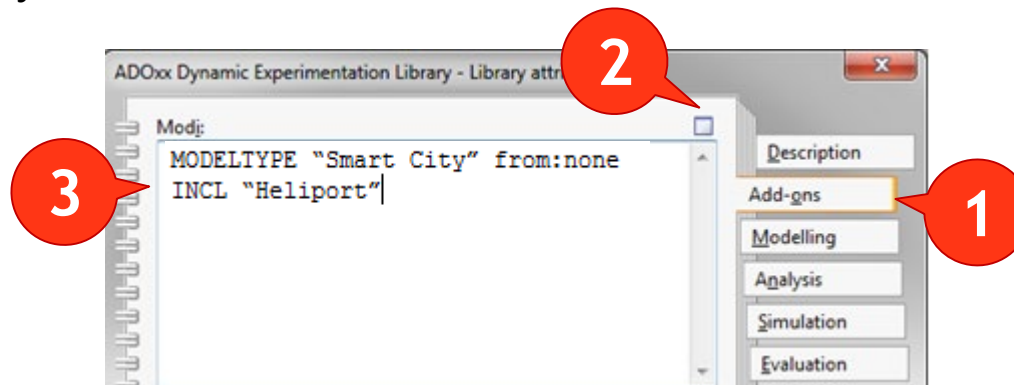
Solution Exercise Application: Static Heliport notation

III. Add the 'Heliport' class to a modeltype

1. Click on „Add-ons“
2. Click on „Dialog“
3. Add the text

```
MODELTYPE "Smart City" from:none  
INCL "Heliport"
```

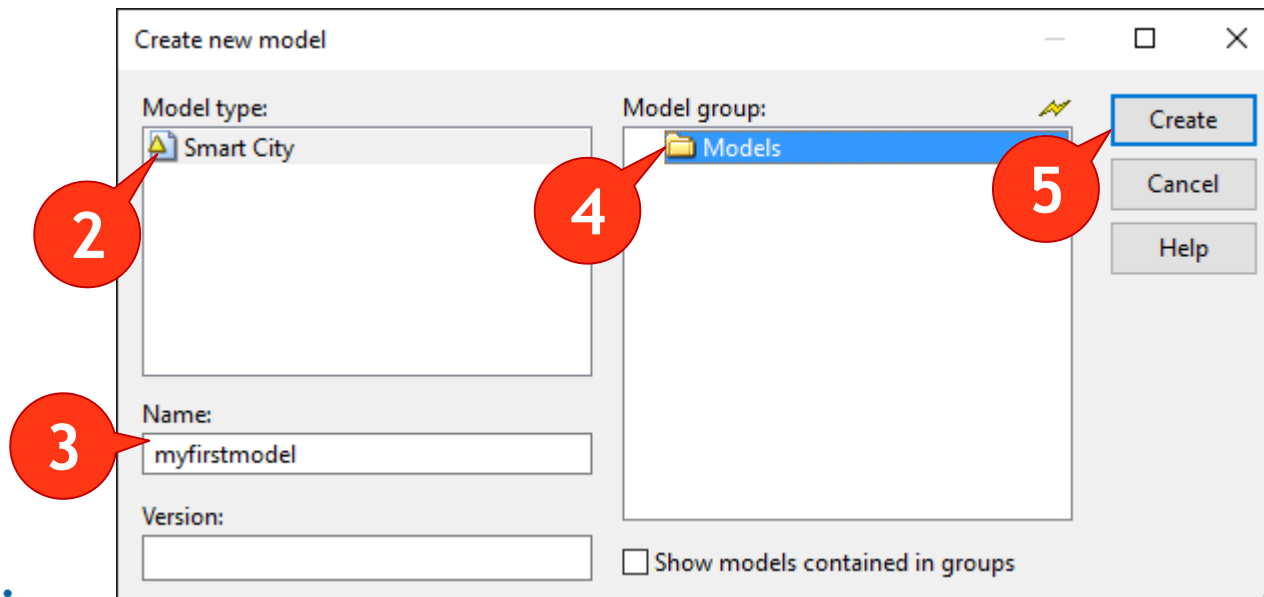
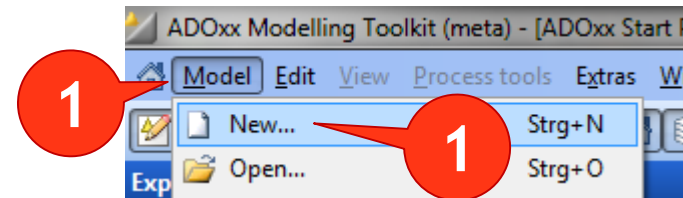
4. Click on „Apply“



Solution Exercise Application: Static Heliport notation

IV. Create Smart City models and use the new 'Heliport' class

1. Click on „Model“ => “New”
2. Select the „SmartCity” model type
3. Define a name for the new model
4. Click on the „Models” model group
5. Click on “Create”



Solution Exercise Application: Static Heliport notation

