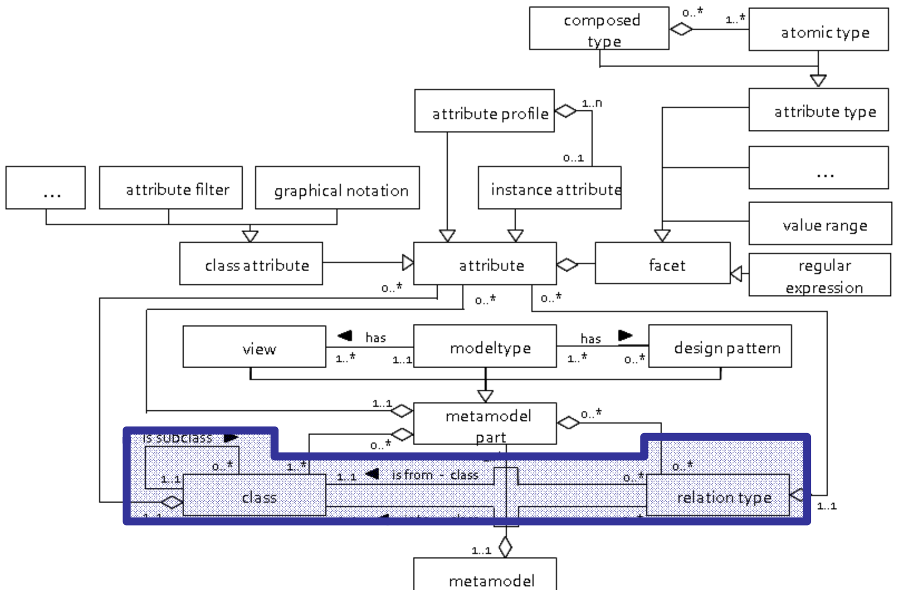
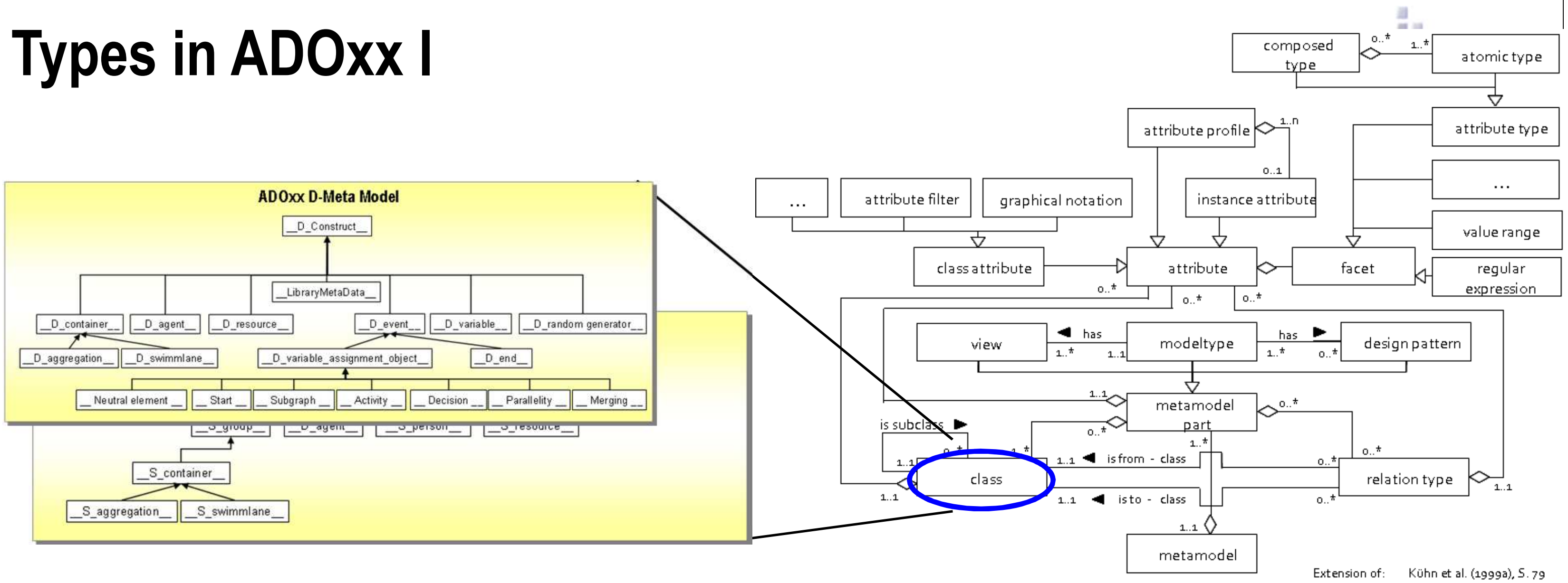


Extension of: Kühn et al. (1999a), S. 79

1. CLASSES and RELATIONS

Class Types in ADOxx I



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

- **Pre-defined Abstract Classes (ADOxx meta model class)**
  - Pre-defined abstract classes are classes that are provided by ADOxx with a given semantic and basic syntax in form of attributes. They can be used to inherit the pre-defined syntax and the attributes to either self-defined abstract classes or to classes.
  - ADOxx functionality that is provided for the pre-defined abstract classes can be used for any inherited concrete class. Hence pre-defined and provided ADOxx functionality is consumed due to inheritance of such pre-defined abstract classes.
  - Pre-defined abstract classes are the ADOxx meta model, hence they exist in every meta model based on ADOxx.
  - Nomenclature: __ Class Name __

Class Types in ADOxx II

- **Abstract Classes**
  - Abstract classes are self-defined classes enabling to structure the meta model and define syntax in form of attributes and semantic, which is inherited by sub-classes.
  - Abstract classes either inherit from the root class of the meta model, or from any other class of the meta model. Hence, they inherit the behaviour from their super-class – which is often a pre-defined abstract class from the ADOxx meta model.
  - Abstract classes enable an efficient meta model, hence they may not be in every ADOxx meta model.
  - Nomenclature: _ Class Name _
- **(Concrete) Classes**
  - Classes are self-defined classes defining a concrete modelling class that can be used, when applying the corresponding modelling language. Hence all model objects in every model created on ADOxx is an instance of a class.
  - Classes inherit the semantic and the attributes from the Pre-defined abstract class and additionally - in case of inheriting - from the abstract class.
  - Classes enable the realisation of a concrete meta model.
  - Nomencladure: Class Name

Selected Pre-defined ADOxx classes for a "Graph-based environment " I

- __ D_Construct ___
  - Super class for „graph-based“ pre-defined meta model.
- __ D_Container __
  - Container class provide the relation „is-inside“, hence every object a drawn on the model having its x/y coordinates within the drawing area of any container b has the relation a Ris-inside b.
- __D_aggregation__
  - Aggregation inherits from __D_Container__, hence also provides the „is-inside“ relation and enables a self-defined „drawing area“. E.g. resizeable rectangel.
- __D_swimmlane__
  - Swimmlane inherits form __D_Container__, hence also provides the „is-inside“ relation but only enables either rows (x=0 to x= maximum) or colums (y= 0 to y= maxium) as possible „drawing area“. E.g. three colums one for input, one for processing, one for output

Selected Pre-defined ADOxx classes for a "Graph-based environment " II

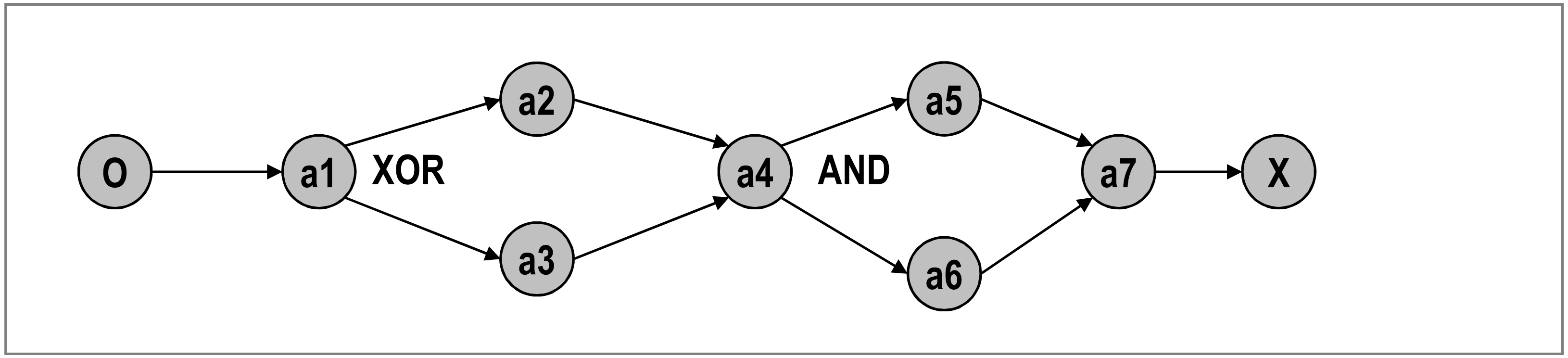
- **__ D_Event ___**
  - Event encapsolates all possible notes of a graph and distinguishes between “D_variable_assignment_object” and “D_end”.
- **__ D_end __**
  - The end concludes the graph and finishes state changes.
- **__D_variable_assignment_objects__**
  - Variable assignment objects enable the change of the state. The state is stored in variables, hence each of the following concepts have the potential to change the status of variables within a graph:
  - Neutral element, start, subgraph, activity, decision, parallelity, merging
- **__D_Neutral element__**
  - Neutral elements do not participate in executing the graph but only display references or state the status.
- **__D_Start__**
  - Start is the starting node of the graph.

Selected Pre-defined ADOxx classes for a "Graph-based environment " III

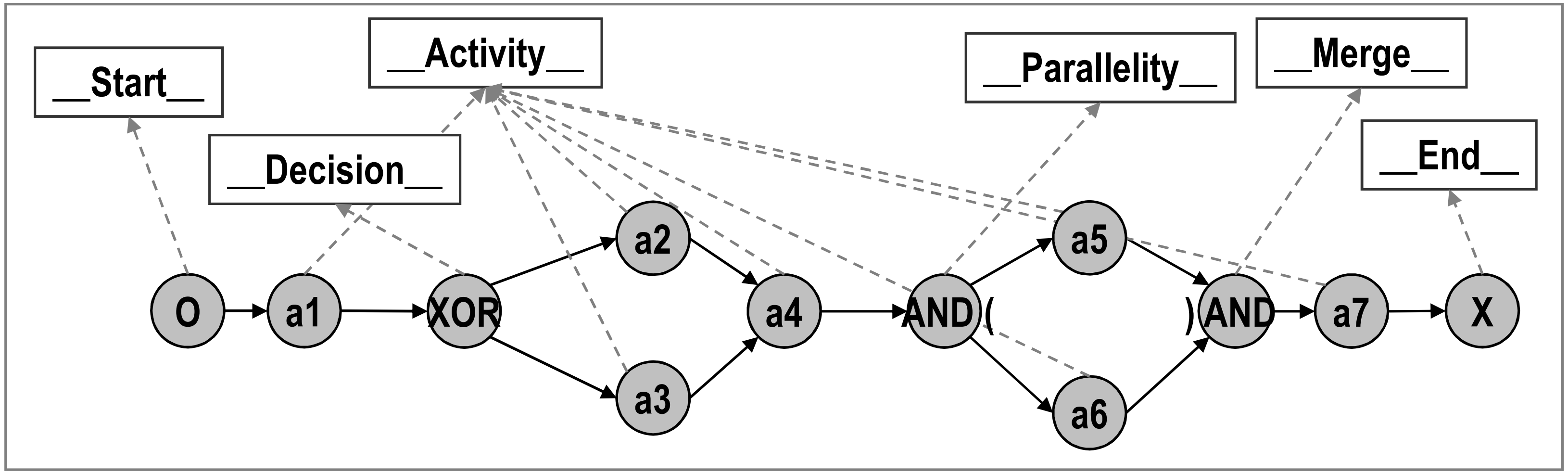
- __ Subgraph ___
  - Subgraph substitutes a sub-graph in the graph to make complex graphs more readable. Technically the subgraph is a pointer to another graph.
- __ Activity__
  - Activity is a node in the graph that performs the typical actions the graph is designed for. Activities are transforming input into output.
- __Decisions__
  - Decisions split the graph in several alternative paths.
- __Parallelity__
  - Parallelity starts a synchronized path of a graph.
- __Merging__
  - Merging ends a synchronized path of a graph.

Selected Pre-defined ADOxx classes for a "Graph-based environment" IV

Sample Graph



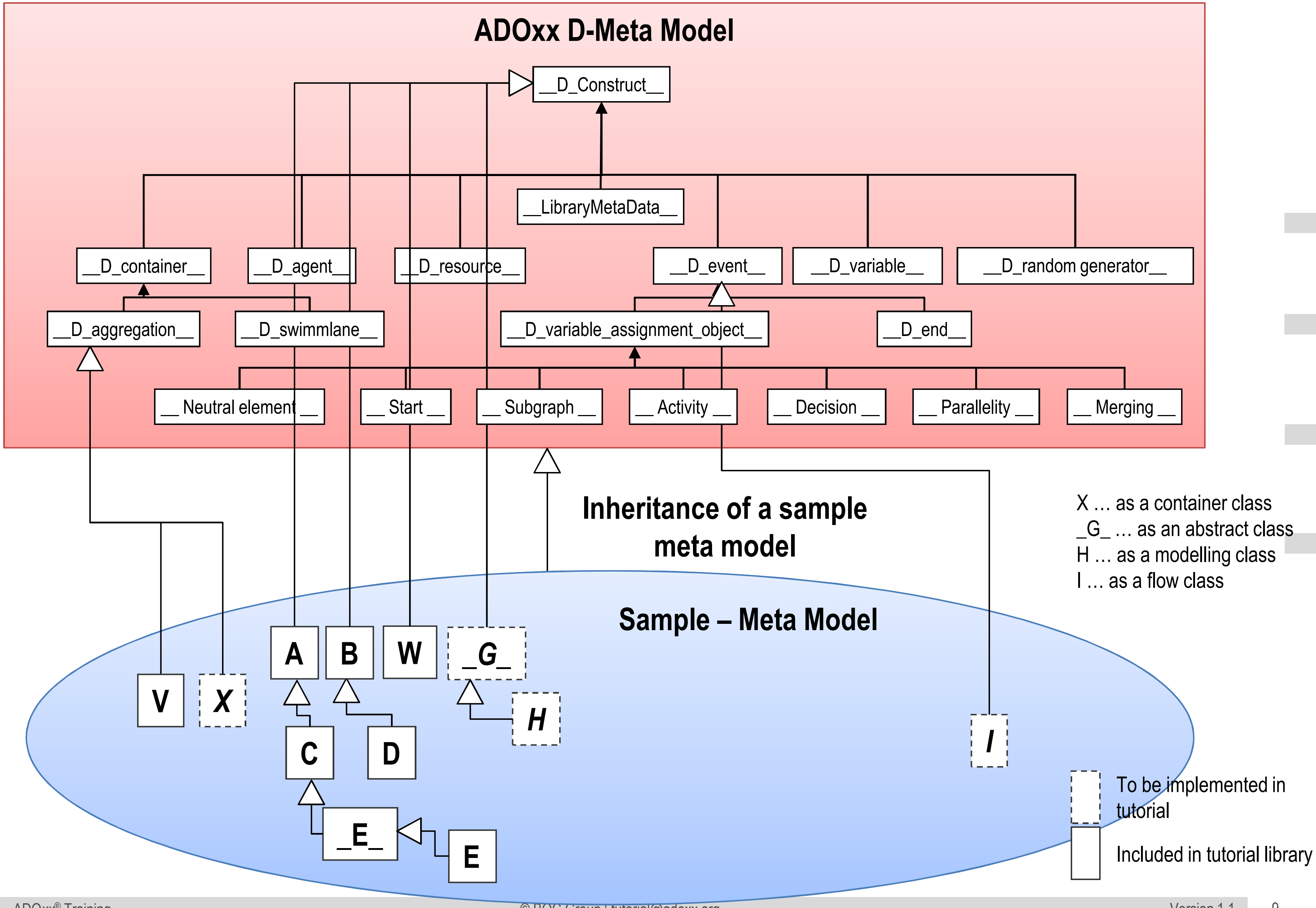
Possible mapping of graph to ADOxx meta model



Selected Pre-defined ADOxx classes for a "Graph-based environment" V

- **__ D_variable ___**
  - Variables are objects that store a certain status of the graph. Hence different variables can be defined, describing different aspects of a graph.
- **__ D_random_generator __**
  - Random generator creates random figures that can be assigned to variables. This is used for simulation.
- **__D_resources__**
  - Resources are properties of graph-nodes represented in an own class hierarchy. Hence descriptive properties need not only be defined as attributes of graph nodes but can be described as classes using class hierarchy from resources.

ADOxx Library Language (ALL)



ADOxx D-Meta Model

Inheritance of a sample meta model

Sample – Meta Model

X … as a container class
_G_ … as an abstract class
H … as a modelling class
I … as a flow class

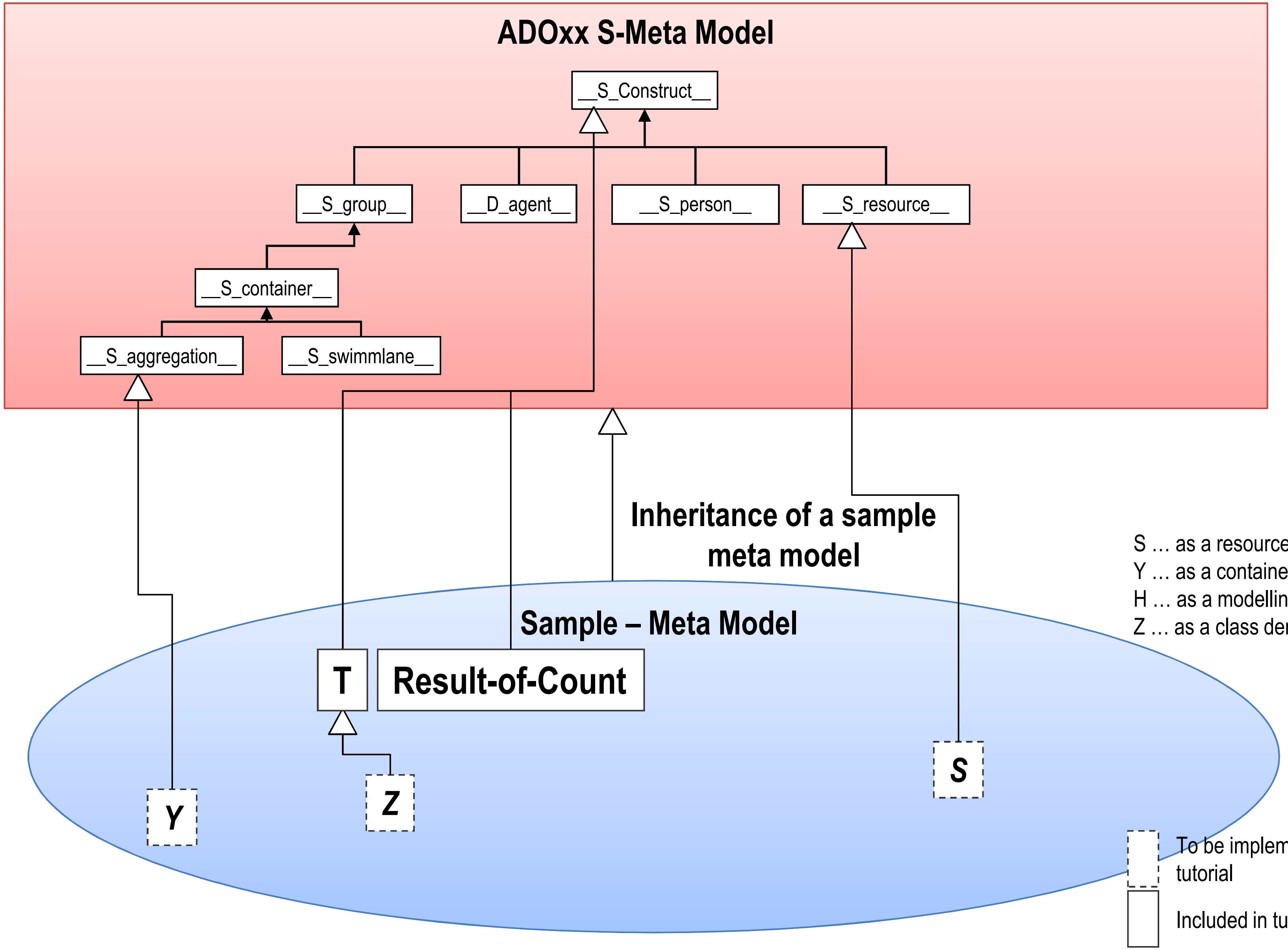
To be implemented in tutorial

Included in tutorial library

Selected Pre-defined ADOxx classes for a "Tree-based environment"

- __ S_Construct ___
  - Super class for „hierarchy” pre-defined meta model.
- __S_Group__
  - Group is a tree node
- __ S_Container __, __S_aggregation__, __S_swimmlane__
  - Is a special form of a tree-node, same as in __D_Container__
- __S_resource__
  - Resources are properties of tree-nodes represented in an own class hierarchy. Hence descriptive properties need not only be defined as attributes of tree nodes but can be described as classes using class hierarchy from resources.
- __S_person__
  - In case persons are represented a special class is reserved for implementing person depending behaviour (privacy etc.).

ADOxx Library Language (ALL)



S … as a resource class
Y … as a container class
H … as a modelling class
Z … as a class derived from T

□ To be implemented in tutorial

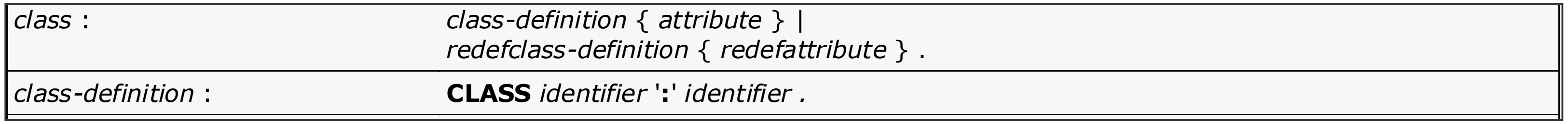
□ Included in tutorial library

Realisation of Meta Model

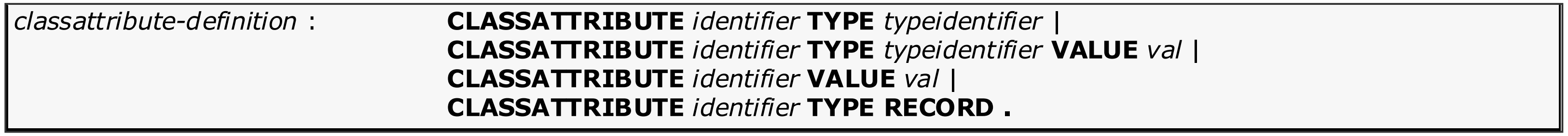
Specification of a meta model in ALL

1. Specify the meta model starting from the „Empty Meta Model“ and add classes etc. with ALL using a text editor. Abstract class is defined by the classattribute isabstract.

2. Translate ALL into the ADOxx interpretable ABL format and import the meta model into ADOxx.



class :	*class-definition* { *attribute* } | *redefclass-definition* { *redefattribute* } .
class-definition :	**CLASS** *identifier* '**:**' *identifier .*



classattribute-definition :	**CLASSATTRIBUTE** *identifier* **TYPE** *typeidentifier* | **CLASSATTRIBUTE** *identifier* **TYPE** *typeidentifier* **VALUE** *val* | **CLASSATTRIBUTE** *identifier* **VALUE** *val* | **CLASSATTRIBUTE** *identifier* **TYPE RECORD .**

Definition of a Modeling Class

Keyword

Inheritance from a modeling class from the meta-model

```
//====================================
CLASS <Aggregation> : <__BP_Aggregation__>
//====================================

//--- Class <Aggregation> - Class attributes-


//--- Class <Aggregation> - Instance attributes-
```

Predefined abstract classes

comments