
asseco
CENTRAL EUROPE

asseco
CENTRAL EUROPE

LIDS7 MDG Extension for EA

Workshop – SAMO Logical Data model

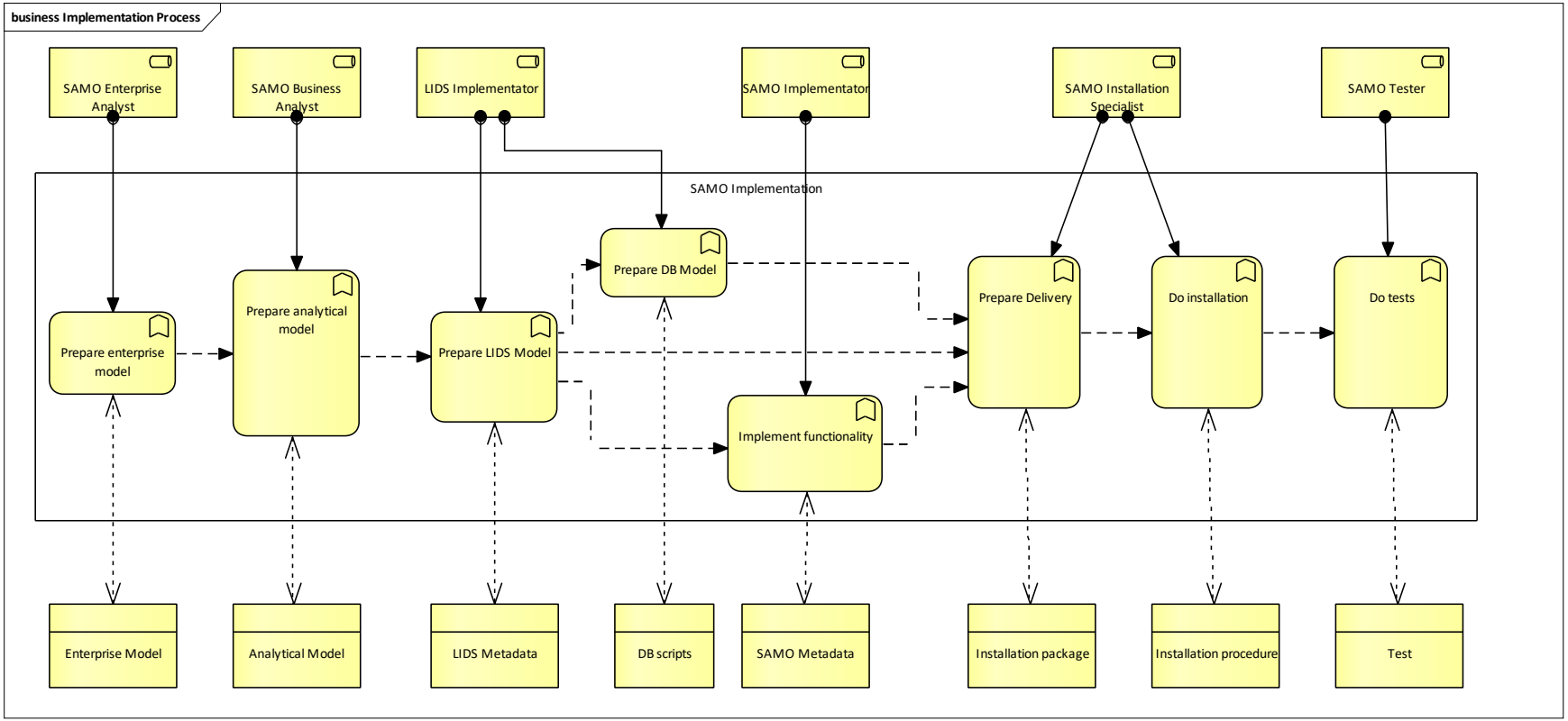
Radim Bystřický, Vojtěch Kučera – GIS&Utility

Bratislava, 4.2.2020

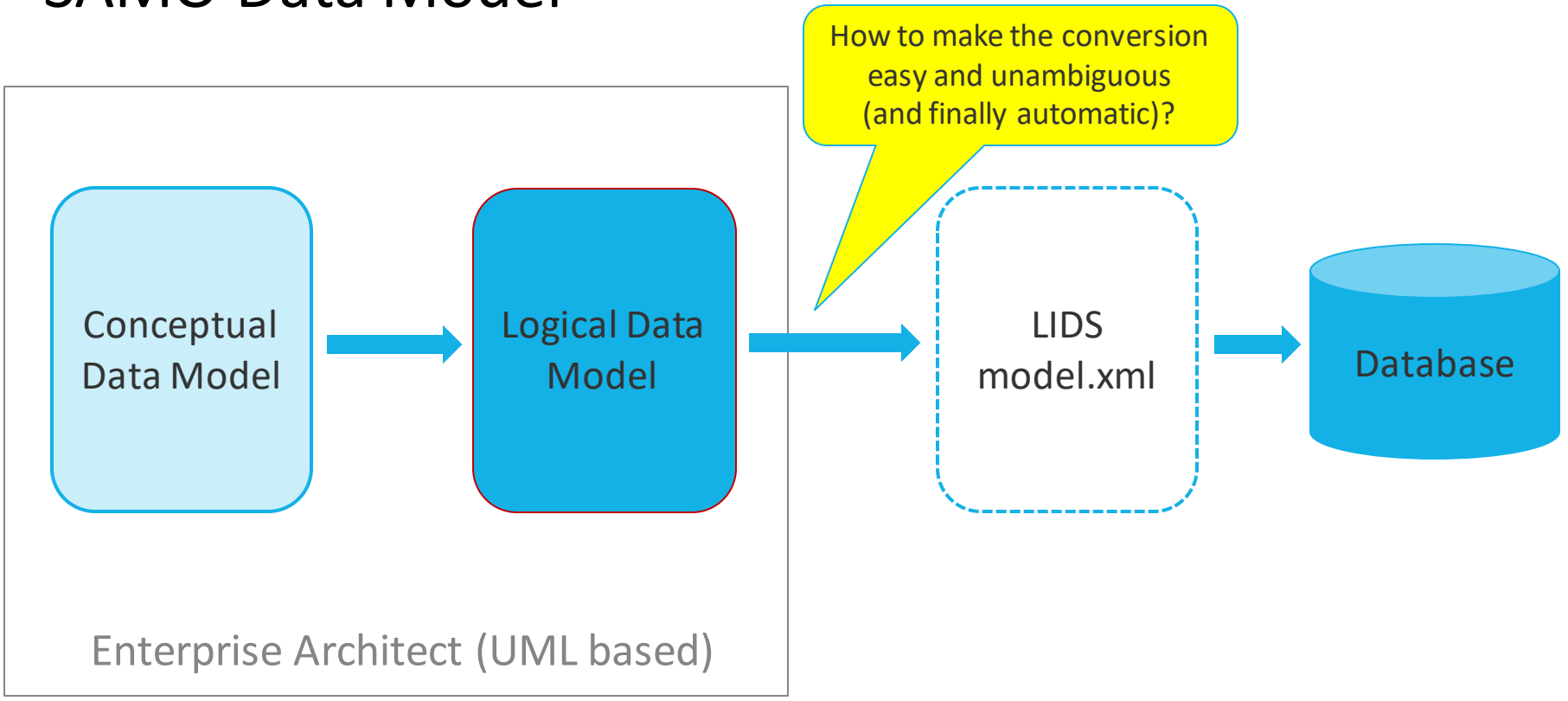
Content:

1. Introduction to LIDS7 MDG Technology
2. How to model Codelists
3. How to model Feature Types
4. Data Types
5. Naming Convention for LIDS model
6. Basic transformation rules
7. Advanced transformation rules
8. Patterns
9. „Production line“ for LIDS model

Implementation line



SAMO Data Model



Introduction to LIDS7 MDG Technology („LIDS Extension“)

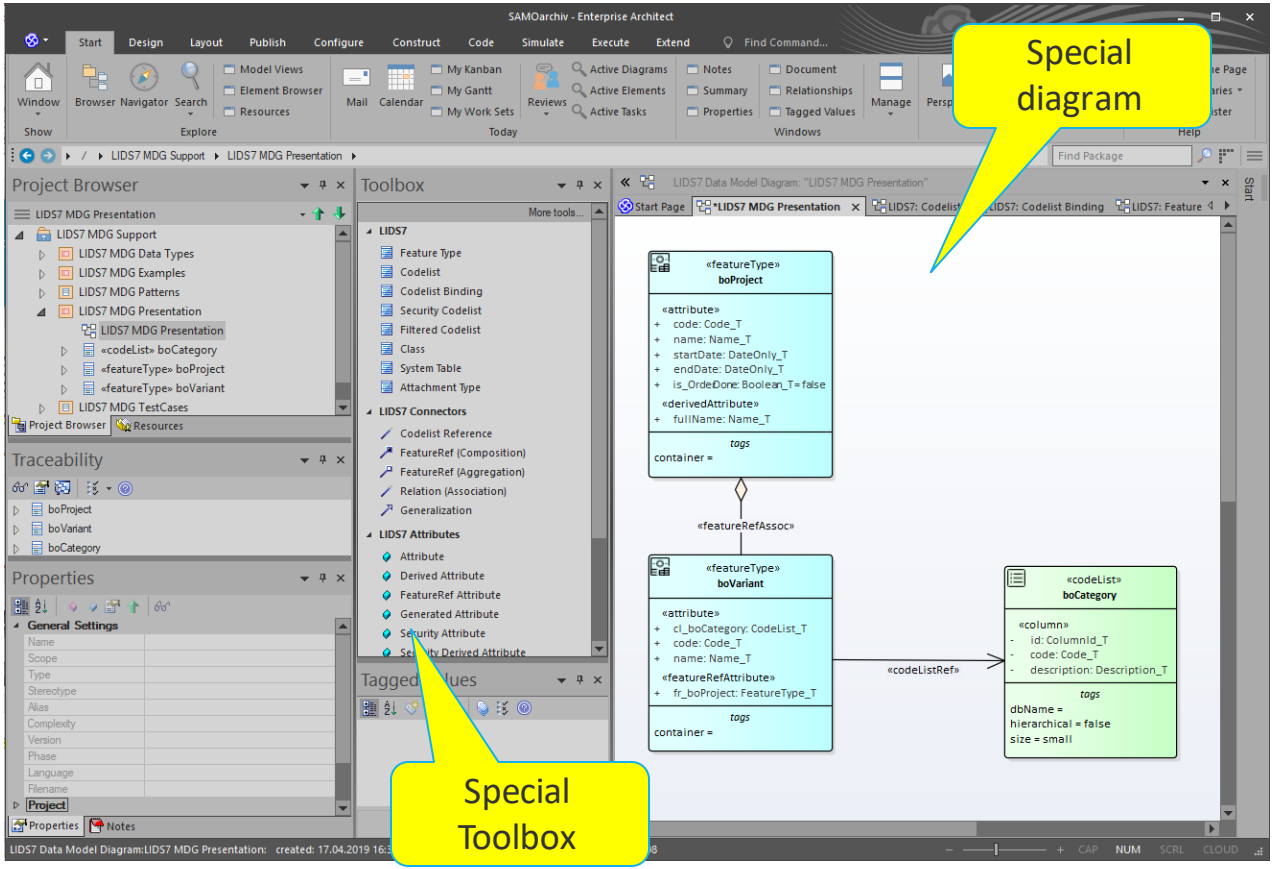
asseco
CENTRAL EUROPE

SPARX MDG Technology

„***MDG Technology*** is a vehicle for providing access to the resources of either a commercially-available technology or a technology that you have created yourself. Such resources include a wide range of facilities and tools, such as **UML Profiles**, code modules, scripts, **Patterns**, images, **Tagged Value Types**, report templates, linked document templates and **Toolbox pages**.

Using Enterprise Architect, you can develop models based on the standard UML specifications, and you can ***extend the core UML*** structures using UML-supported mechanisms such as **Tagged Values**, **Stereotypes**, **Profiles** and **Design Patterns**. These facilities are within the Enterprise Architect core technologies, and you can activate and use further Model Driven Generation (MDG) Technologies that are either integrated with the system or available from external locations.“

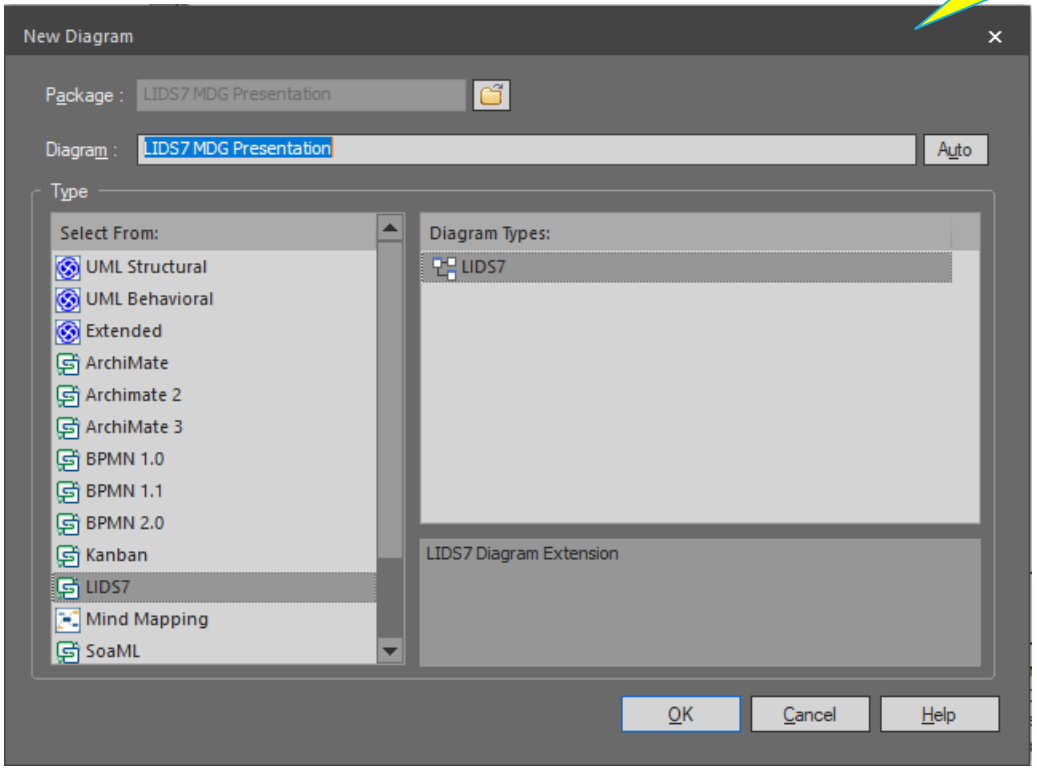
LIDS7 MDG Technology in EA

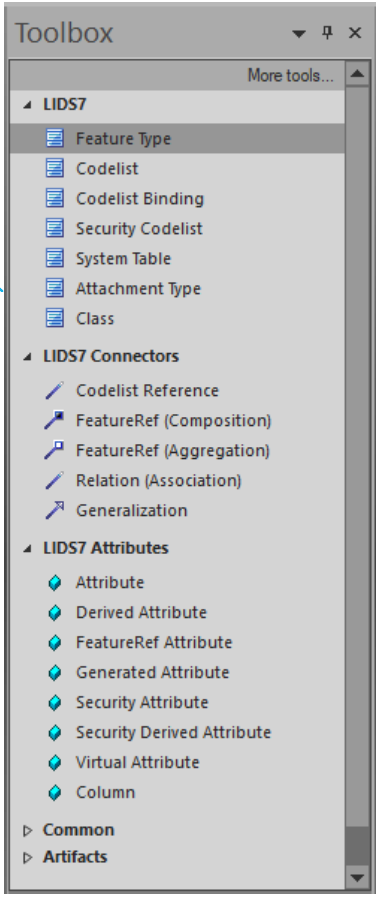


LIDS7 Diagram + Toolbox

Add new diagram

Related Toolbox



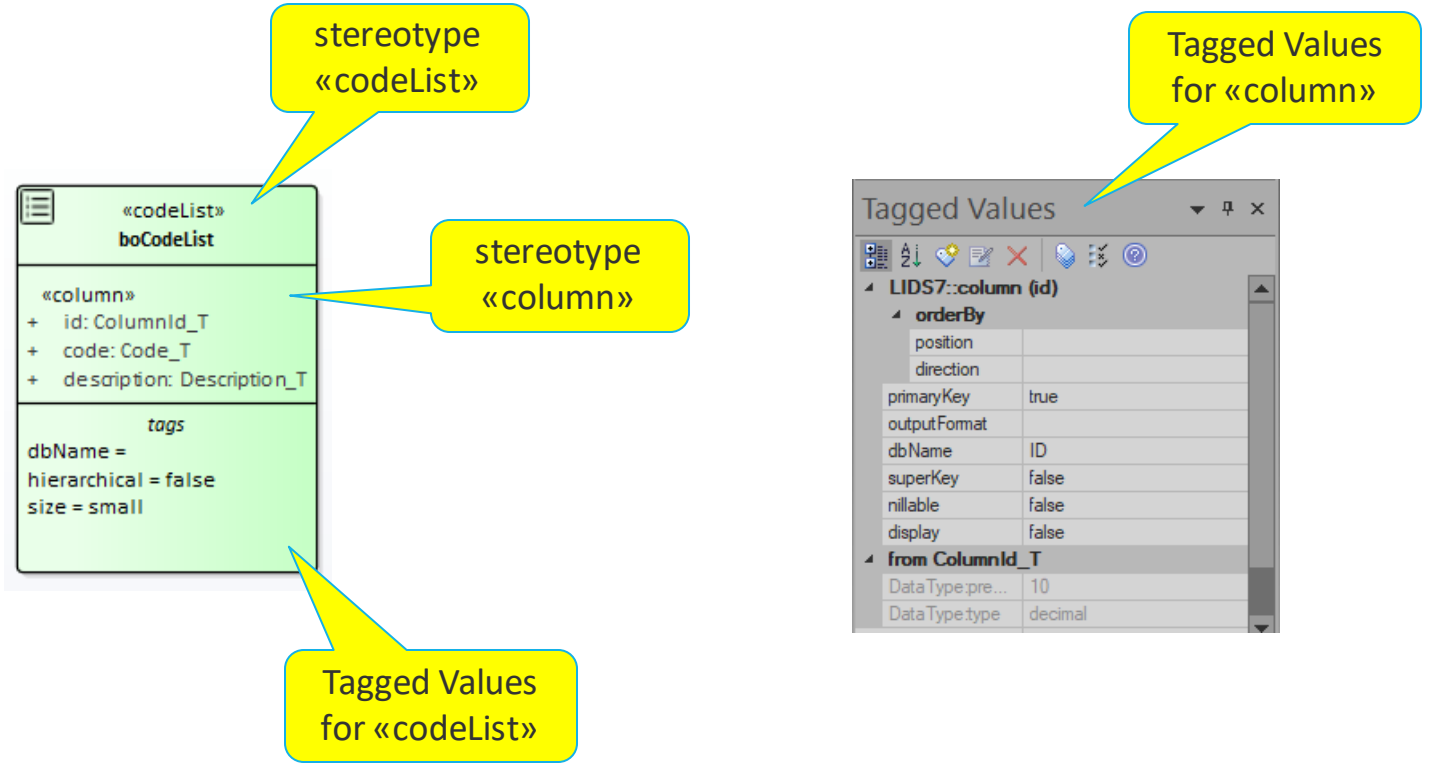


LIDS7 MDG Technology

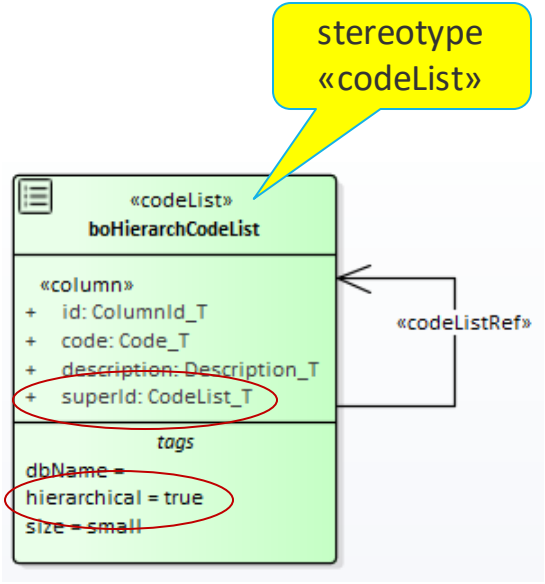
- New specific **<<stereotypes>>** with specific set of **Tagged values**
  - → Class: featureType, codeList, …
  - → Attribute: column, derivedAttribute, featureRefAttribute, …
  - → Association: featureRefAssoc, codeListRef, …
- New **diagram** with specific **Toolbox**
- Set of specific **Data Types**
- **Patterns**
- + **Name Conventions** (doc)
- + Specific transformation rules

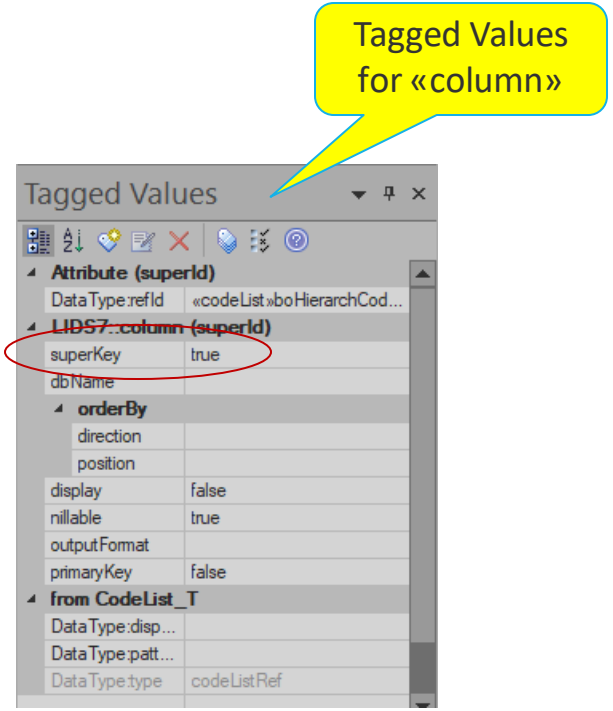
How to model Codelists?

Codelist

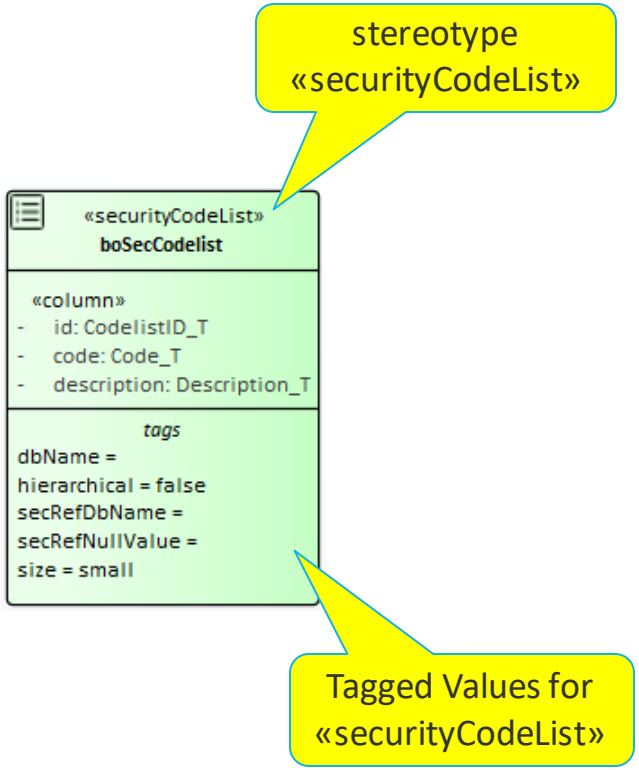

stereotype
«codeList»
«codeList»
boCodeList
«column»
+ id: ColumnId_T
+ code: Code_T
+ description: Description_T
tags
dbName =
hierarchical = false
size = small
stereotype
«column»
Tagged Values
for «codeList»
Tagged Values
for «column»
Tagged Values
LIDS7::column (id)
orderBy
position
direction
primaryKey true
outputFormat
dbName ID
superKey false
nillable false
display false
from ColumnId_T
DataType:pre... 10
DataType:type decimal

Hierarchical Codelist

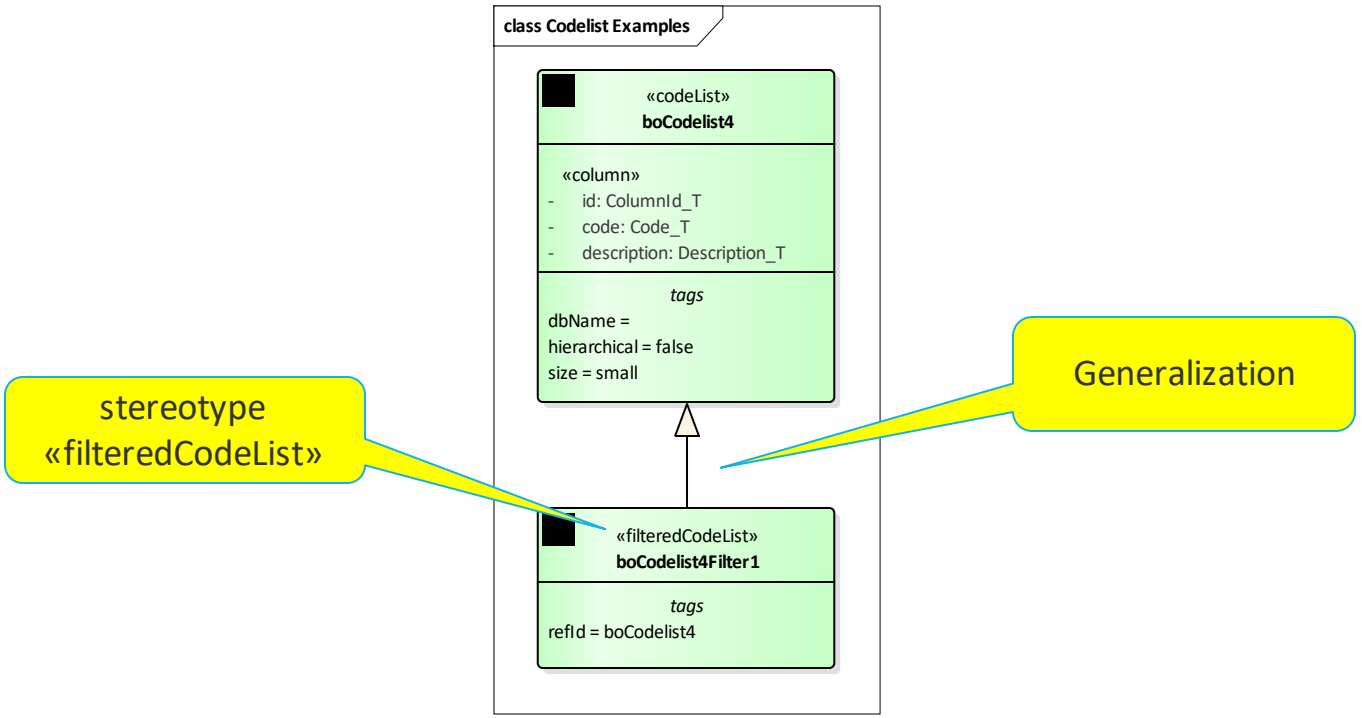




Security Codelist

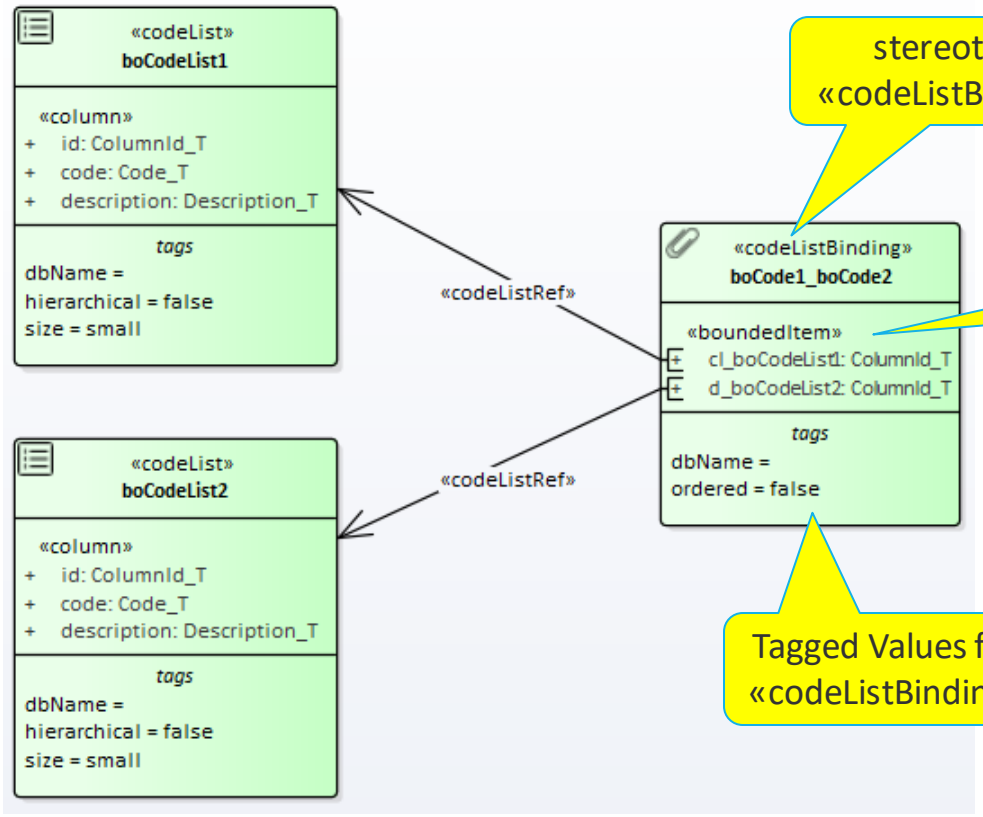


Filtered Codelist



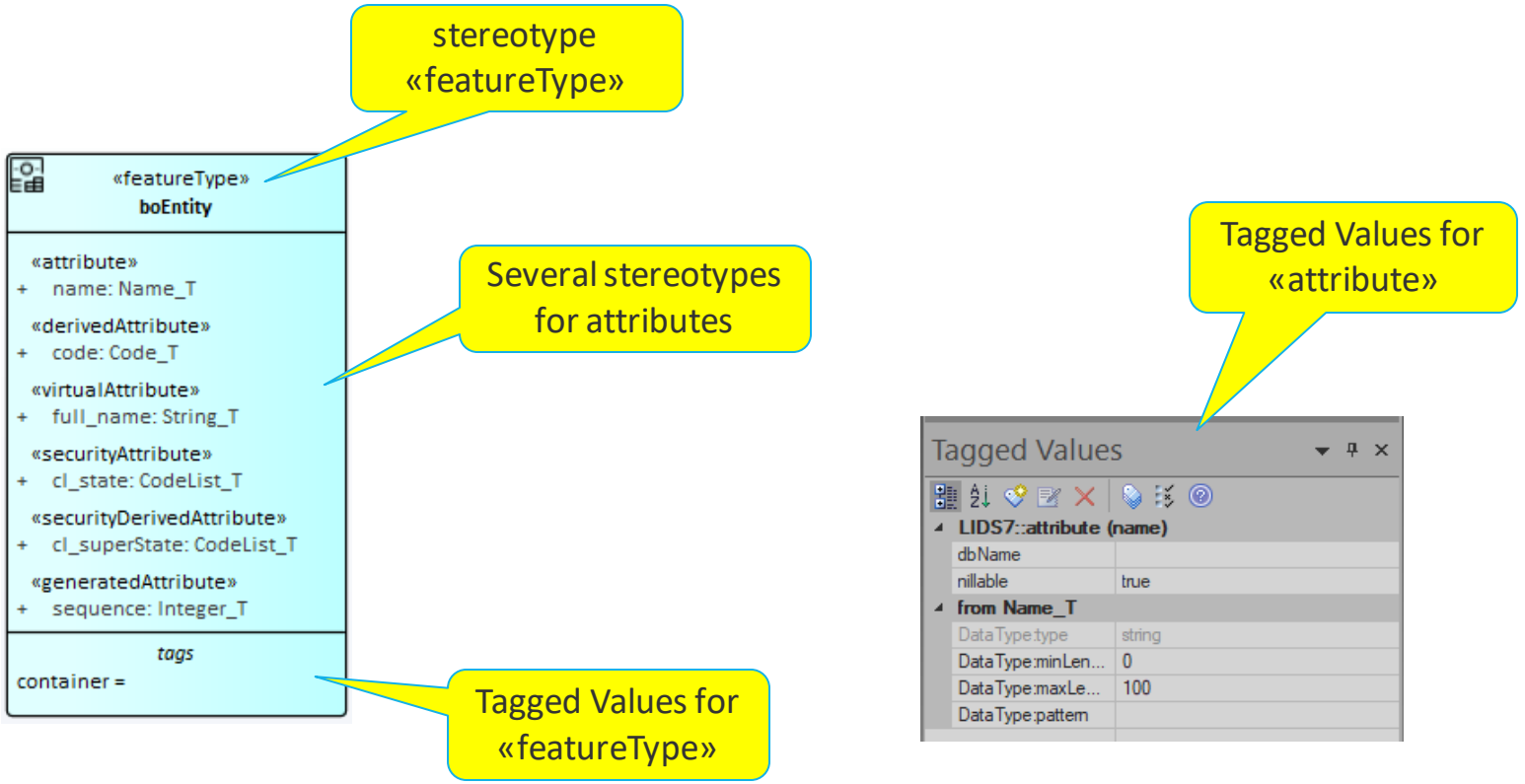
Codelist Binding

- Usage of binding in Feature Type must be done manually („boundedAttributesArray“)

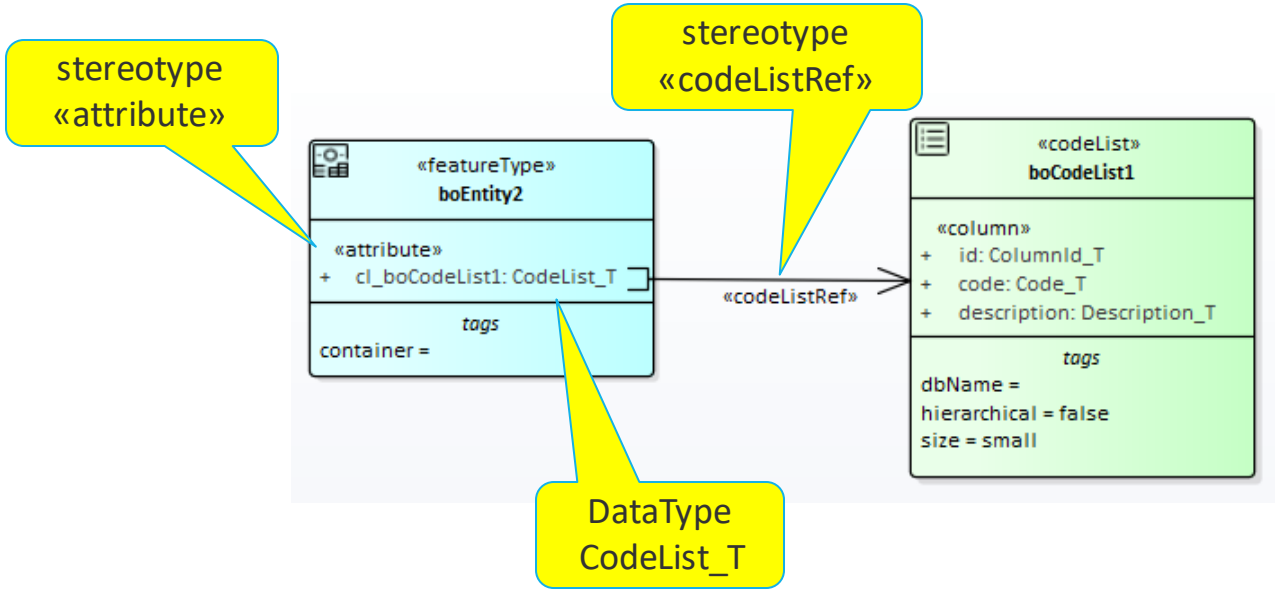


How to model Feature Types?

Feature Type



Feature Type and relation to Codelist

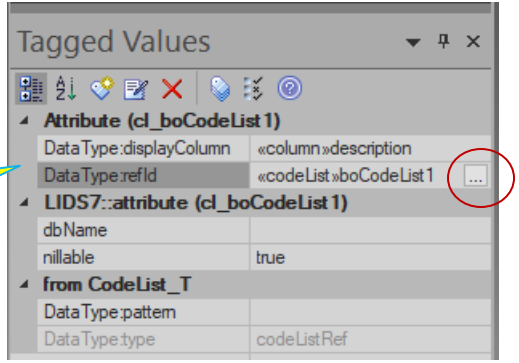


Reference to the codelist is stored in tag of attribute

=> association in model is created just for better understanding (!)

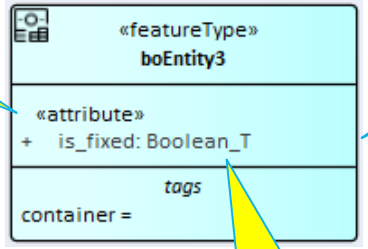
Tagged Values for DataType CodeList_T:

- refId
- displayColumn



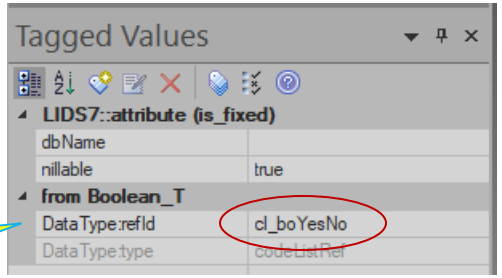
Feature Type and Boolean attribute

stereotype «attribute»



No association in data model!

DataType Boolean_T



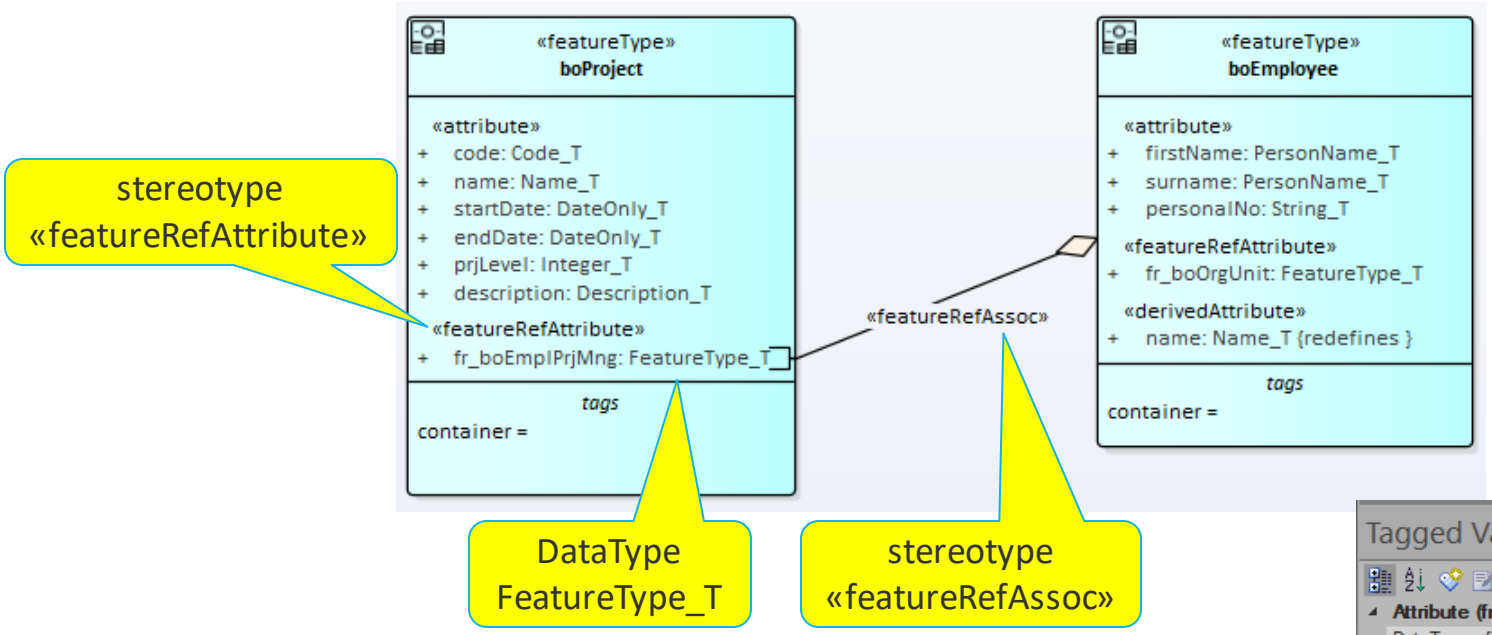
Tagged Values for DataType Boolean_T:

- refId

Reference to „boolean“ codelist is stored in tag of attribute

=> association in model is not necessary and it is not recommended due to simplicity of model

FeatureRef relation – variant Aggregation

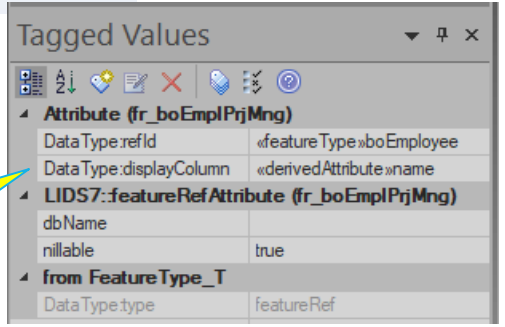


Reference to Feature Type is stored in tag of attribute

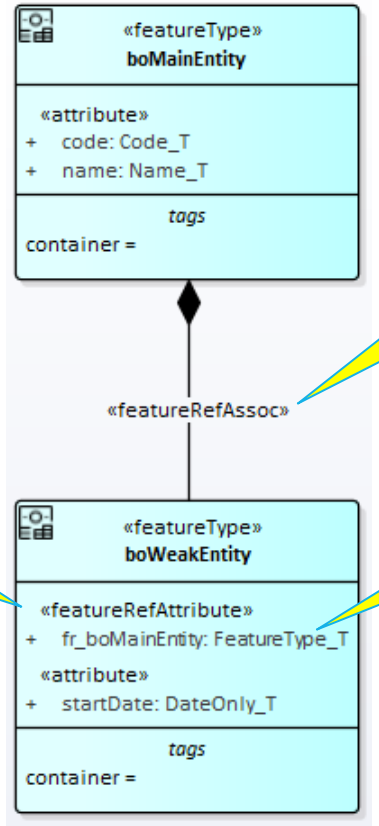
=> association in model is only for better understanding

Tagged Values for DataType CodeList_T:

- refId
- displayColumn



FeatureRef relation – variant Composition



stereotype «featureRefAssoc»

stereotype «featureRefAttribute»

DataType FeatureType_T

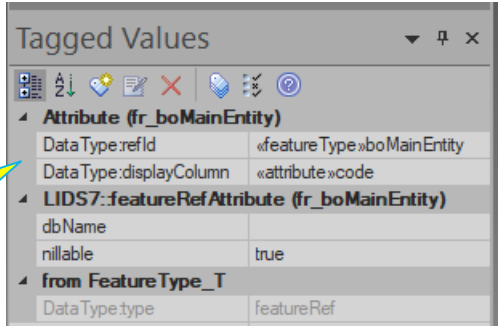
Reference to Feature Type is stored in tag of attribute

=> association in model is only for better understanding

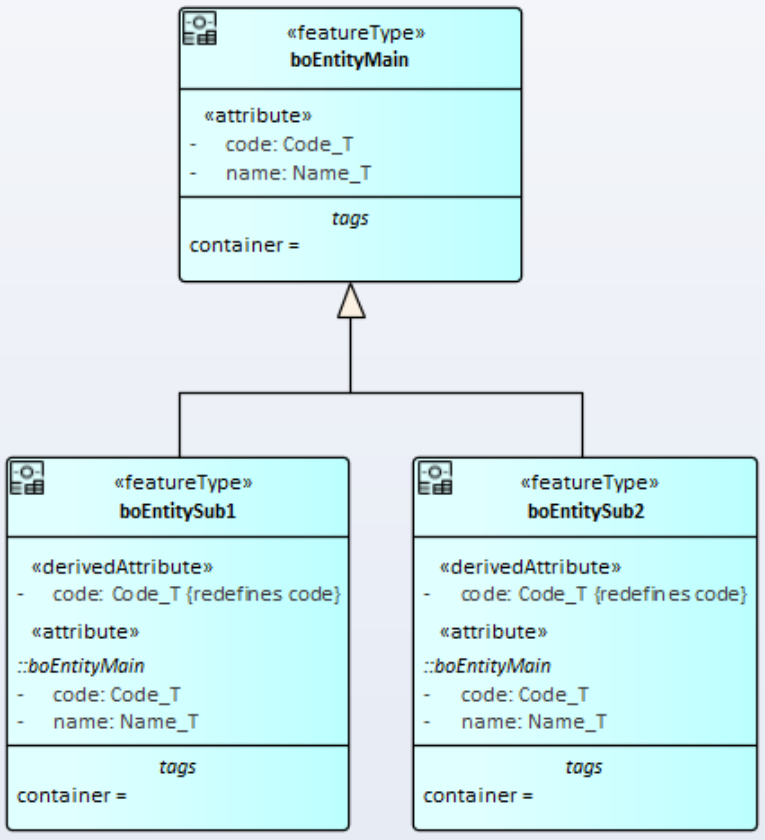
Composite and Aggregation is used for better understanding. Now, there is no impact on generating model.xml.

Tagged Values for DataType CodeList_T:

- refId
- displayColumn

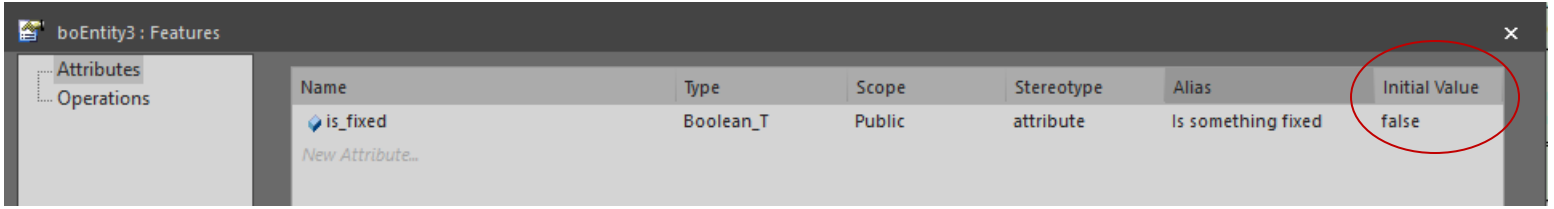


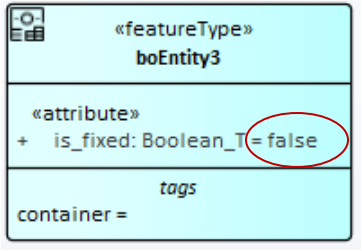
Feature Type and Generalization



- Standard UML modelling
- „root“ ft_5000002 is not necessary to model in EA
- Generalization is a big challenge for modelling in EA

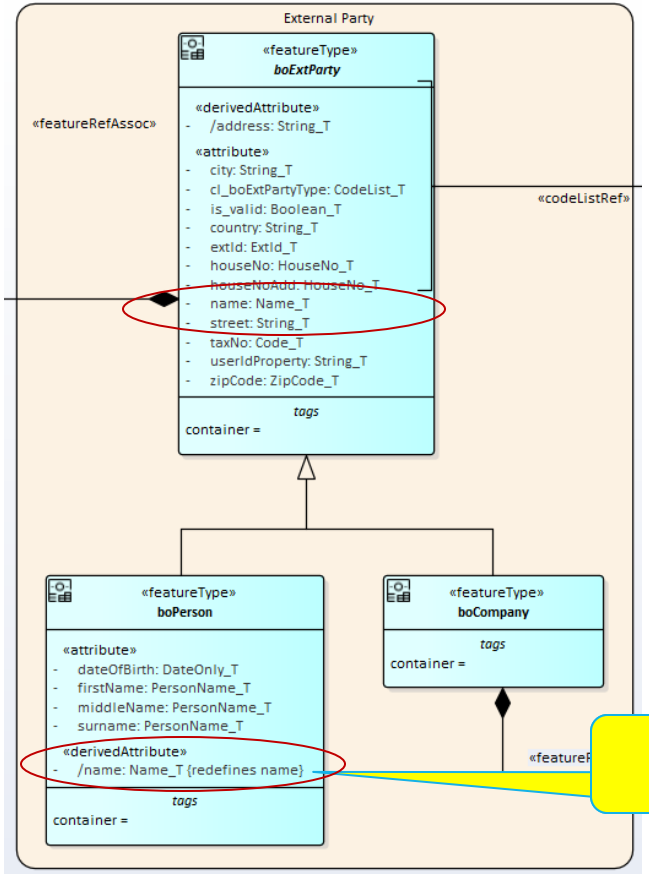
Default value of attribute





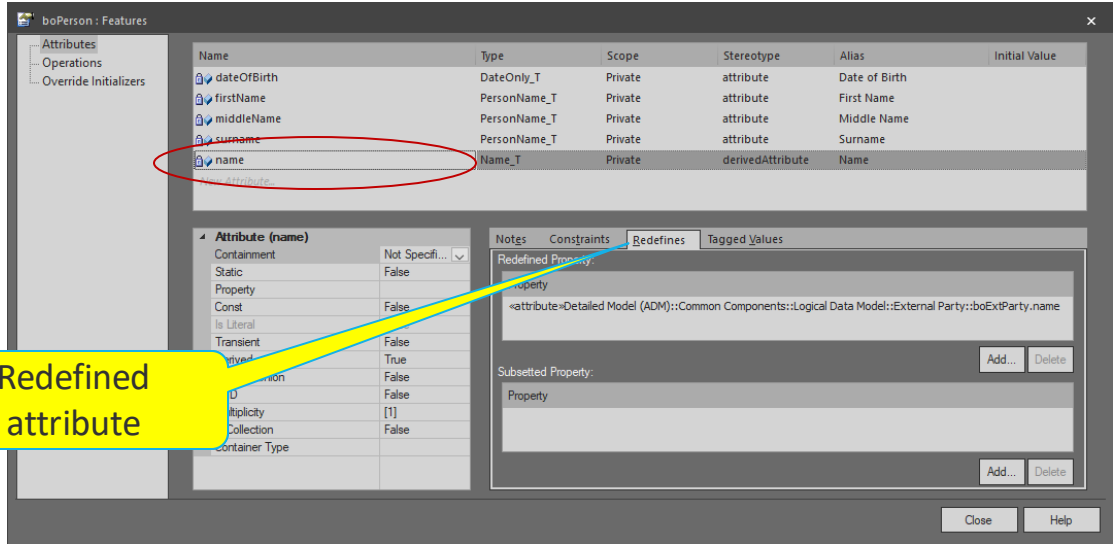
For default value in model.xml is used item Initial Value in EA

Redefinition of attributes

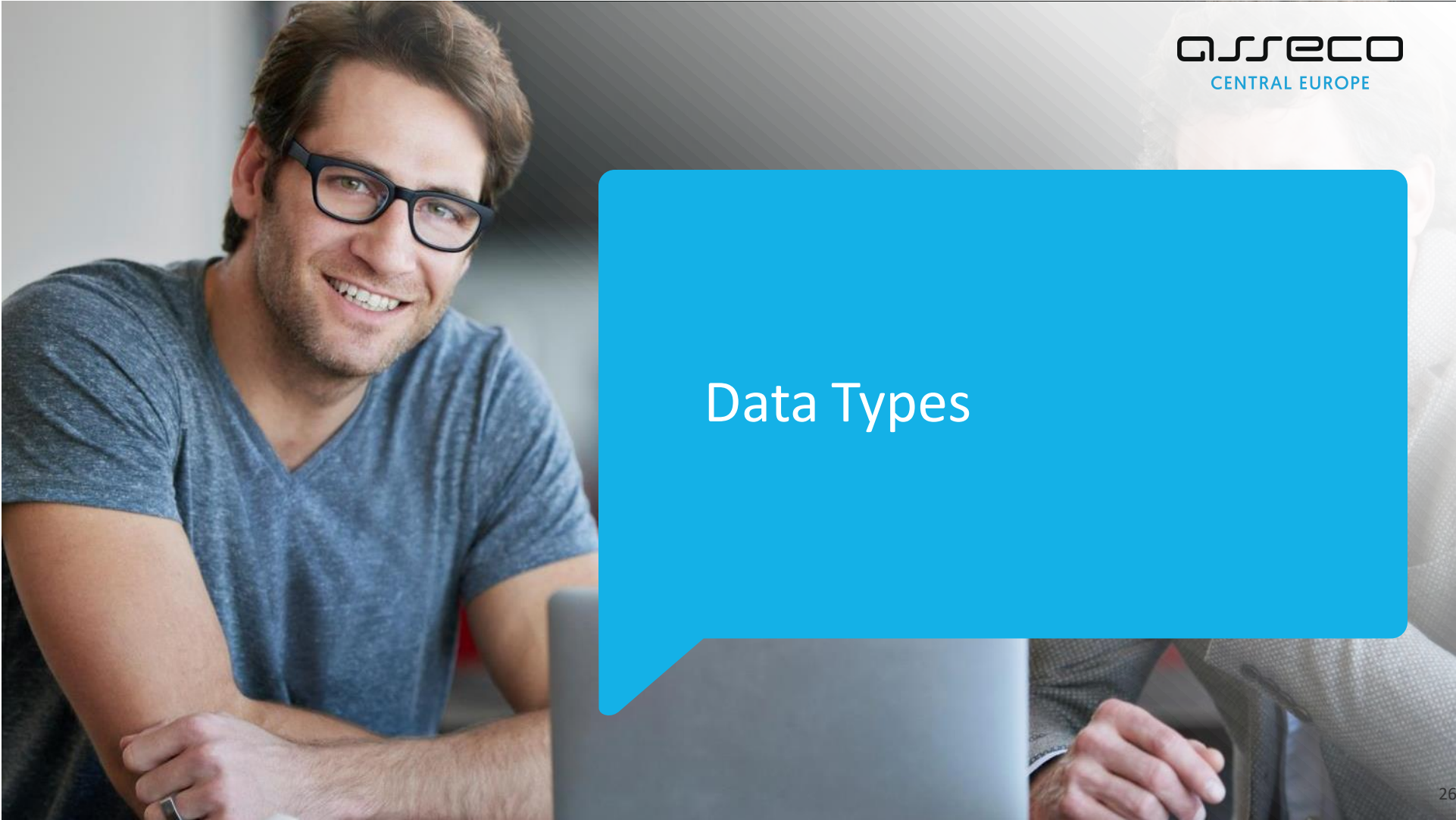


What you can redefine:

- Default values.
- Whether the attribute is mandatory (nillable flag in model).
- Forbidden operations (READ, QUERY, UPDATE).
- Data type (you can change a codelist data type to a different codelist, different codelist must be a filtered codelist based on previous codelist).
- Derived attribute query (you can define deriving query in children feature types, you can change deriving query in children feature types).
- You can change the type of the attribute from normal (parent) to derived in any child.
- You can make multiple redefinition of one attribute.



Redefined attribute



asseco
CENTRAL EUROPE

Data Types

Data Types

- Mandatory
  - → Several universal data types – string, decimal, …
  - → A lot of specific data types with the specific default values of tags

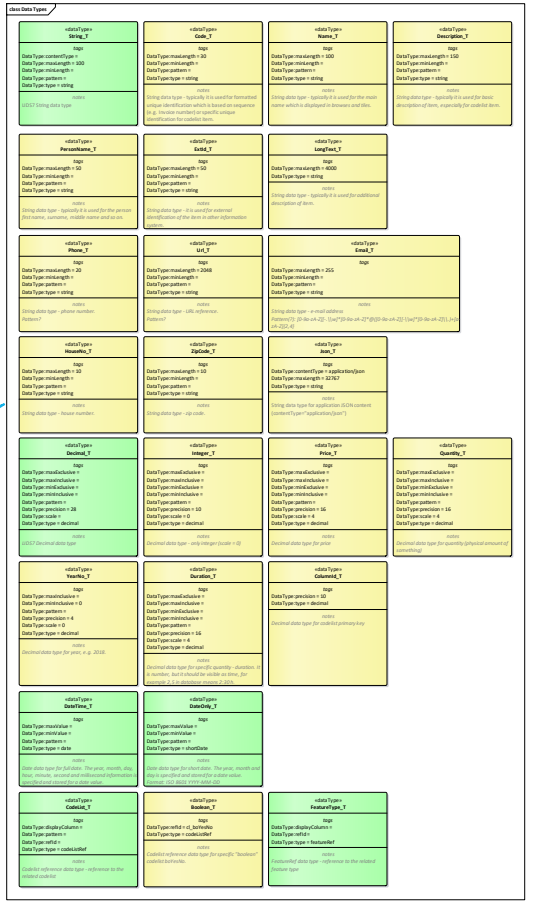
Example for DataType Code_T

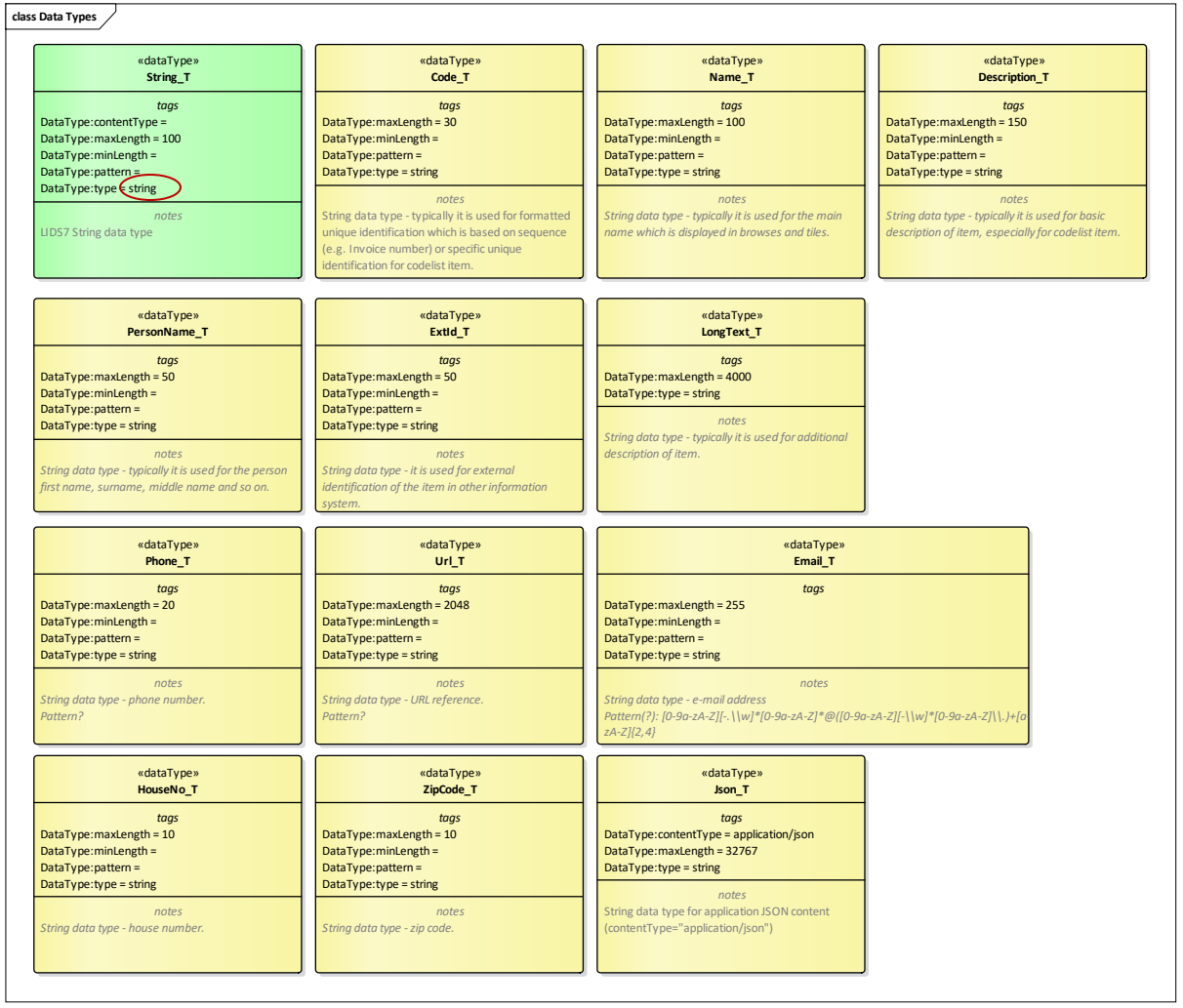
«dataType»
Code_T

tags
DataType:maxLength = 30
DataType:minLength =
DataType:pattern =
DataType:type = string

notes
String data type - typically it is used for formatted unique identification which is based on sequence (e.g. Invoice number) or specific unique identification for codelist item.

25 dataTypes

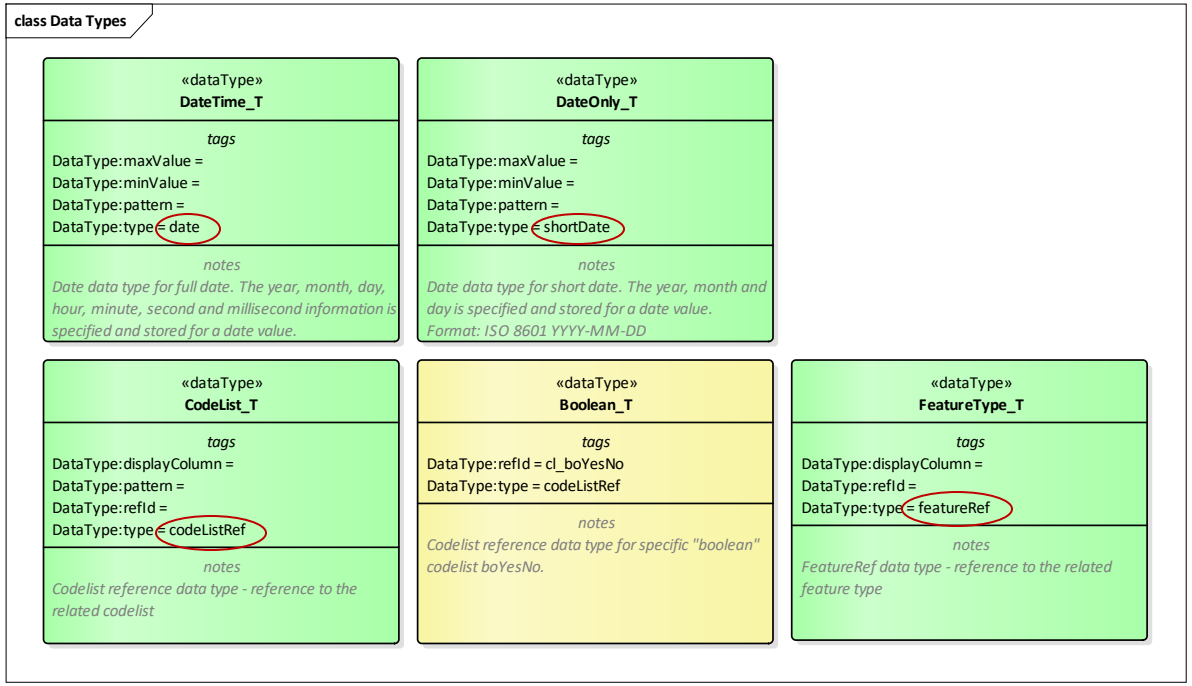




Data Types based on ‚string‘

Data Types based on ‚decimal‘

Other Data Types



asseco
CENTRAL EUROPE

Naming Convention for LIDS model

Naming Conventions (example I)

featureType	**ft_**	*Id*	**ft_{dom}FeatureTypeName** *Note: It is recommended to make the identifier at most **15 (max 20)** characters long (including prefix).*	ft_boDefect ft_eleCable
		DB	*Not required (Feature Type is represented in database via the assigned container DB name).*	
featureAttribute	**at_**	*Id*	**at_{dom}FeatureTypeName_attributName**	at_boDefect_code at_eleCable_description
		DB	**attributName**	CODE DESCRIPTION
container	**ct_**	*Id*	**ct_{dom}ContainerName** If the container is intended just for one feature type, then the name should be the same as for the feature type. If the container is for more than one feature types that are connected via relation Generalization, then the name should be derived from the "highest" feature type in hierarchy.	ct_boDefect ct_eleCable
		DB	**CT_{dom}ContainerName**	CT_BODEFECT CT_ELECABLE

Keep it as short as possible

One underscore!

No underscores in DB!

Naming Conventions (example II)

featureAttribute variant: Codelist reference	*Id*	**at_{dom}FeatureTypeName_cl_{dom}CodeListName{diff}** Where optional **{diff}** is used in han one reference to the same code necessary, the **CodelistName** may be truncated.	at_boDefect_cl_boDefectType
	DB	**cl_{dom}CodelistName{diff}**	CL_BODEFECTTYPE
featureAttribute variant: Feature reference	*Id*	**at_{dom}FeatureTypeName_fr_{dom}FeatureTypeNameRef{diff}** Where **FeatureTypeNameRef** is eature type being reffered to. Optional suffix **{diff}** is used in case when more than one reference to the same feature is needed. If necessary, the FeatureTypeNameRef may be truncated. Especially in case of a self-reference, the FeatureTypeNameRef may be omitted and for suffix **{diff}** is recommended "superId" or "sup" (if a shortened version is required). *Note: Identifier may be at most **30** characters long (including prefix).*	at_boDefect_fr_boWorkOrder at_boOrgUnit_fr_sup
	DB	**fr_{dom}FeatureTypeNameRef{diff}**	FR_BOWORKORDER

cl_ prefix

fr_ prefix

No SID!

is_ prefix

Name Conventions (example III)

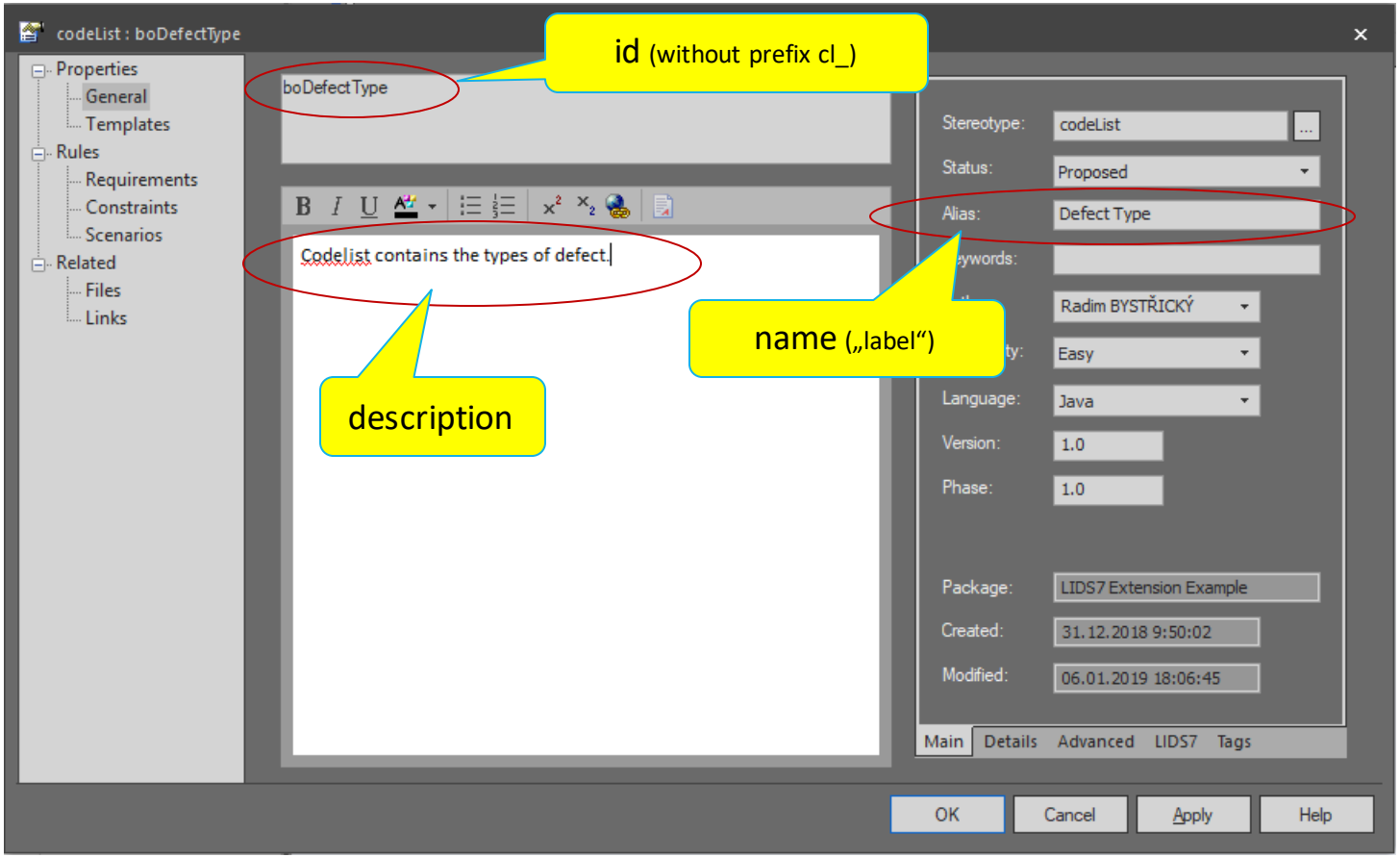
is_ prefix

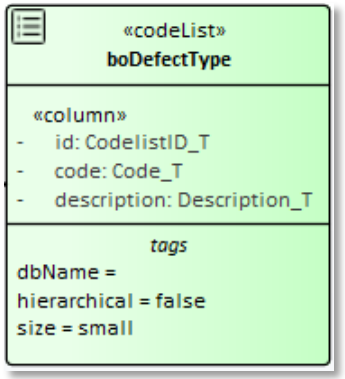
featureAttribute variant: Boolean	*Id*	**at_{dom}FeatureTypeName_is_Name** The similar convention as for the codelist reference is	at_boDefect_is_fixed
		used here because the Boolean value is modeled as reference to the specific codelist with two values Yes/No.	
	DB	**is_Name**	IS_FIXED

- Description of all naming conventions is available as a documentation of SAMO implementation.
- Naming conventions can be set by each project

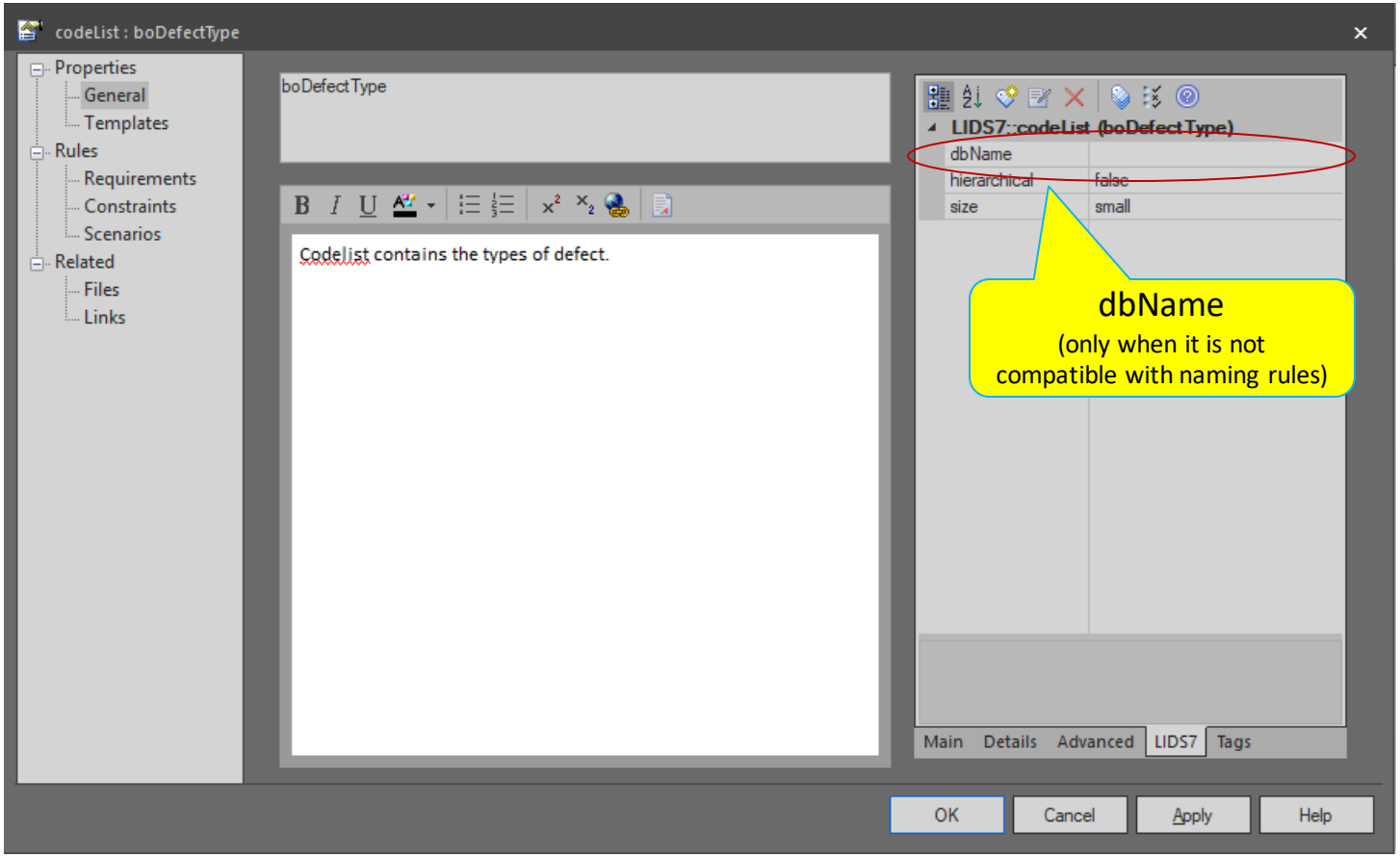
Basic transformation rules

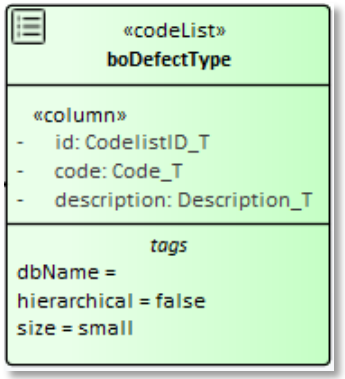
Codelist


codeList : boDefectType
Properties
General
Templates
Rules
Requirements
Constraints
Scenarios
Related
Files
Links
boDefectType
id (without prefix cl_)
Codelist contains the types of defect.
description
name („label“)
Stereotype: codeList
Status: Proposed
Alias: Defect Type
Radim BYSTŘICKÝ
Easy
Language: Java
Version: 1.0
Phase: 1.0
Package: LIDS7 Extension Example
Created: 31.12.2018 9:50:02
Modified: 06.01.2019 18:06:45
Main
Details
Advanced
LIDS7
Tags
OK
Cancel
Apply
Help

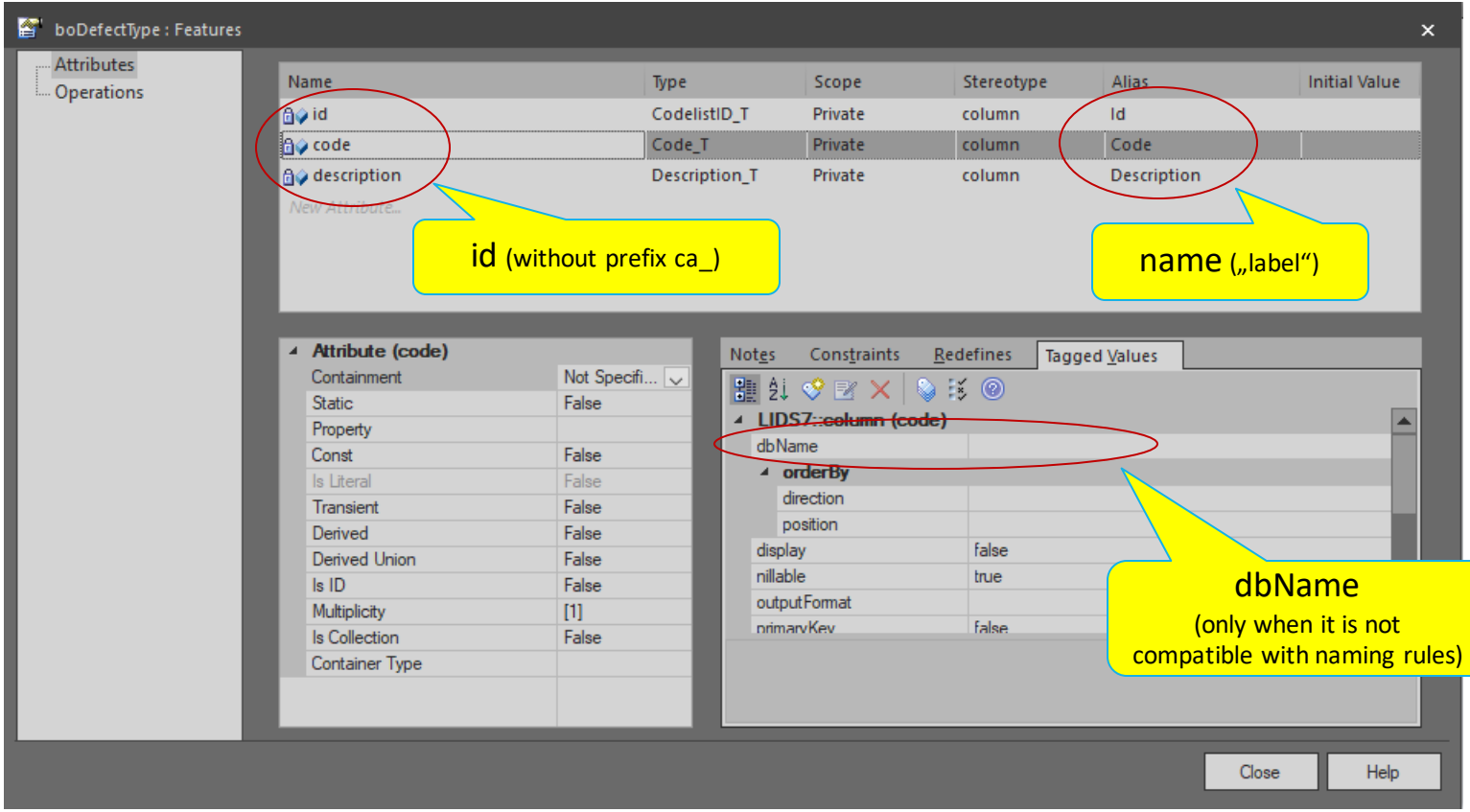

«codeList»
boDefectType
«column»
- id: CodelistID_T
- code: Code_T
- description: Description_T
tags
dbName =
hierarchical = false
size = small

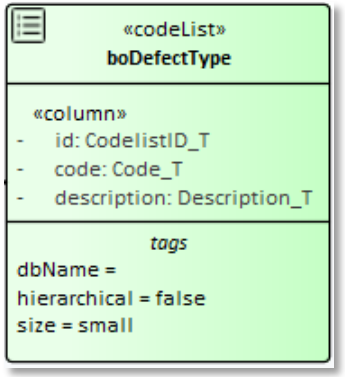
Codelist (II)


codeList : boDefectType
Properties
General
Templates
Rules
Requirements
Constraints
Scenarios
Related
Files
Links
boDefectType
Codelist contains the types of defect.
LIDS7::codeList (boDefectType)
dbName
hierarchical
false
size
small
dbName
(only when it is not compatible with naming rules)
Main
Details
Advanced
LIDS7
Tags
OK
Cancel
Apply
Help

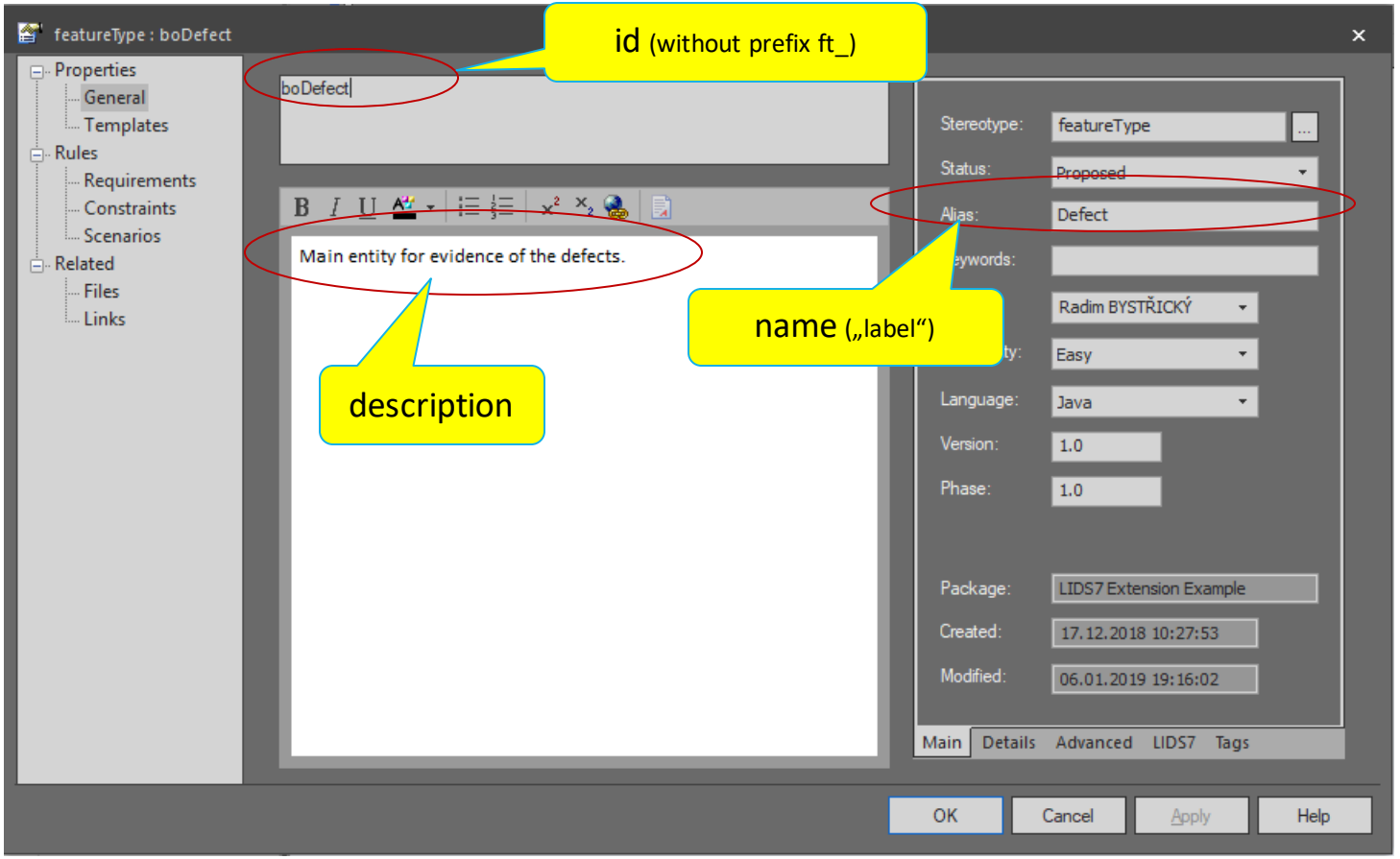

«codeList»
boDefectType
«column»
- id: CodelistID_T
- code: Code_T
- description: Description_T
tags
dbName =
hierarchical = false
size = small

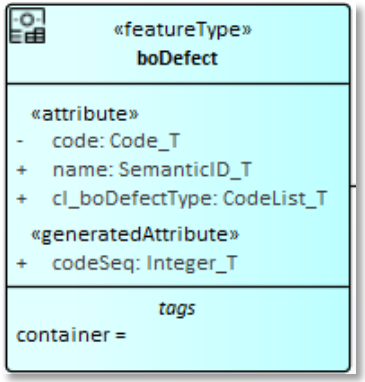
Codelist Column



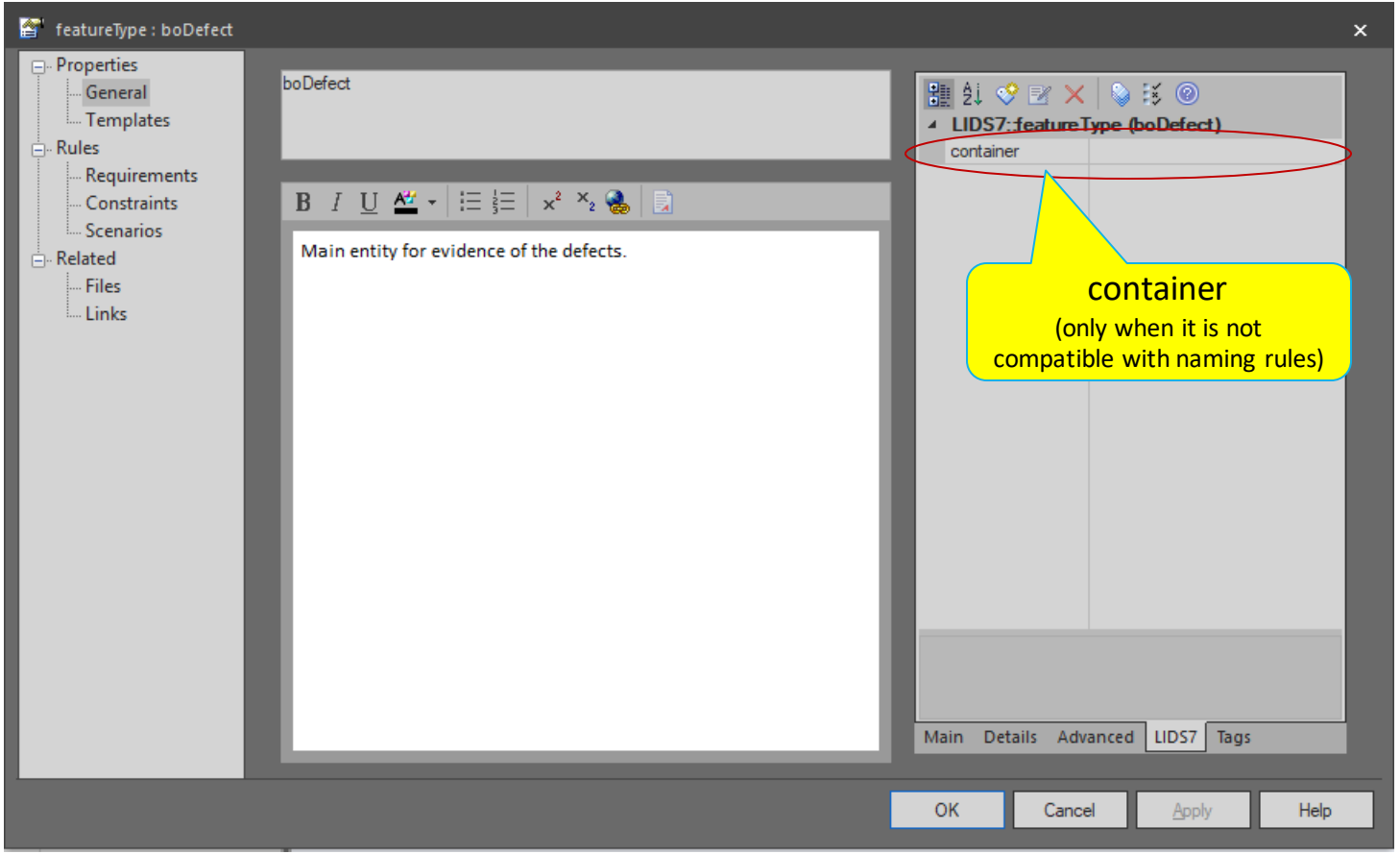


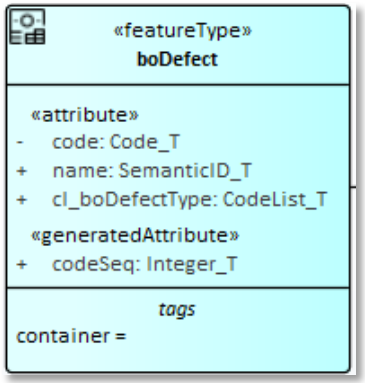
Feature Type



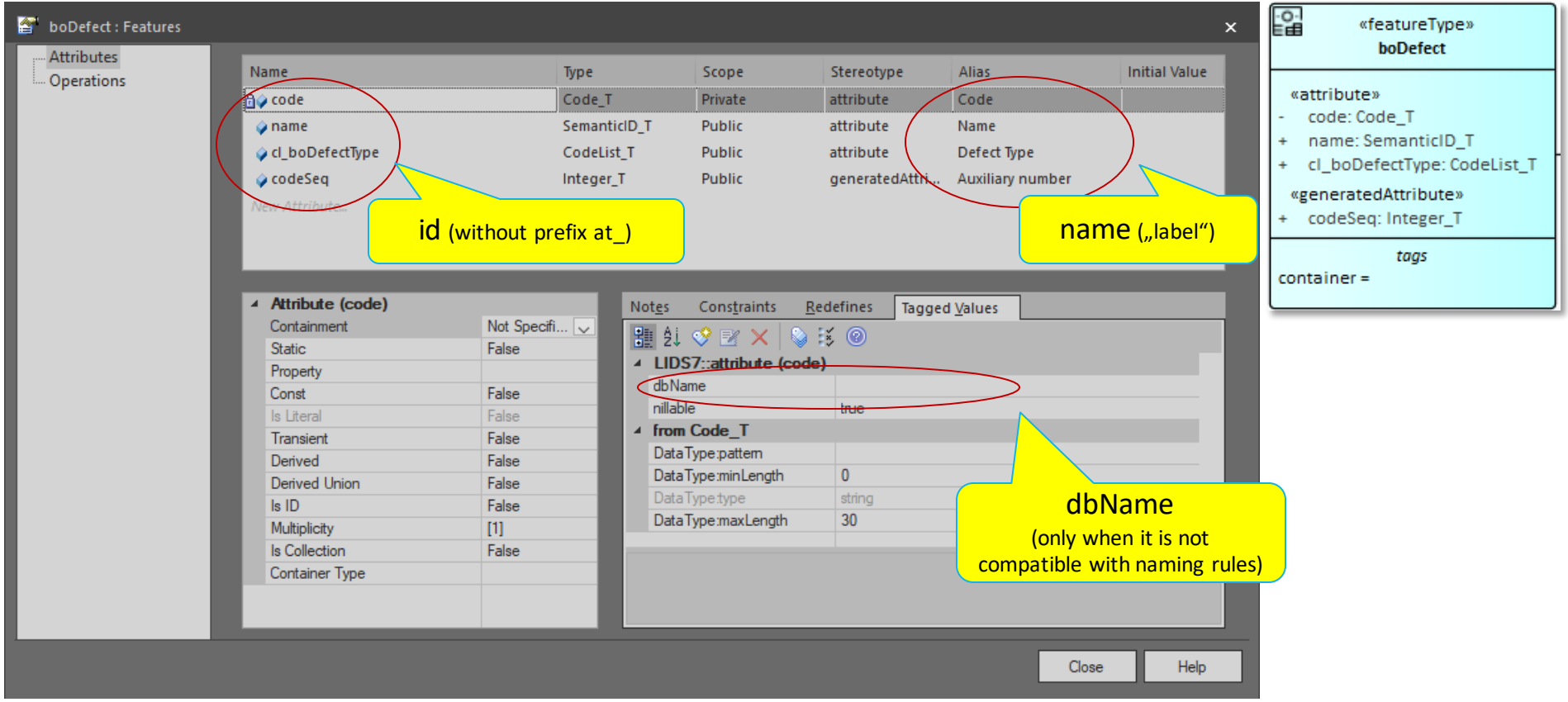


Feature Type (II)

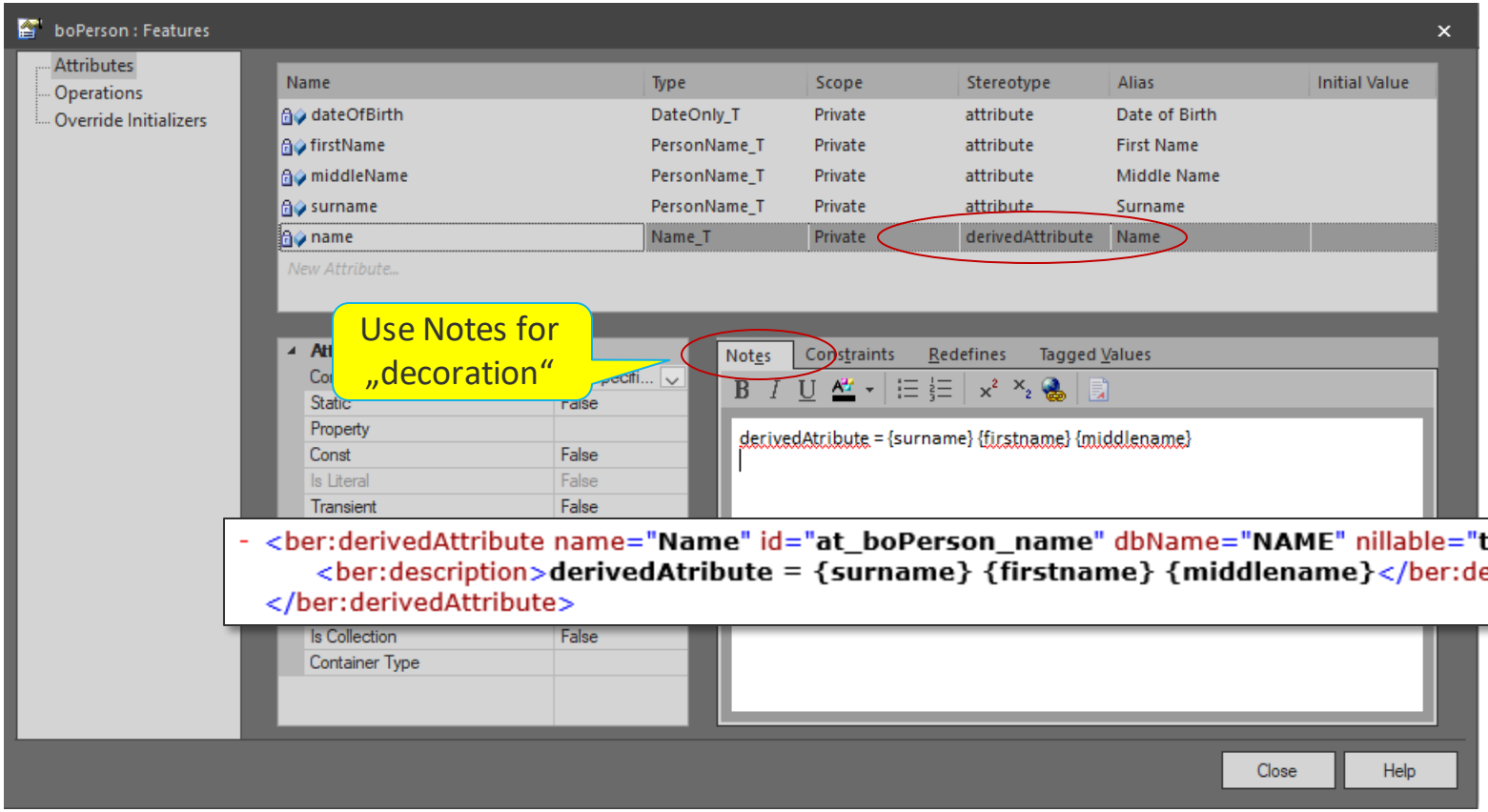


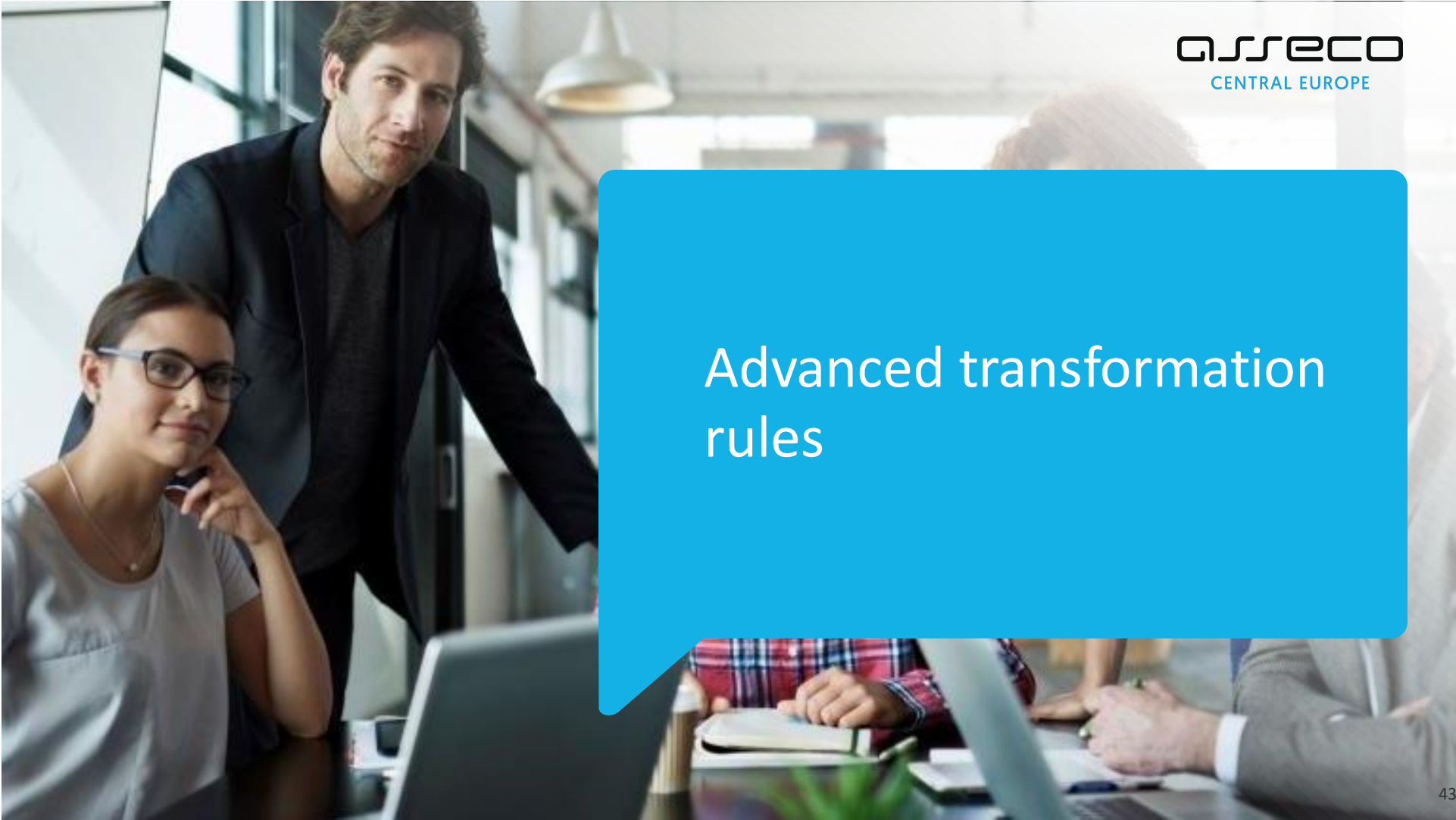


Feature Attribute



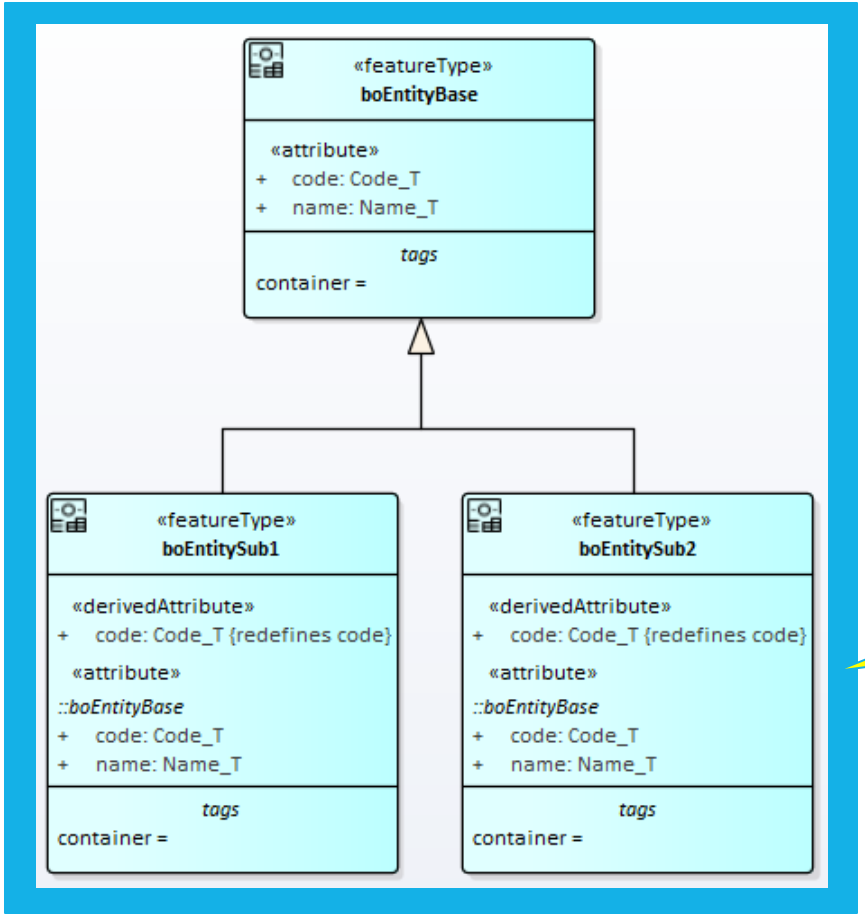
Description of attribute





Advanced transformation rules

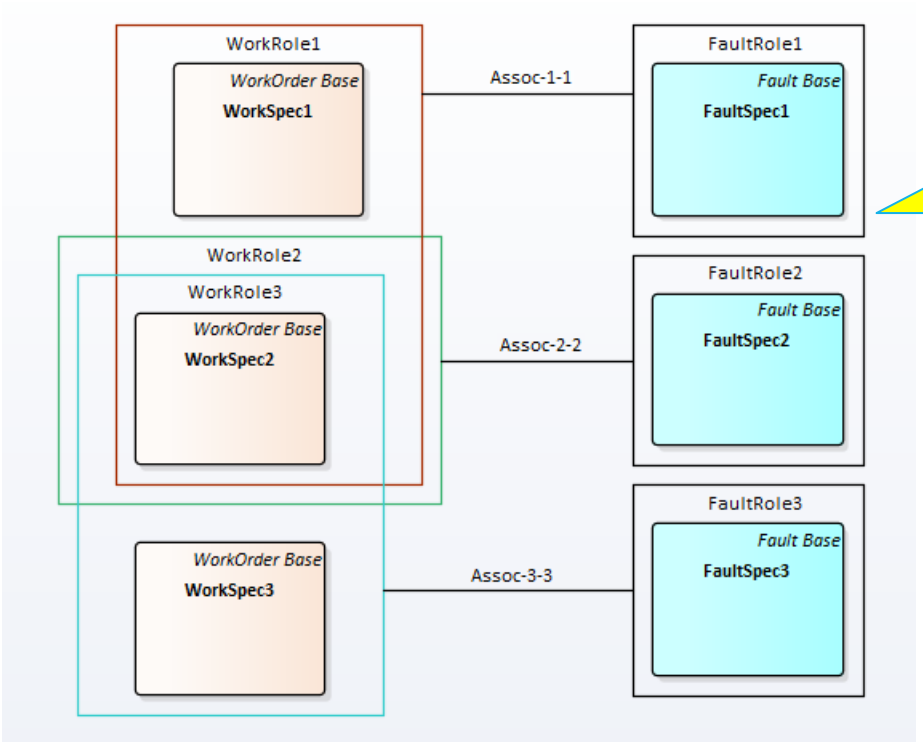
Containers



- If the feature type is not a part of the „inheritance tree“ then the container is generated as 1:1 with this feature type
- If the feature type is a part of the „inheritance tree“ then the container is generated for the „root“ feature type and all other feature types are included in this container
- It is possible to use the tagged value ‚container‘ to define a specific container for the feature type

The container ct_boEntityBase contains all feature types that are related via the generalization

Roles



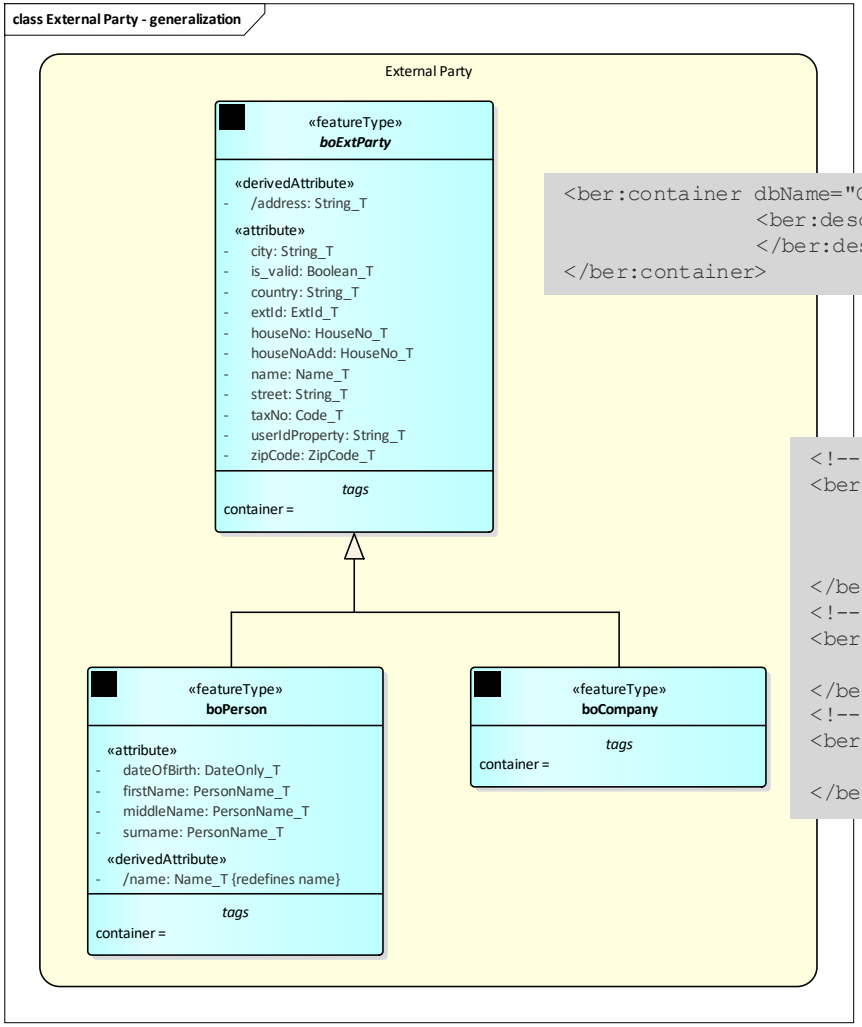
In standard modeling you need several associations and roles for the same „business“ relation „WorkOrder for fixing Fault“ (in the case there exists an association between the complex „inheritance“ structures).

Attention

- Redesign of roles in the model can lead to a loss of information about previous relations between objects!

Recommendations for now:

- Design one role with all feature types in the „inheritance“ tree
- Allow all-to-all combination
- Refactoring later – LIDS should allow to set the combinations in special extension



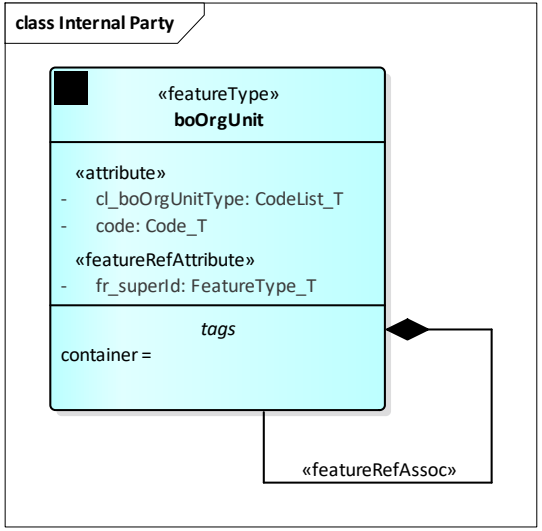
One container

```
<ber:container dbName="CT_BOEXTPARTY" id="ct_boExtParty" name="External Party" versioned="true">
               <ber:description>Table containing the information about given external party.
               </ber:description>
</ber:container>
```

```
<!-- Role :: BOEXTPARTY -->
<ber:relationRole dbName="BOEXTPARTY" id="rt_boExtParty" name="External Party">
               <ber:ftItem refId="ft_boExtParty"/>
               <ber:ftItem refId="ft_boCompany"/>
               <ber:ftItem refId="ft_boPerson"/>
</ber:relationRole>
<!-- Role :: BOCOMPANY -->
<ber:relationRole dbName="BOCOMPANY" id="rt_boCompany" name="Company">
               <ber:ftItem refId="ft_boCompany"/>
</ber:relationRole>
<!-- Role :: BOPERSON -->
<ber:relationRole dbName="BOPERSON" id="rt_boPerson" name="Person">
               <ber:ftItem refId="ft_boPerson"/>
</ber:relationRole>
```

Three „default“ roles

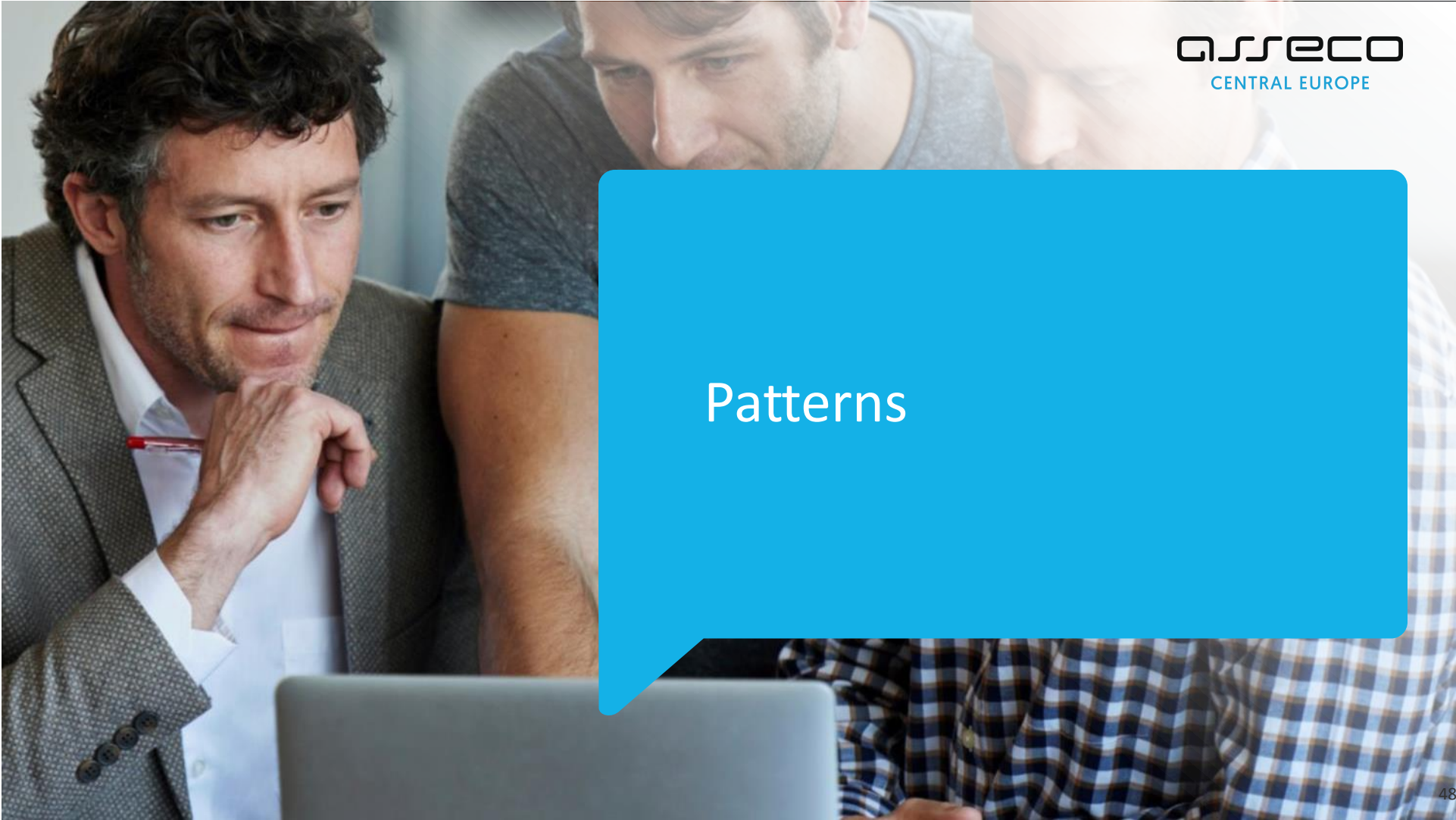
Roles - self-reference



For self-reference association you need two different roles

```
<ber:relationRole dbName="BOORGUNIT" id="rt_boOrgUnit_superior" name="Organizational Unit">
    <ber:ftItem refId="ft_boOrgUnit"/>
</ber:relationRole>
<ber:relationRole dbName="BOORGUNIT" id="rt_boOrgUnit" name="Organizational Unit">
    <ber:ftItem refId="ft_boOrgUnit"/>
</ber:relationRole>
```

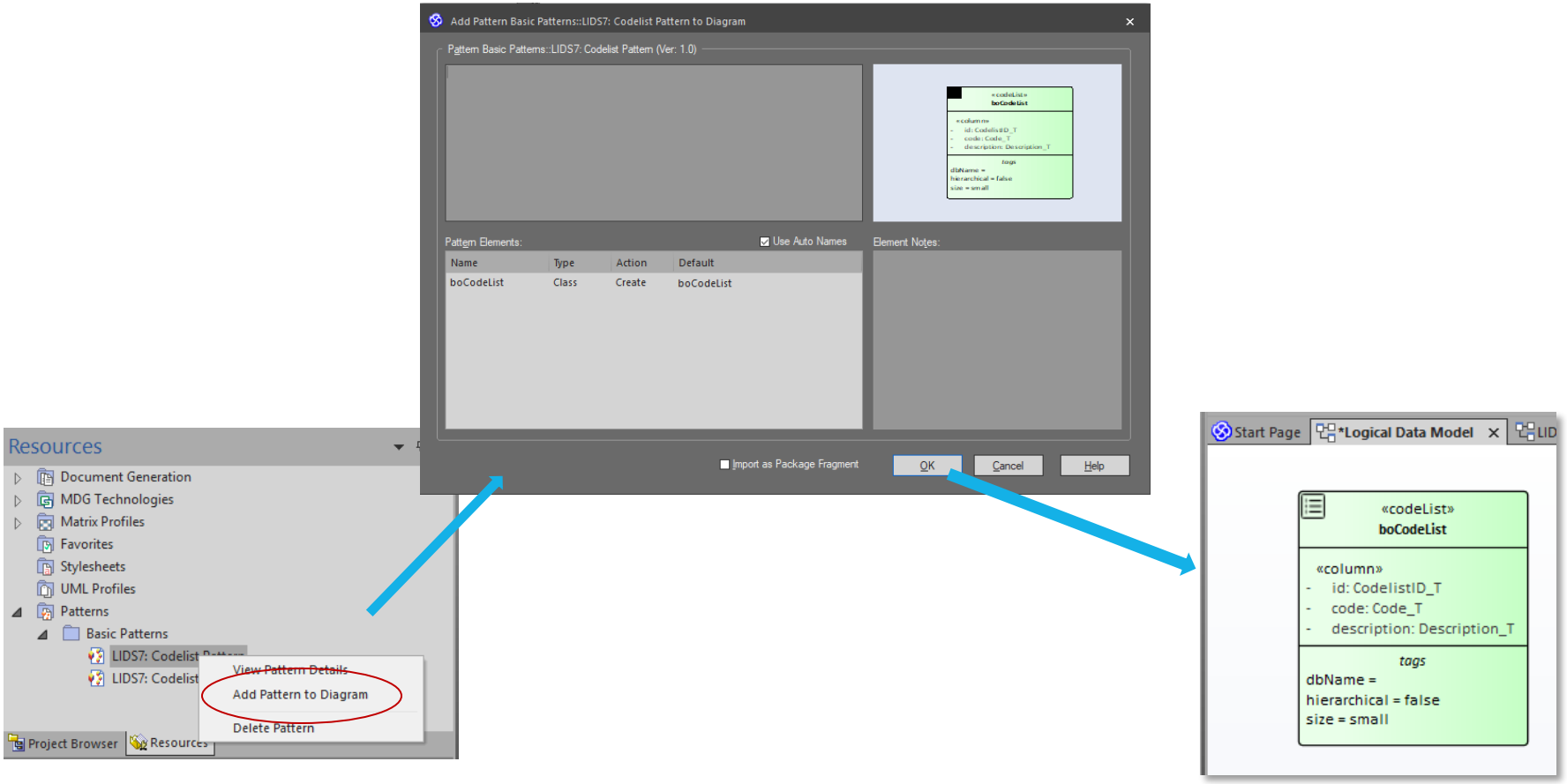
```
<ber:featureRefRelationAssoc id="as_boOrgUnit_boOrgUnit" name="boOrgUnit -&gt; boOrgUnit">
    <ber:masterRole cardinality="1" refId="rt_boOrgUnit_superior"/>
    <ber:childRole cardinality="unbounded" refId="rt_boOrgUnit"/>
</ber:featureRefRelationAssoc>
```



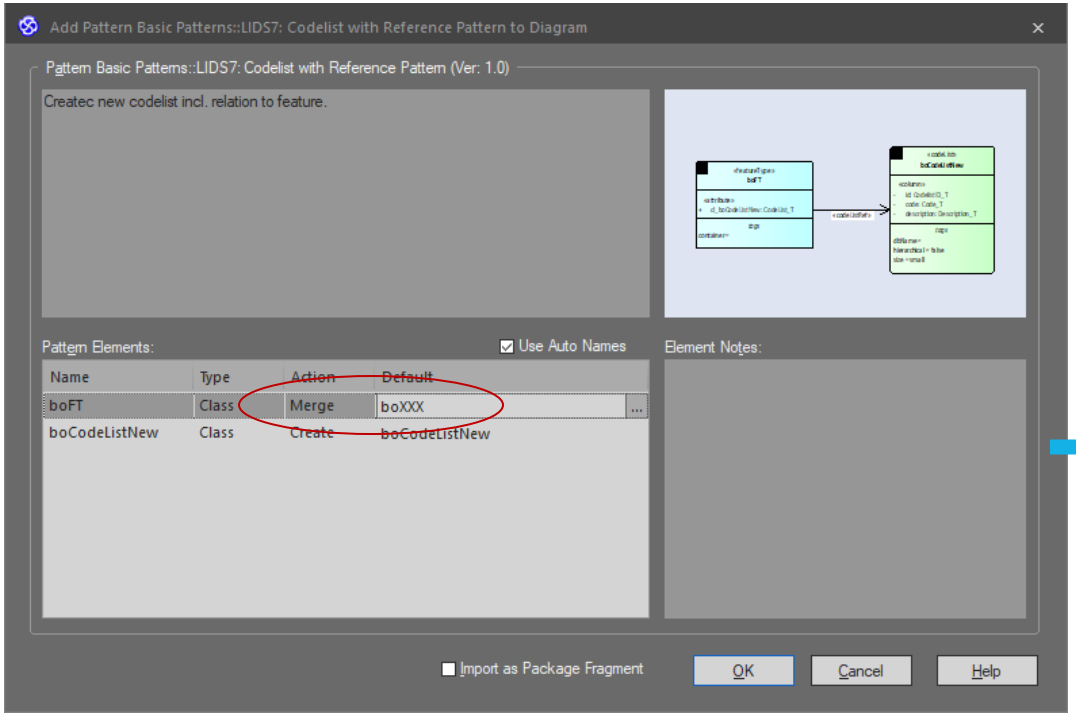
asseco
CENTRAL EUROPE

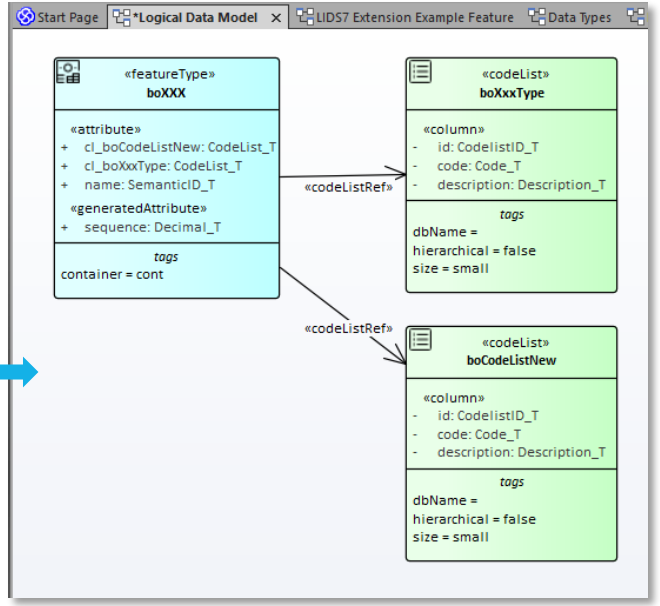
Patterns

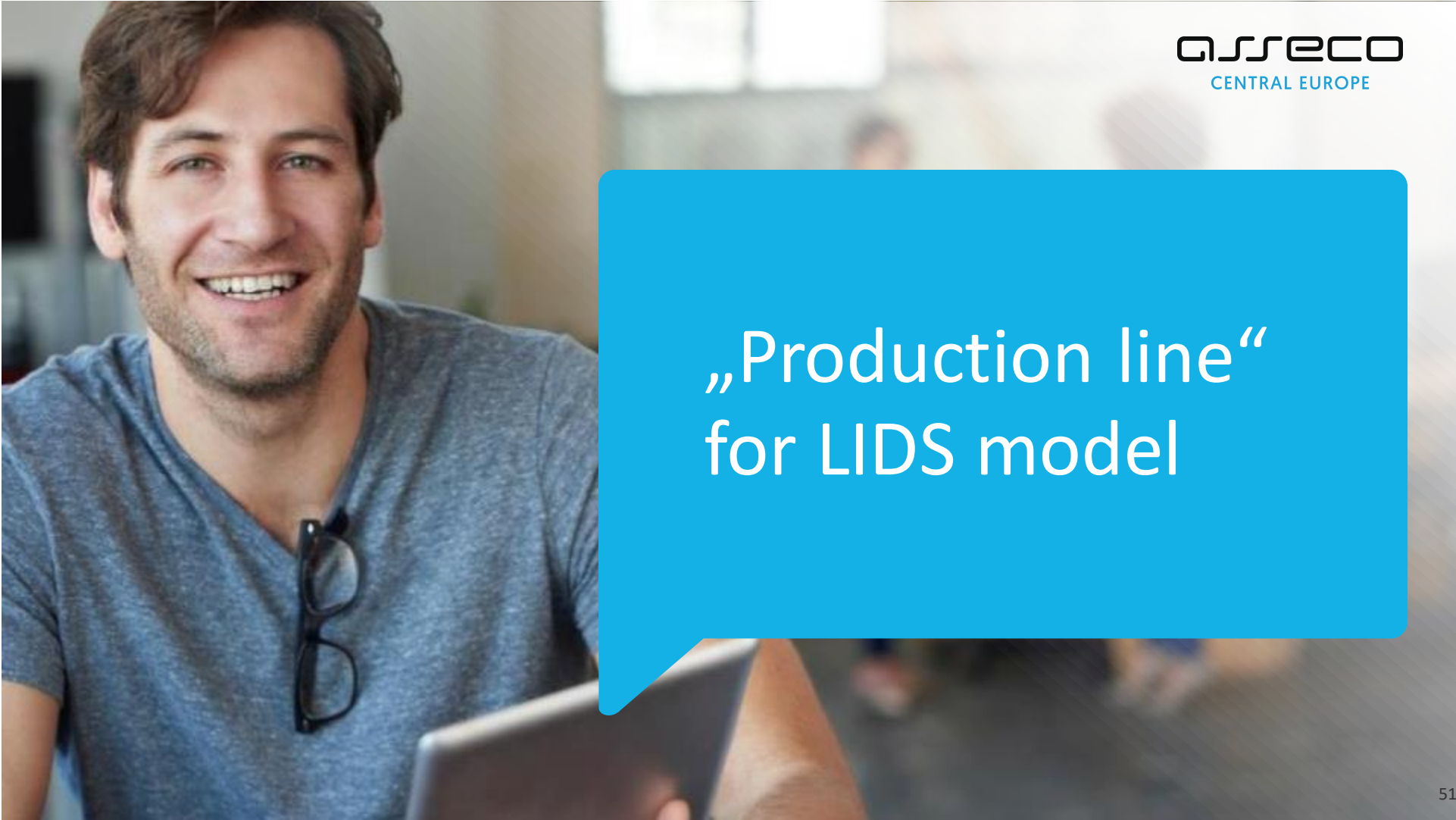
Patterns


Add Pattern Basic Patterns::LIDS7: Codelist Pattern to Diagram
Pattern Basic Patterns::LIDS7: Codelist Pattern (Ver: 1.0)
Pattern Elements:
Use Auto Names
Element Notes:
Name
Type
Action
Default
boCodeList
Class
Create
boCodeList
Import as Package Fragment
OK
Cancel
Help
Resources
Document Generation
MDG Technologies
Matrix Profiles
Favorites
Stylesheets
UML Profiles
Patterns
Basic Patterns
LIDS7: Codelist Pattern
LIDS7: Codelist
View Pattern Details
Add Pattern to Diagram
Delete Pattern
Project Browser
Resources
Start Page
*Logical Data Model
«codeList»
boCodeList
«column»
- id: CodelistID_T
- code: Code_T
- description: Description_T
tags
dbName =
hierarchical = false
size = small

Patterns with relation



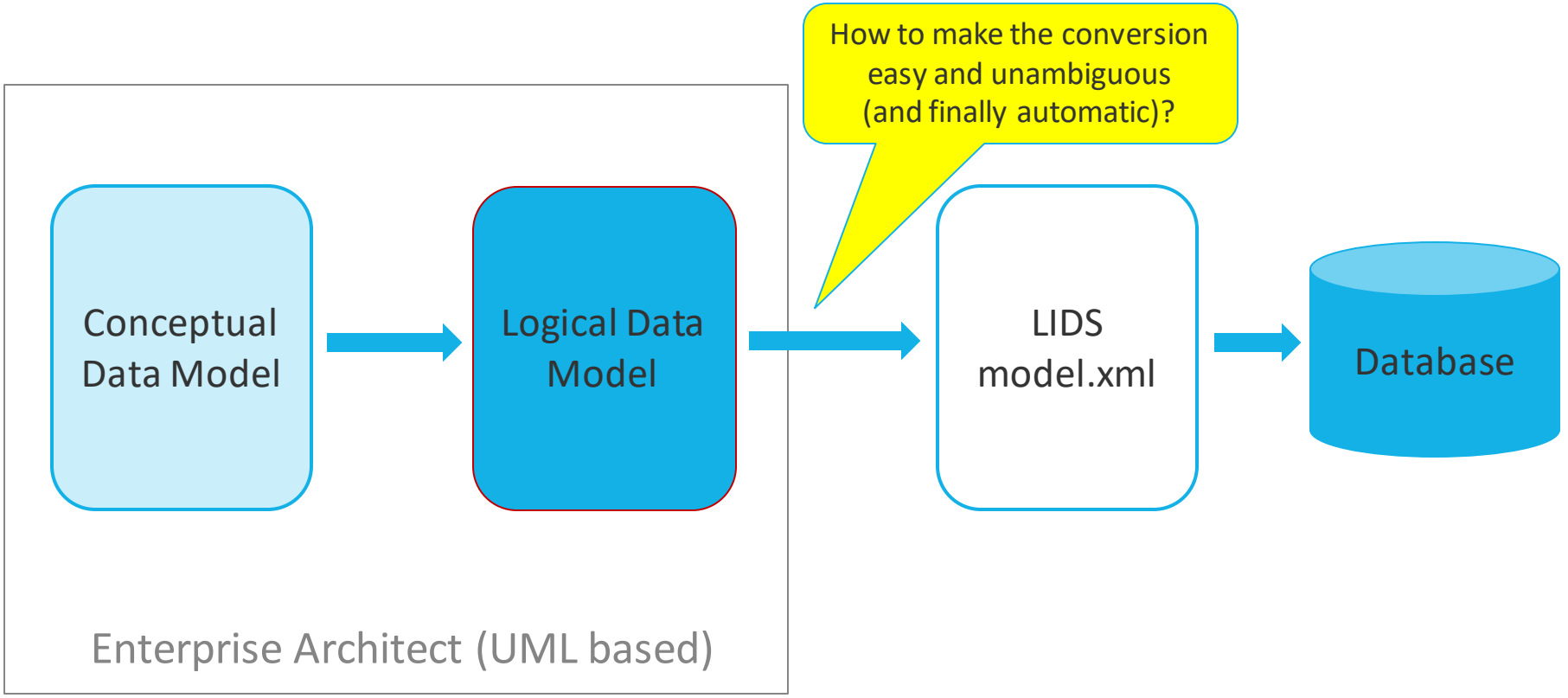




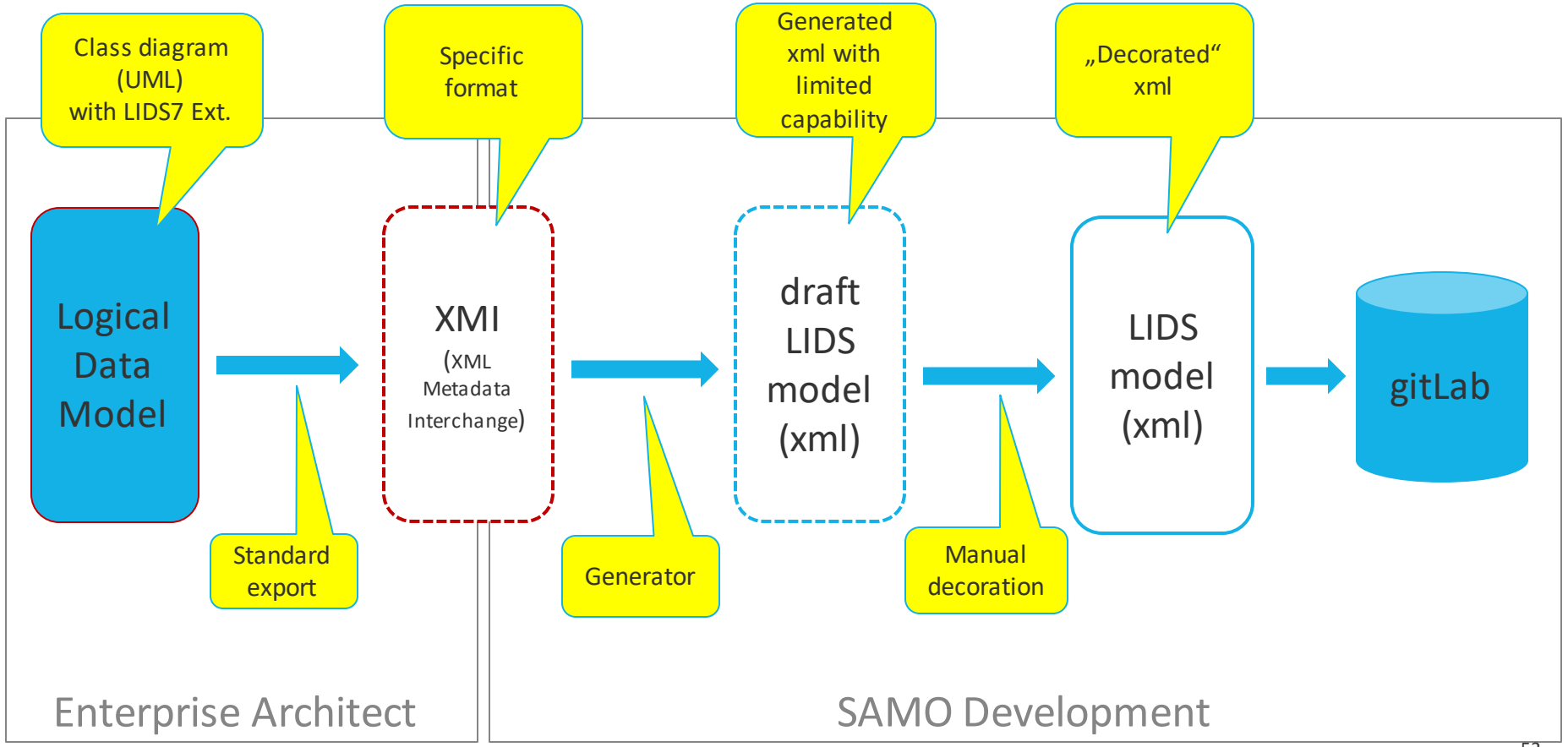
asseco
CENTRAL EUROPE

„Production line“ for LIDS model

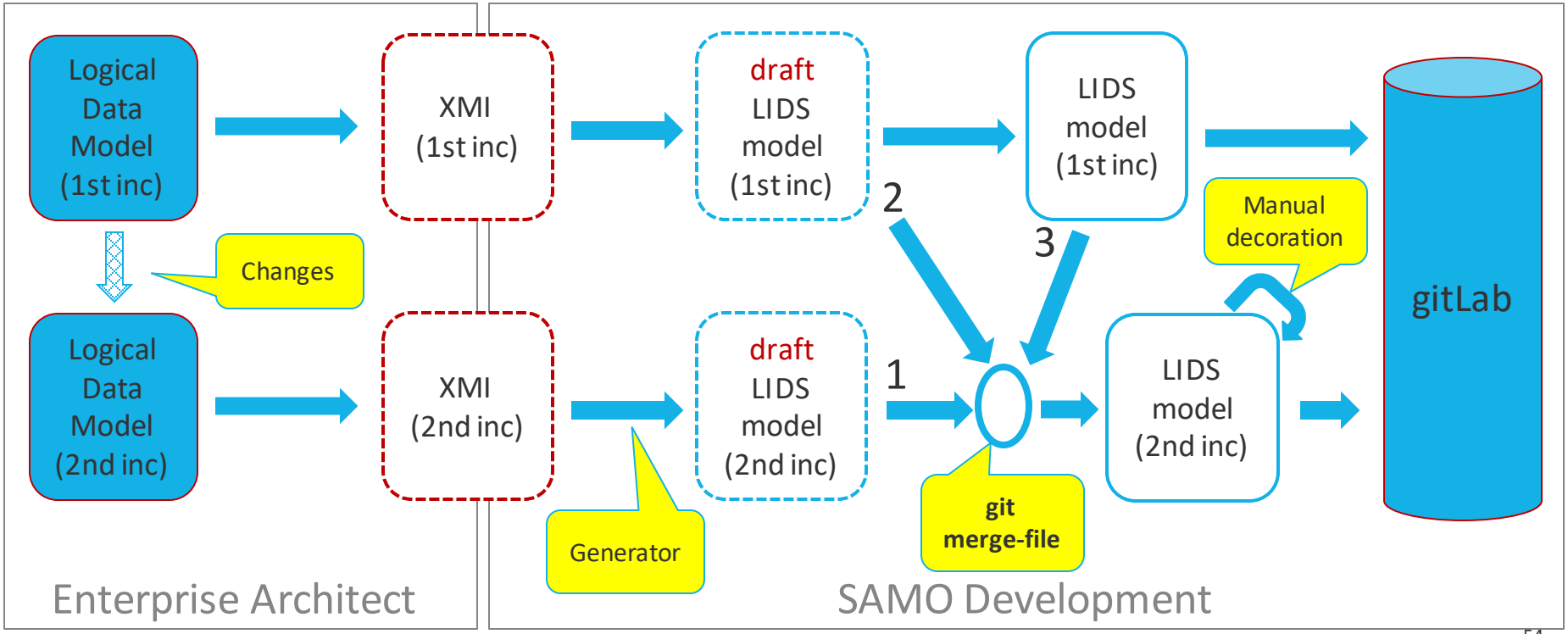
SAMO Data Model



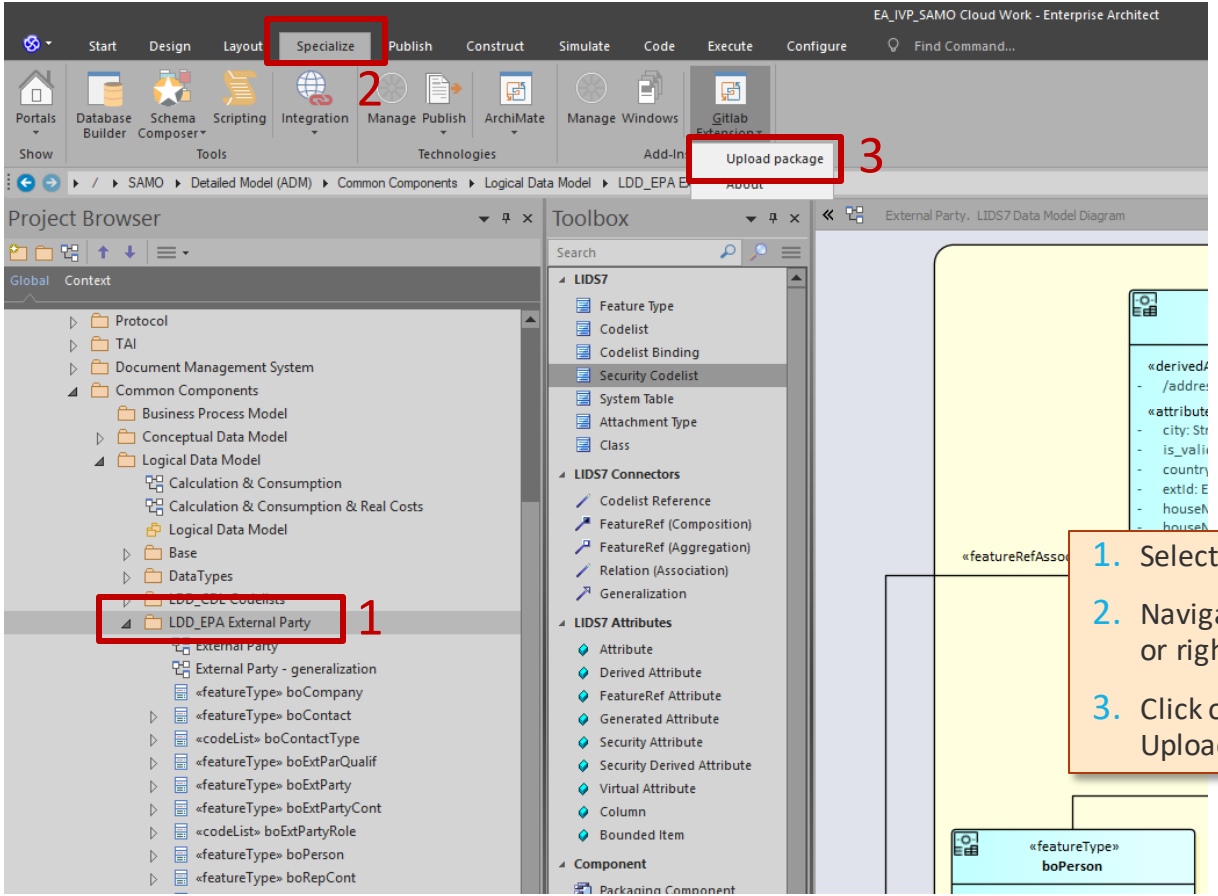
Automatization - concept



Automatization – increments



EA2LIDS Add-in



1. Select package in project browser
2. Navigate into Specialize category in top menu or right-click to see the context menu
3. Click on Gitlab Extension module and then on Upload package button

EA2LIDS Add-in

1. You can see pipeline steps in top left corner of the window as colored rectangles with name of the step below them. If pipeline step was successful, the color is green, red if otherwise.
2. Link to gitlab commit and links to generated file and decorated file are in section below pipeline steps.
3. Large text area is supposed to show log messages. There is a standalone log section for every pipeline step with same name as the pipeline step.
4. Generator version is located in top right corner.

Automatization - support

- Add-in EA
- Git support

- Impact
  - → All team members have to work in the same way
  - → Logical Data Model in EA is the development artefact (!)
    - No „sketching“
    - Problem has to be fixed in the source (= EA)
    - Manual „decoration“ only for structures that are not supported in Generator

asseco
CENTRAL EUROPE

Asseco Central Europe, a.s.

Trenčianska 56/A
821 09 Bratislava
Slovak Republic
Phone: +421 2 20 838 400

Budějovická 778/3a
140 00 Praha 4
Czech Republic
Phone: +420 234 292 500

sales@asseco-ce.com

asseco.com/ce

Solutions
for demanding
business.

Legal disclaimer

This presentation is the property of Asseco Central Europe (Asseco CE) business group. Information presented serves for marketing purposes only and constitutes neither an offer to sell nor a solicitation to buy. Asseco CE accepts no liability whatsoever for any loss arising directly or indirectly from the use of, reliance of any information contained in this presentation or for any omission of the information. The processing, copying, recording on information carriers, as well as making this presentation or any part thereof available in any way to third parties requires the prior consent of Asseco CE member.