

In the following we will define a second type of model, for describing resources needed during the cooking processes. The main concept here will be `CookingResource`, specialized in 2 types of resources: `CookingIngredient` and `CookingObject`. The only relation will be “always requires”, to indicate that:

- a certain ingredient always requires a certain tool (e.g., milk always requires to have some kind of recipient – e.g., a bottle);
- a certain tool must always be used with another tool (e.g., mixer always requires some recipient).

In other words, this relation to describe dependencies, to connect any kind of resource (both ingredients and tools/objects) to some objects that they depend on (must be used together with).

Create the concept `CookingResource` under `_D_construct` (using `New class`). Create `CookingIngredient` and `CookingObject` under `CookingResource`.

Create the relation “always requires” with `New relationclass`, from any `Cooking Resource` to a `Cooking Object`.

Define the following `GraphRep` for `CookingIngredient` (a simple red circle with the Name below):

```
GRAPHREP
```

```
PEN color:red
```

```
ELLIPSE rx:1.2cm ry:1.2cm
```

```
ATTR "Name"
```

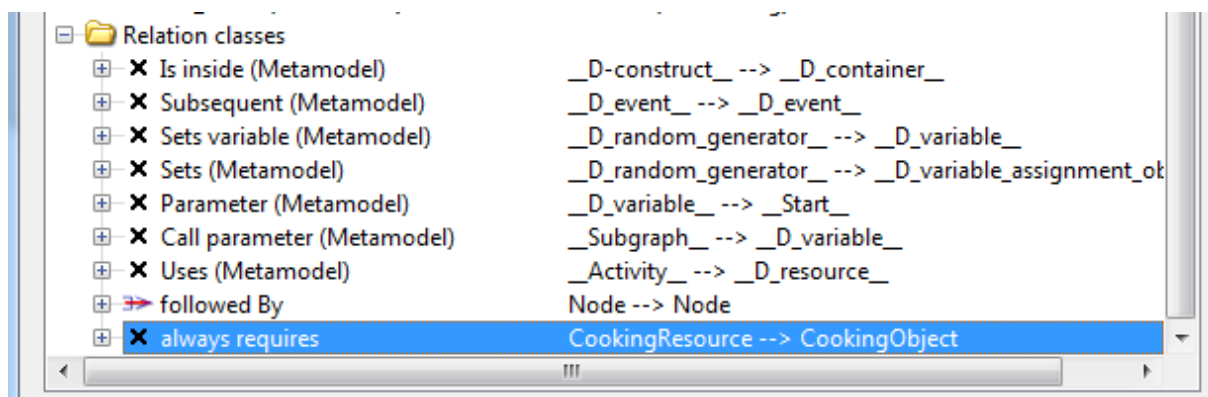
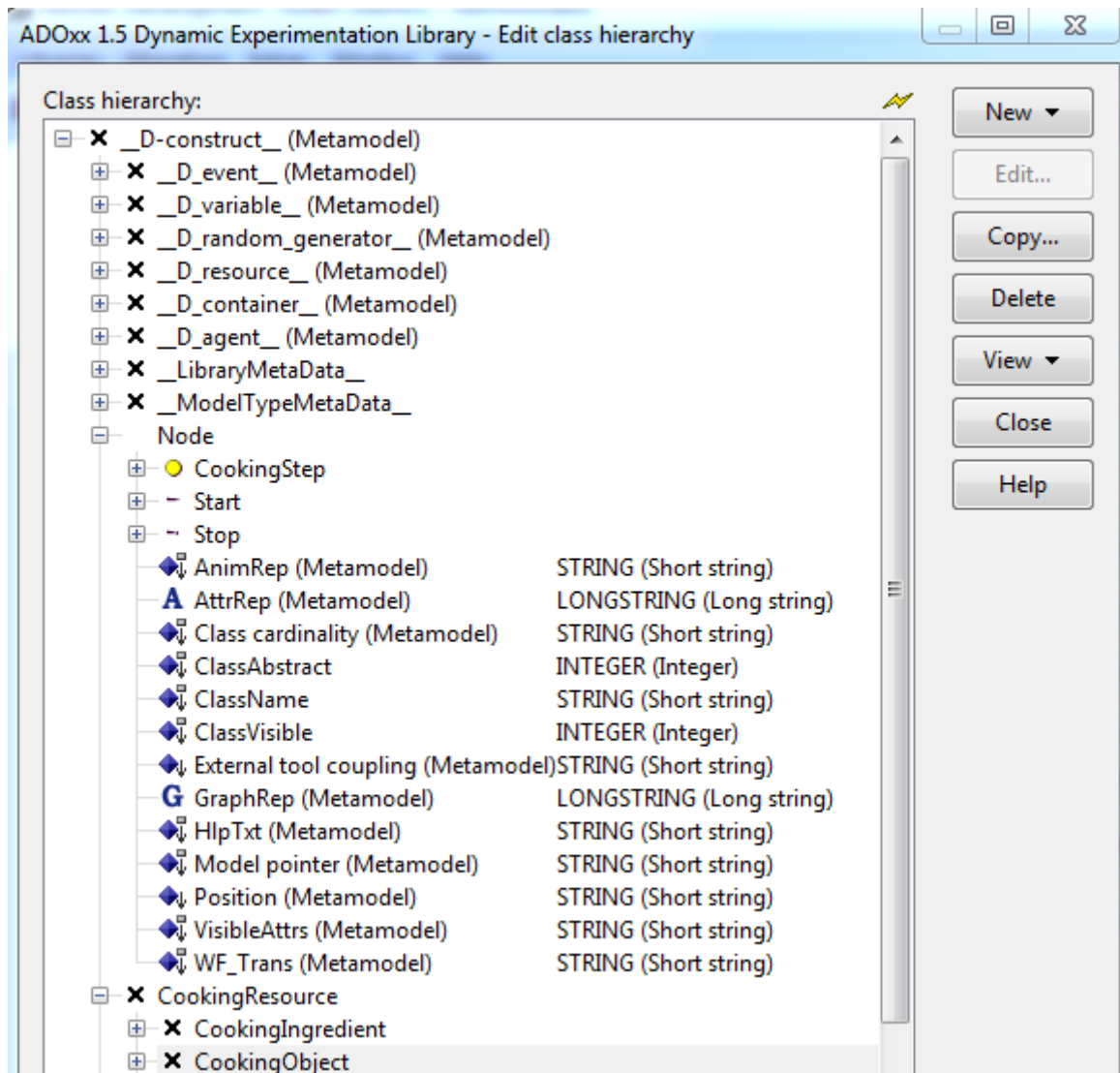
Define the following `GraphRep` for `CookingObject` (a simple rectangle)

```
GRAPHREP
```

```
PEN color:blue
```

```
RECTANGLE x:-1cm y:-1cm w:2cm h:2cm
```

```
ATTR "Name" y:1cm w:c
```



Define the following GraphRep for the relation "always requires":

GRAPHREP

PEN style:dot w:0.1cm endcap:round

EDGE

TEXT "always requires" w:c

END

ELLIPSE rx:0.3cm ry:0.3cm

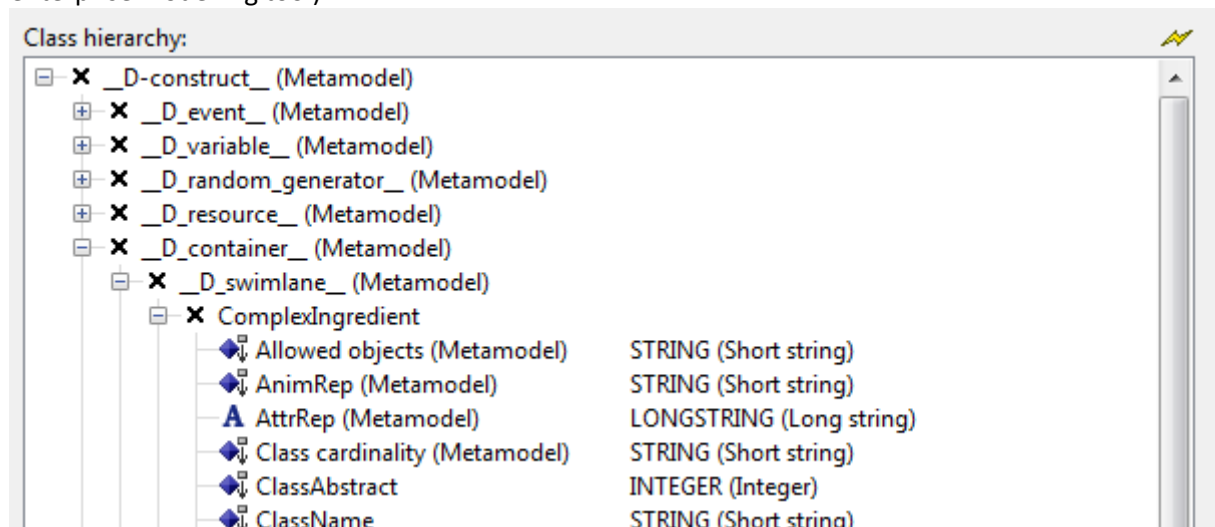
Notice the new elements:

- The PEN has now a line style (dotted) and the endcap parameter which will make the dots slightly rounded
- A START section is missing, hence no shape is defined for the beginning of the connector
- The EDGE section includes only a static text
- The END section will make the “arrow head” look like a circle

Now we will also include a container concept, which is a graphical shape that may contain other symbols to suggest some grouping. In our case, it will be called a ComplexIngredient and will be used to group together multiple ingredients. For containers, _D_construct provides a dedicated subconcept called _D_container, which in turn provides two types of containers:

- _D_swimlane acts like a process swimlane (it spreads vertically or horizontally and it can be moved or resized only on one dimension)
- _D_aggregation is a more flexible container – a box that can be resized and positioned more freely.

Create the ComplexIngredient under _D_swimlane, to benefit from its built-in behavior (you may notice other _D_construct specializations – they are the ones that you can also find in the ADONIS enterprise modelling tool).



The following is the GraphRep for our swimlanes:

GRAPHREP swimlane:horizontal

PEN w:0.1cm color:green

TABLE w:10cm h:4cm cols:3 rows:1

w1:3cm w2:1cm w3:100%

h1:100%

STRETCH off

RECTANGLE x:(tabx0) y:(taby0) w:(tabw1+tabw2+tabw3) h:(tabh1)

LINE x1:(tabx1) y1:(taby0) x2:(tabx1) y2:(taby1)

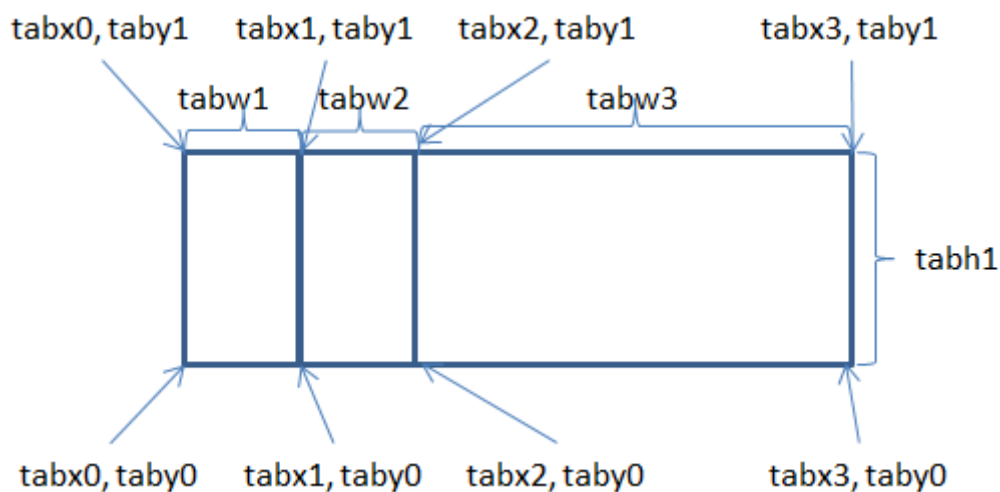
LINE x1:(tabx2) y1:(taby0) x2:(tabx2) y2:(taby1)

ATTR "Name" x:(tabw1/2) y:(tabh1/2) w:c h:c

Explanations:

- The swimlane parameter sets the general behavior to horizontal (meaning that the swimlane will stretch over the entire width of the modelling canvas, and the resizing or repositioning will only be possible on the vertical axis)
- The TABLE command defines a table frame of 3 columns and 1 row. The parameters w1-w3 define the widths of the 3 columns (the last one occupies the entire space left). The parameter h1 occupies the entire height since the table only has 1 row.
- The ATTR command inserts the name in the center of the first cell.

Notice that the TABLE command only defines an invisible table frame, it does not actually draw the table borders! TABLE only generates some special names for the coordinates of the table corners and cell sizes, as shown in the figure below:



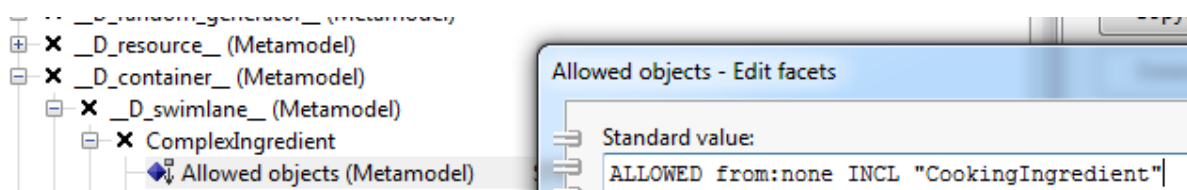
To actually draw the borders, we need to apply the usual drawing commands (LINE, RECTANGLE, POLYLINE etc.) by using these special names:

RECTANGLE was used to draw the outer border. We start from tabx0 and taby0 (built-in names for the first corner of the table), use a width obtained by summing tabw1-to-3 (built-in names for the 3 column widths) and the height tabh1 (built-in name for the height of the first row in the table).

LINE was used to draw the 2 inner borders, by using as coordinates the “corner names” defined by the table frame.

Next, we have to restrict what kind of elements are allowed inside this swimlane. Since it is a complex ingredient, we will only allow the grouping of ingredients. For this, type the following in the predefined attribute “Allowed objects”:

ALLOWED from:none INCL "CookingIngredient"



Now let's define the new model type that allows the use of all the new concepts and the new relation. Just like before, go to the Library Attributes, find the Modi attribute where the general structure of the language is defined in terms of what concept belongs to what model type:

GENERAL order-of-classes:custom

```
MODELTYPE "Cooking Recipes"  
INCL "CookingStep"  
INCL "Start"  
INCL "Stop"  
INCL "followed By"
```

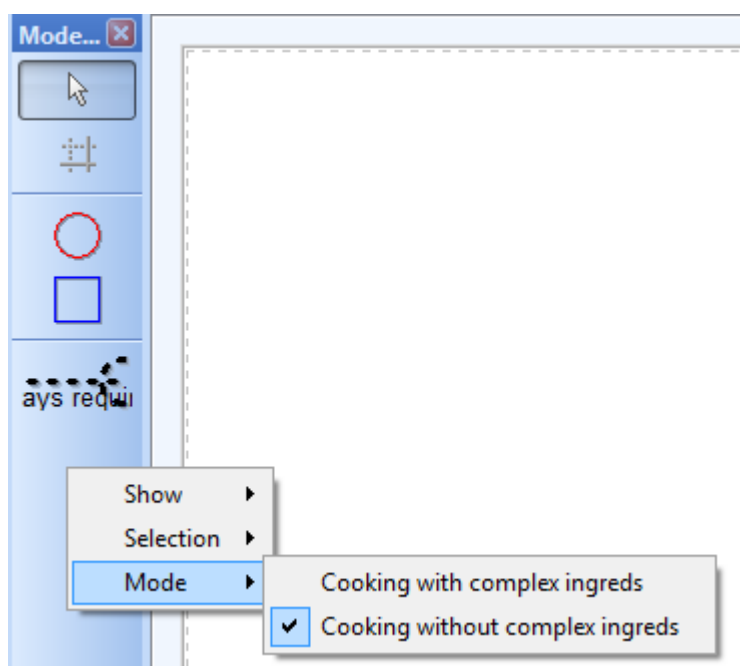
```
MODELTYPE "Cooking Resources"  
INCL "ComplexIngredient"  
INCL "CookingIngredient"  
INCL "CookingObject"  
INCL "always requires"  
MODE "Cooking with complex ingreds" from:all  
MODE "Cooking without complex ingreds" from:all  
EXCL "ComplexIngredient"
```

Notice that we added a model type "Cooking Resources" which includes the new concepts and the new relation. In addition, we also define 2 "modes":

- One that allows us to use all these concepts (from:all)
- One that provides a reduced set of concepts (from:all with an exclusion of ComplexIngredient).

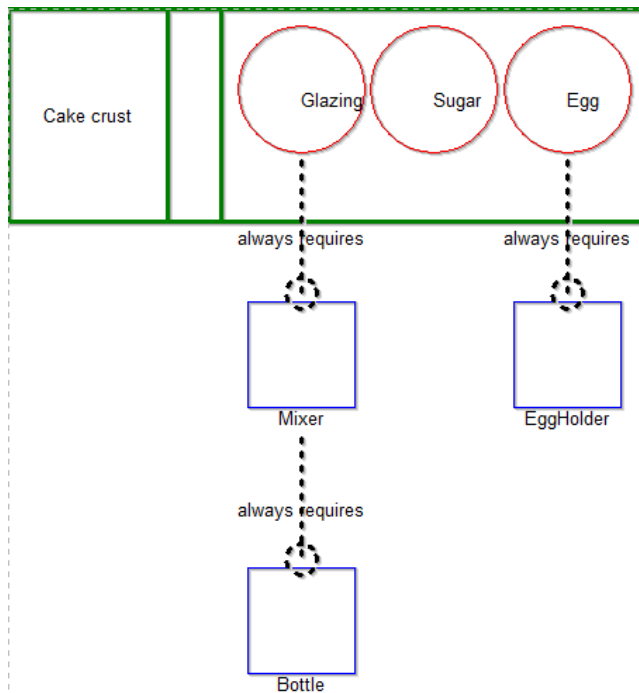
The modes are used when the number of concepts in a model type is very big and there is a danger that not all of them fit in the toolbar. In this case, the model type may have simplified versions, where not all concepts are available. In our case, we have the possibility of ignoring the swimlanes.

Go to the modelling tool. A right-click on the concept toolbar allows us to select the desired mode. Notice that in the second mode, the swimlane for complex ingredients is not available.



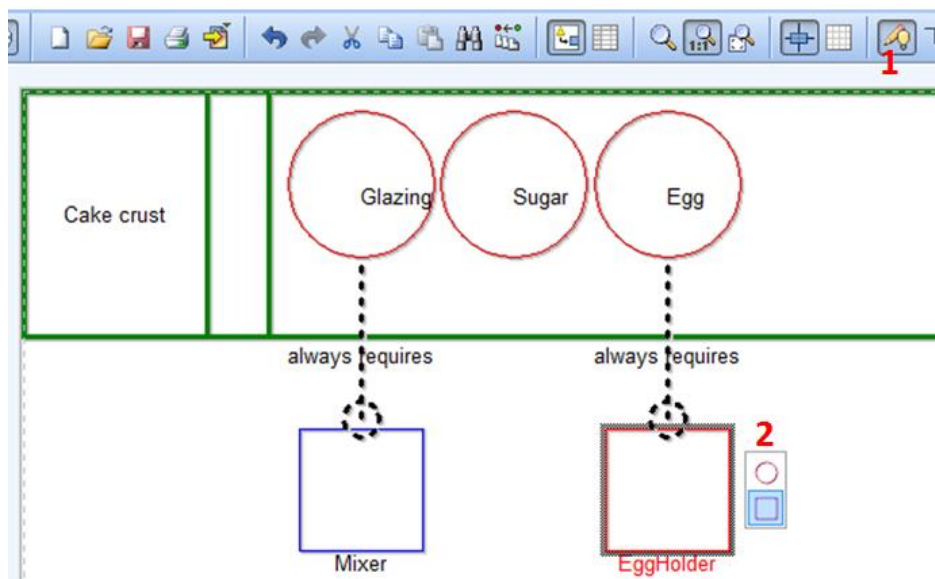
Select the full Mode (“Cooking with complex ingreds”), which allows us to create a model like the one in the next figure.

- Notice that CookingObjects cannot be inserted in the swimlane.
- Notice that the swimlane can only be resized vertically. If you add multiple swimlanes and you can move them up and down to reorder them.
- Notice that the dotted connector cannot connect two ingredients, but it can connect an ingredient to a cooking object or two cooking objects.



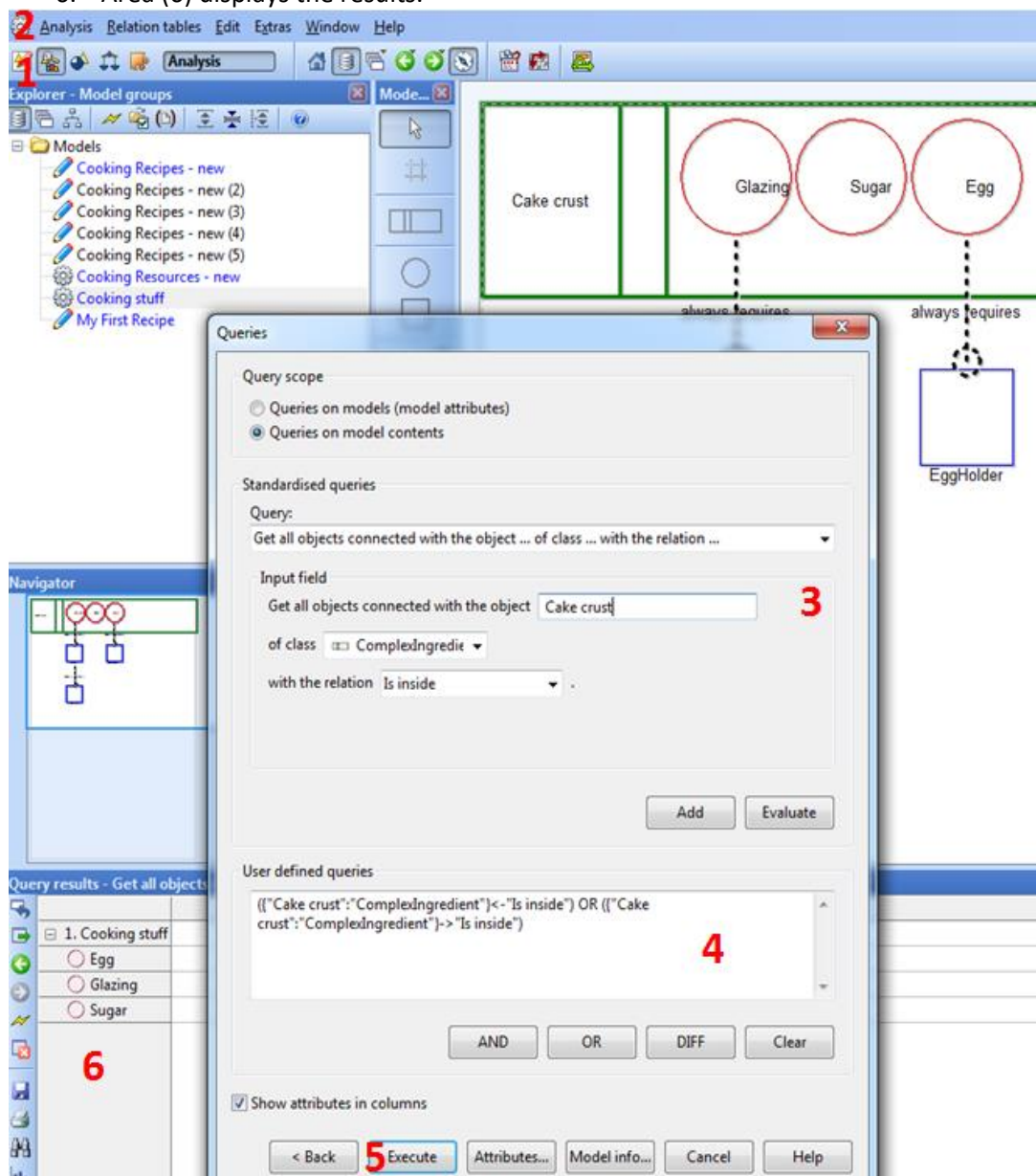
Also notice some interesting functionality that is built-in based on the syntactic definition of our language:

1. The button highlighted with (1) activates a “modelling assistant”
2. Every time you select an element, you will see near it the shapes of concepts that are allowed to be connected to it, based on the metamodel definition (here, the circle and the square). Selecting one of them will automatically generate an element of that type, plus the connector to it (thus, it is a usability mechanism defined on top of the language syntax!)



Another built-in functionality which adapts to the language syntax and semantics is the query engine which allows us to run queries on the model contents, regardless of the type of model:

1. First you need the button marked with (1) to activate the Analysis menu
2. Then select from the menu Analysis-Queries/Reports, followed by a selection of the model you want to query
3. Several types of queries are available – some of them based on *semantics* (get all objects of a certain class, get all objects with a certain value for a specific attribute), others on *syntax* (get all objects connected to a specific object, or with a specific connector etc.). Different drop-down lists allow us to select the concepts, relations and properties that are relevant for the model we want to query. In this example we create a query that will return all the objects contained in the “Cake crust” swimlane. Notice that we selected the relation “Is inside” which is a pre-defined relation that ADOxx generates between any container and its contained elements (therefore a relation based on position, not based on a visible connector!).
4. The lower area displays the query in the AQL language (the internal query language of ADOxx)
5. The Execute button will run the query
6. Area (6) displays the results.



Try to use the same query engine to a Cooking Recipe model and you will notice that the selection lists will adapt to the concepts present in that model. The next example builds a semantics-based query: “Get all cooking steps with a cost < 5”

The screenshot shows a modelling tool interface. On the left, a list of models is visible: 'Cooking Recipes - new (2)', 'Cooking Recipes - new (3)', 'Cooking Recipes - new (4)', 'Cooking Recipes - new (5)', 'Cooking Resources - new', 'Cooking stuff', and 'My First Recipe'. The main area displays a diagram with three nodes: 'Prepare tools' (a small blue square), 'Add 300 grams of potatoes' (a large yellow circle), and 'Add salt' (a small yellow circle). Arrows connect these nodes, with a red arrow from 'Prepare tools' to 'Add 300 grams of potatoes' labeled 'cost not estimated'. A 'Queries' window is open in the foreground, showing a query engine interface. The 'Query scope' section has 'Queries on model contents' selected. The 'Standardised queries' section shows a query: 'Get all objects of class ... with attribute ...'. The 'Input field' section shows 'Get all objects of class' with a dropdown set to 'CookingStep', and 'with attribute' with a dropdown set to 'Cost', followed by a comparison operator '<' and a value '5'. The 'User defined queries' section shows a query: '<'CookingStep'>[?'Cost' < 5]'. The background also shows a 'Query results' window with a table of results.

Query results - Get all objects of class "CookingStep"	Cost
1. Cooking Recipes - new (4)	
Add salt	2
Prepare tools	0

As we add additional semantics to each model types, the querying possibilities will be enriched. However, the limitation of AQL is that it cannot access data that is external to the modelling tool and it cannot create queries that cross between multiple models. This is why we need to export the knowledge captured in diagrams to RDF - a semantic format that will preserve the relations between models and will allow us to connect models to any external data.

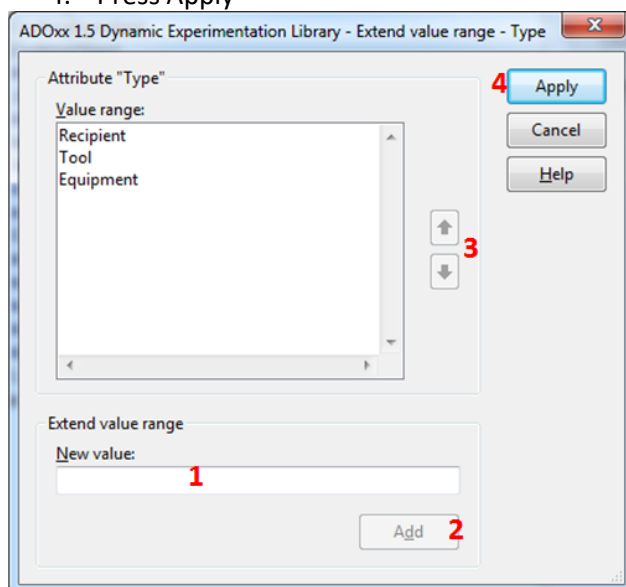
In the following, we will extend the semantics of models with some additional properties:

- The CookingObject will get a "Type" property to indicate if it's a Tool, a Recipient or some Equipment. Obviously, we could create some new concepts for these, by adding some subclasses in the concept hierarchy. But this is necessary only in the following cases:
 - if some semantic specialization must be applied (i.e., distinct properties added to each specialized concept). If all we want to have is a type distinction, a selection list property is sufficient. Even if we want to have graphical distinction, we can program the GraphRep to look differently based on the type attribute value (as already discussed)
 - if we want to provide one-click access for the specialized concepts: the modeller will be able to pick them with a single click on the concept toolbar. If we add the type as a property, then the modeller will have to go to the property sheet ("notebook") in order to set the type. This is a usability concern and it is also influenced by how many concepts will be available in the concept toolbar (if too many, the modeller might prefer to reduce them by using the type property!)
- The CookingObject will also get an "Equipment usage" description attribute which will become active only if the Equipment option is selected from the Type list.

- Models can get “model-level attributes”. These are properties that can be edited with right-click on the free modelling area, then selecting Model Attributes. They are properties of a model as a whole, also called “metadata”. By default, any model has the properties of Author, Creation date, Keywords, Description, if it’s the model of a planned situation (“To-be”) or the model of an existing situation (“As-is”) etc. These can be replaced with our own model-level properties. We will add the following attributes to the CookingRepice models:
 - “contributor list”, a multiline string where the modeler will write the names of the people who contributed with knowledge to the model
 - “appropriate for” will be a multi-selection list of options to indicate if the food described by the model is fit for Breakfast, Lunch, Dinner or any combination of these.

First create the new Type property for CookingObject. Right click on the concept, select New attribute. Give it the name “Type” and the type “Enumeration”. This will allow us to have a unique selection list, which can be presented in different ways: drop down, radio buttons, check boxes. After creating the attribute, we get the window below:

1. Define a value to be included in the list
2. Press Add, then repeat the previous step for all values to be presented to the modeller
3. Adjust the order of the values conveniently
4. Press Apply



After creating the attribute, double-click on it to set the default selection value.

Next, create the "Equipment usage" attribute, of type STRING. In the Facets of the attribute, check the box MultilineString to be able to write a longer description text.

Next, make both attributes visible in the property sheet, by using the AttrRep attribute of CookingObject.

- For the Type attribute, indicate how the selection list will be presented (radio, dropdown etc.), by using the ctrltype parameter.
- For the "Equipment usage" attribute, indicate that it should be active only if the "Equipment" value was selected in the Type attribute. This is done by collecting the Type value with AVAL, just like we did before in GraphRep. The "enabled" parameter allows us to declare when should the "Equipment usage" slot become active (based on a boolean test applied to the Type value).

NOTEBOOK

CHAPTER "Description"

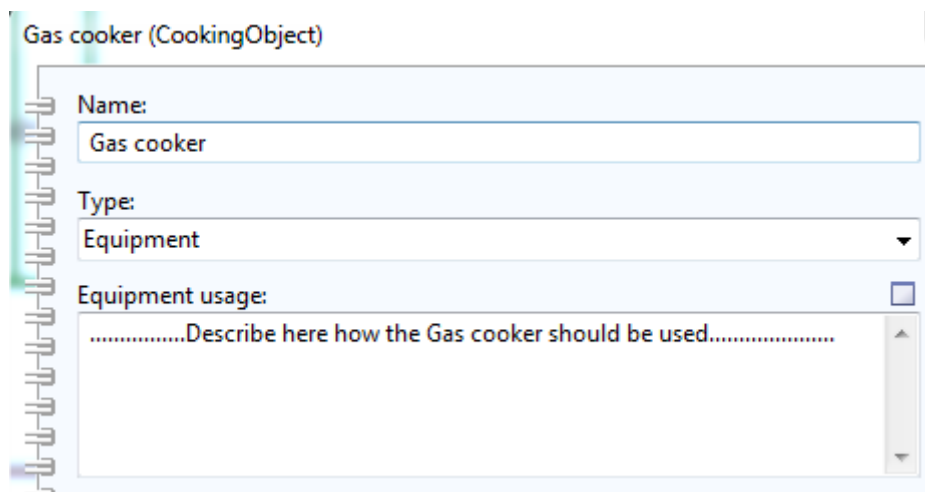
ATTR "Name"

ATTR "Type" ctrltype:dropdown

AVAL t:"Type"

ATTR "Equipment usage" enabled:(t="Equipment")

In the modelling toolkit you should see this behavior in the CookingObject property sheet: the "Equipment usage" box will be active only if the Type "Equipment" is selected:



Gas cooker (CookingObject)

Name:
Gas cooker

Type:
Equipment

Equipment usage: ☒

.....Describe here how the Gas cooker should be used.....

Return to the Development toolkit.

The creation of model-level attributes is a bit more complicated. These are created in the following steps:

- First all model attributes must be defined in the pre-defined class `__ModelTypeMetaData__`, regardless of the model type where they will be included
- Then another attribute must be created in the same class, with the type `STRING` and as content the `AttrRep` code that you would use to make those attributes visible
- Finally, in the Library Attributes, where the model types are defined, they can get an `attrrep` parameter whose value must be the name of the `STRING` attribute where the model-level `AttrRep` code was saved.

The steps can be followed in the next figure:

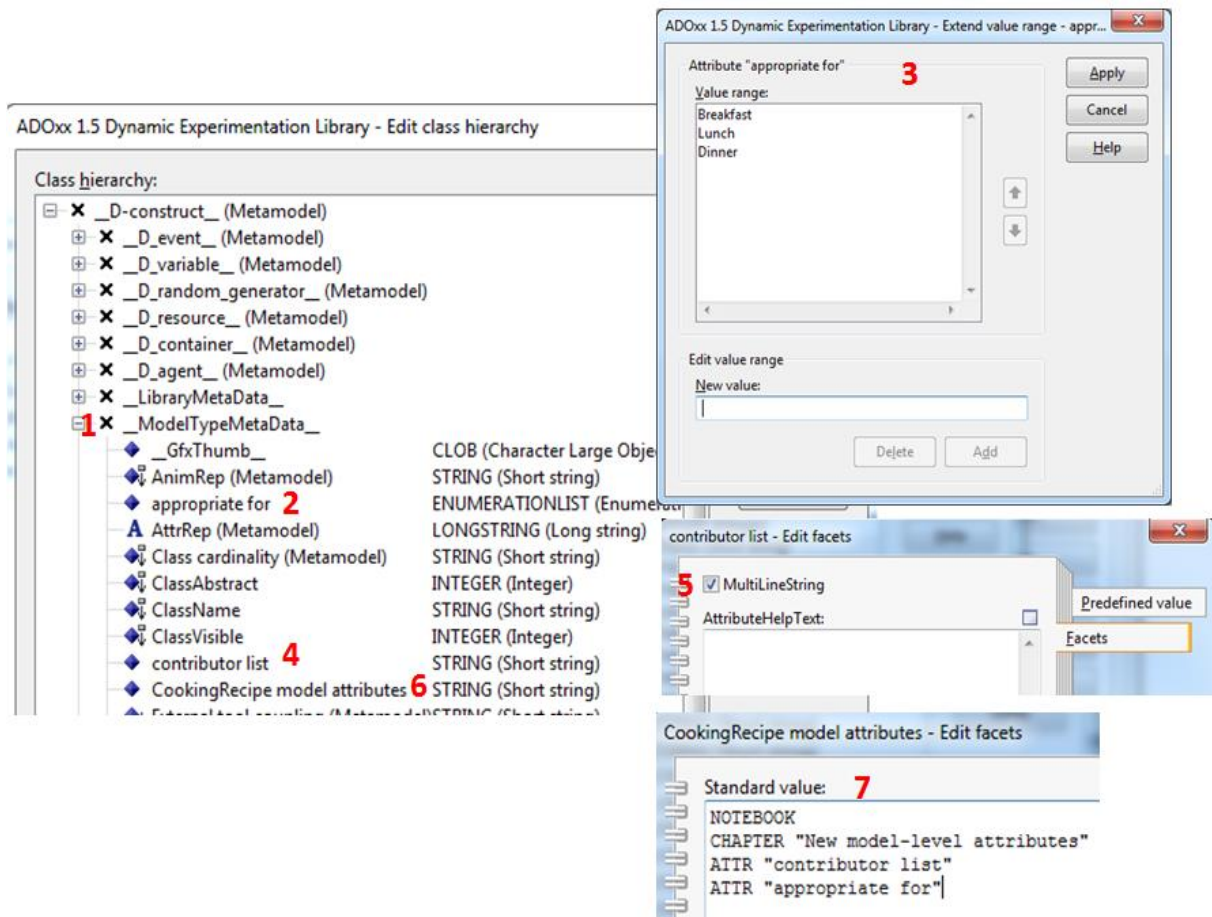
1. Find the predefined class `__ModelTypeMetaData__`
2. Create the "appropriate for" attribute, of type `ENUMERATIONLIST`. This will allow us to create a multi-selection list.
3. Include the selection options: Breakfast, Lunch, Dinner.
4. Create the "contributor list" attribute, of type `STRING`.
5. Check the `MultiLineString` option to have multiple lines (to write on more than a single row)
6. Create the "CookingRecipe model attributes" attribute, of type `STRING`
7. Write in it the `AttrRep` code to make the previously defined attributes visible:

NOTEBOOK

CHAPTER "New model-level attributes"

ATTR "contributor list"

ATTR "appropriate for"

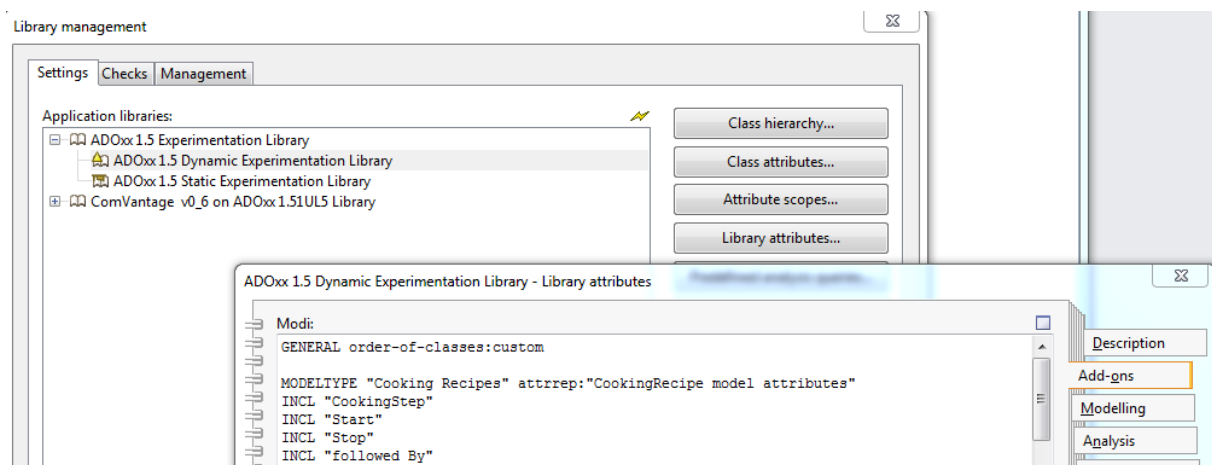


Close the class hierarchy and move to the Library attributes. Add to the Model type definition for Cooking Recipes the parameter attrrep, pointing to where we stored the AttrRep code:

```

MODELTYPE "Cooking Recipes" attrrep:"CookingRecipe model attributes"
INCL "CookingStep"
INCL "Start"
INCL "Stop"
INCL "followed By"

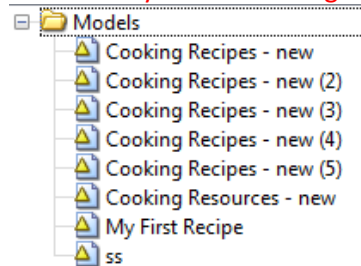
```



We will also make another model-level change. It is useful to define distinctive icons for each model type. By default all models will have the same icon in front of them when listed in the folders of the

modelling tool. It is useful to provide a visual indication of which of the models are recipe models and which are resource models (sometimes the model name does not help).

In order to define customized icons for each model type, we need to import in ADOxx 16 pixel icons. They can be created with any drawing software (e.g., Paint). Or, you can search in Google for “16x16 icons” and you will find large collections of already available icons.



We will use two 16x16 icons found in the Web, to distinguish between model types as shown in the next figure. Do not use bigger pictures, as they cannot be resized!



The two icons must be imported in ADOxx using the same feature as in previous file imports (Extras-File Management-selection of library-Import). After the import, use the Library Attributes to extend the definition of the model types with the parameter bitmap (the files were imported with the names “a.gif” and “b.gif”):

MODELTYPE "Cooking Recipes" attrrep:"CookingRecipe model attributes" **bitmap:"db:\\b.gif"**

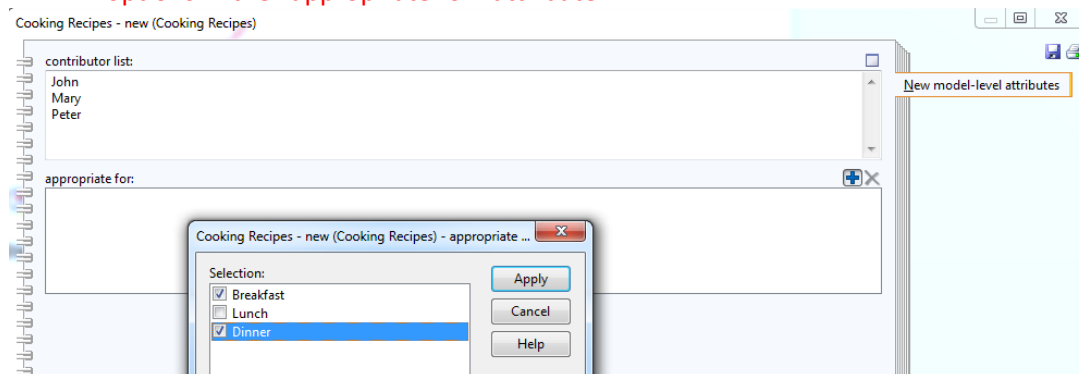
INCL

MODELTYPE "Cooking Resources" **bitmap:"db:\\a.gif"**

INCL

Now start the modelling tool and notice the following:

- The new icons in the model list
- Right-click inside a model of type Cooking Recipes and you will see the new model-level attributes: you can type on multiple lines in the contributor list; you can select multiple options in the “appropriate for” attribute.



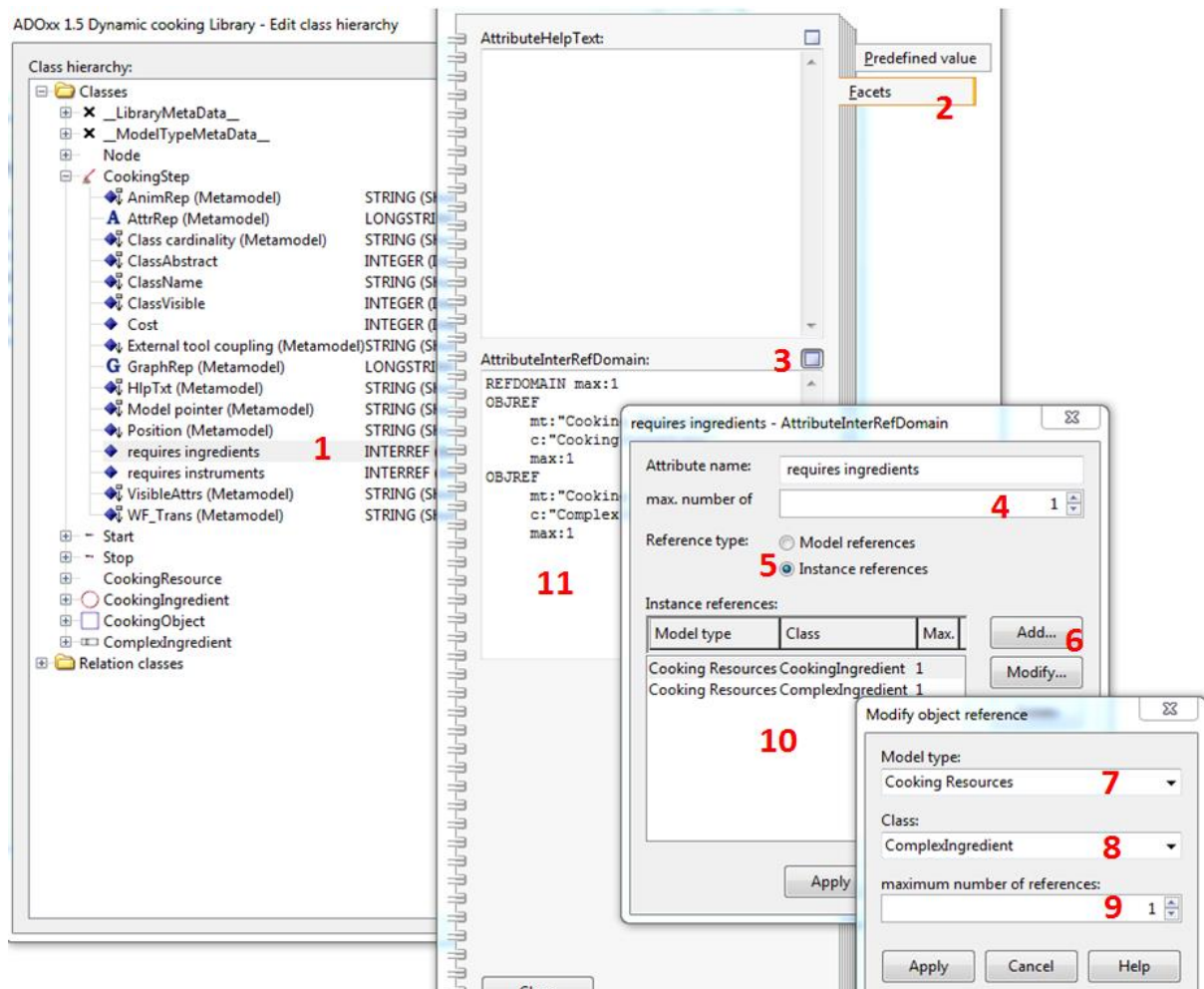
In the following, we will link the two types of models that we have created. A cooking step should provide hyperlinks to:

- One ingredient of any type (simple or complex), which is necessary in the cooking step.
- One cooking object required to execute the step.

The limit of one was chosen for simplification. It is possible to define any number of targets for a hyperlink but if there are more, the hyperlink will not jump directly to the target, but it will first display a list of all the possible targets, to choose from. Later we will extend the example to also provide in the cooking step multiple hyperlinks to multiple ingredients. You also have to consider that, if many ingredients are necessary in a step, it might be that the model is not detailed enough! If you need to create too many hyperlinks in the same element, probably you need to decompose your step in smaller steps!

To create these hyperlinks, select the CookingStep concept, then New attribute and define both attributes ("requires ingredients", "requires instruments") as having the type INTERREF. The next figure describes how to configure the first of them:

1. Select "requires ingredients"
2. Go to Facets
3. Click the details button for AttributeInterRefDomain
4. Indicate the maximum number of targets for the link (1)
5. Indicate if the link target will be an element of a model (instance), or a model as a whole
6. Press Add to define a possible target. A new window will pop-up
7. Select the model type where the targets will belong (Cooking Resources)
8. Select the target concept (CookingIngredient)
9. Indicate how many cooking ingredients can be targeted (1)
10. Press Apply and see the possible targets accumulated in a table. Repeat from step 6 to also add ComplexIngredient as a possible target, also with a maximum of 1 (however, note that even if different target concepts are allowed, only one instance of them will be linked for each hyperlink, because of Step 4)
11. You can also type some code to achieve the same result, which can be seen in area (11)



Do the same for “requires instruments”, but this one should target ONLY the CookingObject concept (also with a maximum of 1 targets).

Make both hyperlinks visible by including them in the AttrRep definition for CookingStep:

NOTEBOOK

CHAPTER "Description"

ATTR "Name"

ATTR "Cost"

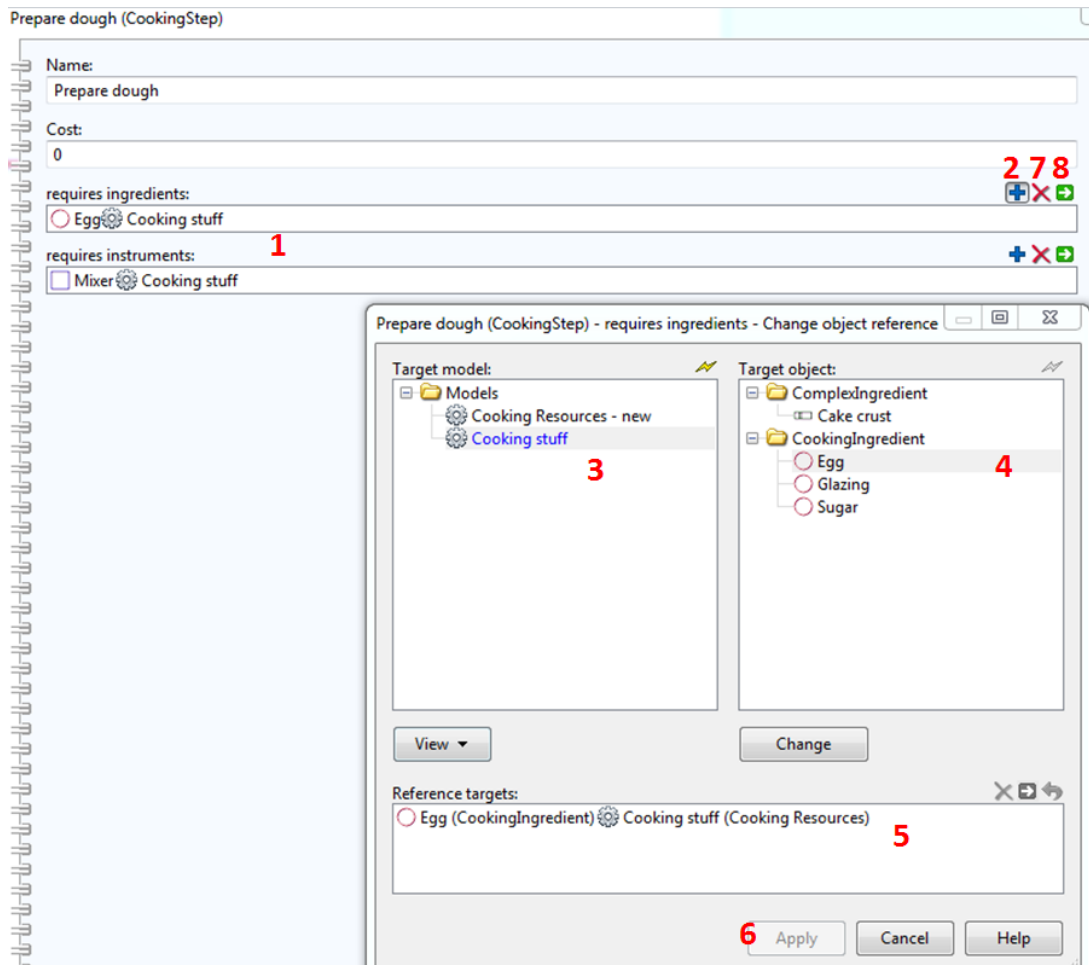
ATTR "requires ingredients"

ATTR "requires instruments"

Now go back to the modelling toolkit and double-click on a CookingStep:

1. Notice the new attributes. Each of them has some buttons on the right-end of the property slot.
2. The + sign opens the window where you can set the target of the hyperlink.
3. First you select the target model (if it does not show up in the list, you might have to open it first; depending on what you select in the View button, it's possible that it only displays open models)
4. From the target model, you select the target object. Notice that for “requires ingredients” it's possible to link either a CookingIngredient or a ComplexIngredient
5. After selecting the target, it is displayed in area (5)
6. Press Apply
7. If you later want to remove the hyperlink, use the icon (7)
8. If you want to navigate the hyperlink, use the icon (8); it will take you to the target.

Do the same for the “requires instruments” hyperlink. This one should link a step to a CookingObject.



The hyperlinks are more useful if the user can click directly on the graphical symbol in order to navigate to the target model. For this, the GraphRep must be extended with a HOTSPOT element. Replace the GraphRep of the Cooking Step with the following code (or, instead of replacing, comment away the old code):

GRAPHREP

FILL color:royalblue

RECTANGLE x:-1.5cm y:-1cm w:3cm h:2cm

FONT "Arial" h:14pt color:red

ATTR "Name" y:1.5cm w:c

AVAL ing:"requires ingredients"

IF (LEN ing)

FILL color:lightblue

ELLIPSE x:1.5cm y:-1cm rx:0.35cm ry:0.25cm

HOTSPOT "requires ingredients" x:1.25cm y:-1.35cm w:0.7cm h:0.5cm text:"jump to ingredient"

ENDIF

AVAL tool:"requires instruments"

IF (LEN tool)

FILL color:green

ELLIPSE x:-1.5cm y:-1cm rx:0.35cm ry:0.25cm

HOTSPOT "requires instruments" x:-1.75cm y:-1.35cm w:0.7cm h:0.5cm text:"jump to tool"

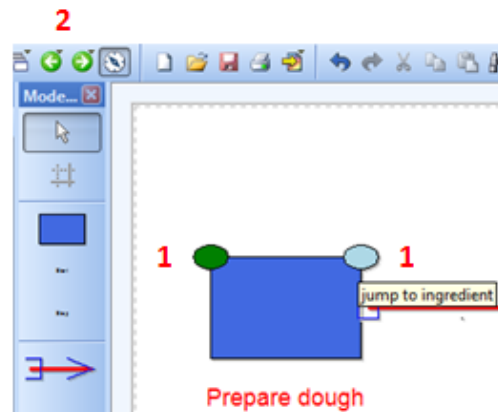
ENDIF

Notice that the GraphRep has 3 parts:

- First, the basic symbol is defined, as a blue rectangle with the Name displayed below.
- Then, the “requires ingredients” attribute is read. We test if it is empty (LEN is the string length, so the test checks if the length is different than zero). If it’s not empty, a small ellipse is drawn in the top-right corner of the rectangle, and a HOTSPOT is defined on top of the ellipse surface (the hotspot is a rectangle, so the overlapping is not 100% perfect; a hint text is also attached to suggest to the user where the hotspot will take him). We could extend the HOTSPOT graphics in any way allowed by the GraphRep syntax, for example to display the name of the link target in the hotspot.
- Do the same for “requires instruments”, but define the ellipse and its hotspot in the top-left corner.

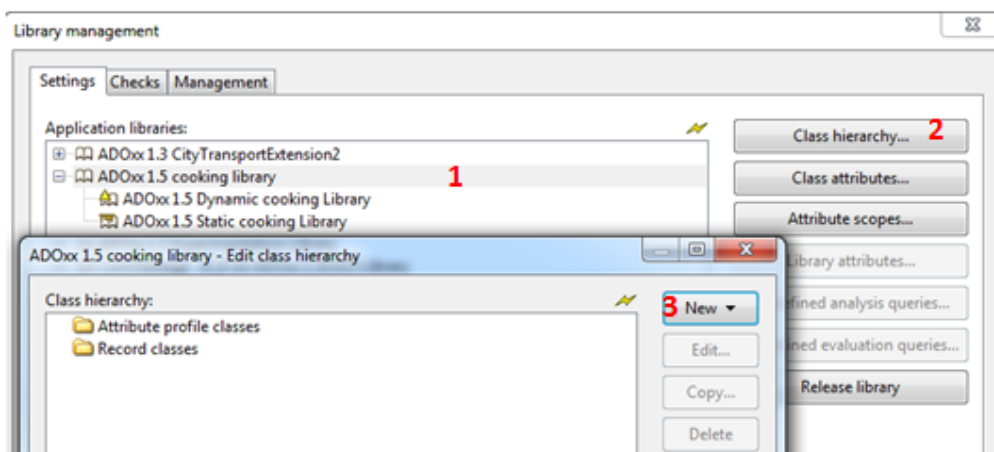
Go to the modelling toolkit. Notice in the next figure:

1. The two hotspots and the hint text that appears when the cursor stops on the hotspot. Click on a hotspot to jump to its target.
2. Notice the back/forward buttons that allow us to navigate between models.



In the following we will create a complex attribute which will look like a table with several fields. To include tabular attributes, we first need to create a special type of class (record class) where we define the table structure. This type of classes cannot be created in the concept hierarchy, they must be defined on library level.

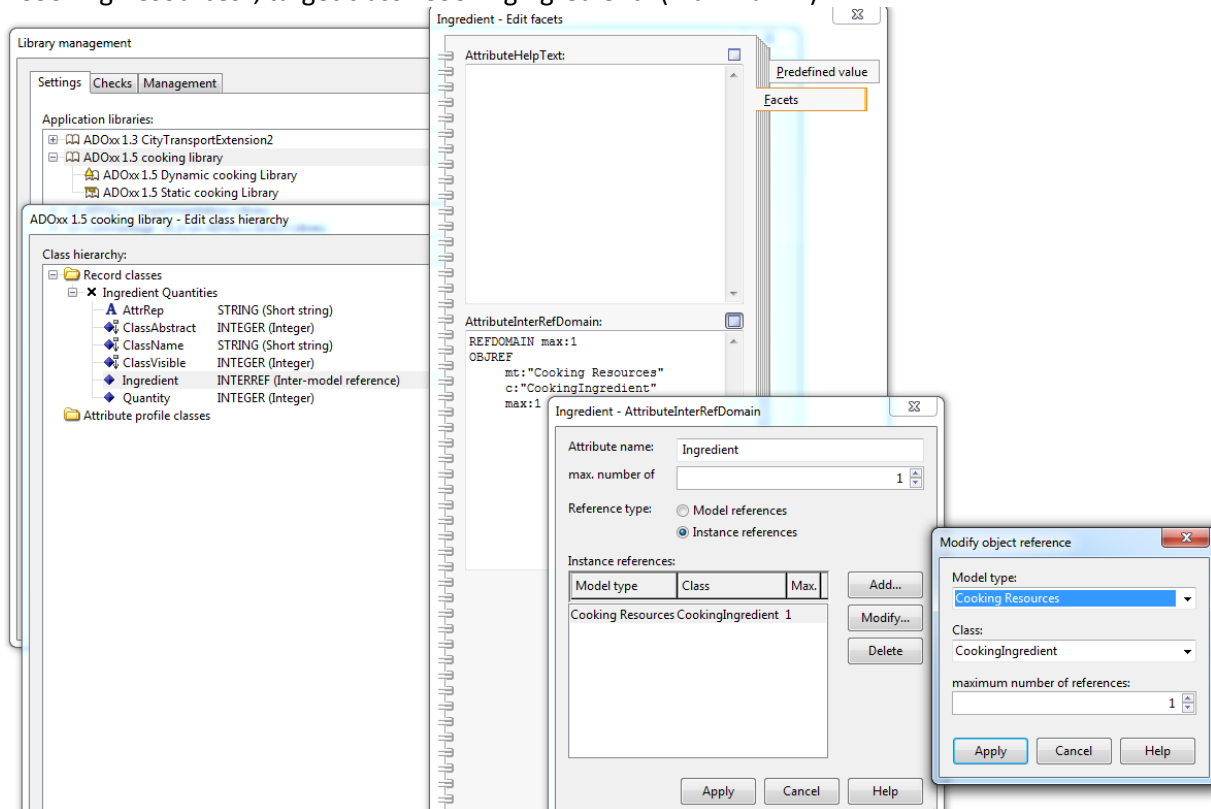
1. Select the library (not only its “Dynamic” part!)
2. Open its Class hierarchy. Notice that at library level we don’t really have a concept hierarchy, only two types of classes.
3. Press New and select Record class. Give it the name “Ingredient Quantities”



A record class defines the structure of a table attribute. Therefore all the fields of the table must be defined as attributes of the record class. Right-click on “Ingredient Quantities” and create two attributes:

- Ingredient, of type INTERREF (notice that hyperlinks can be included in tables!)
- Quantity, of type INTEGER

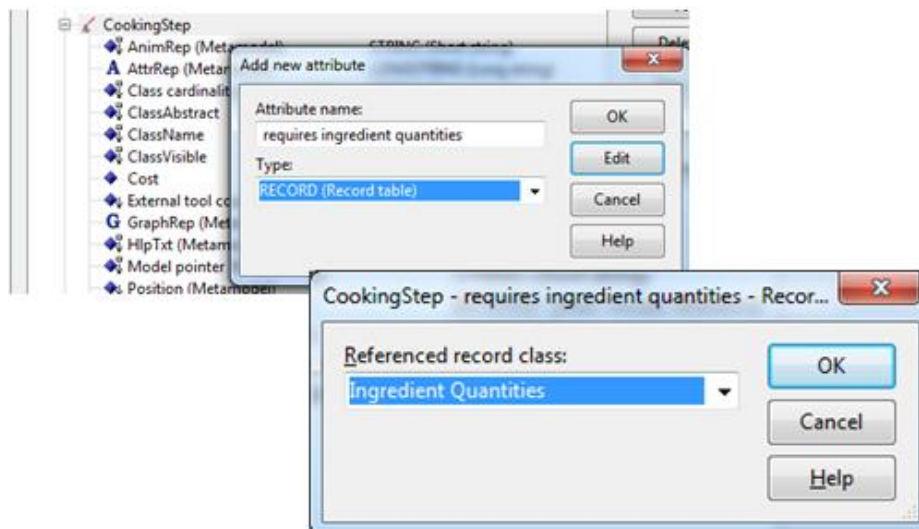
For Ingredient, define the link target: maximum 1 target, Instance reference, target Model type “Cooking Resources”, target class “CookingIngredient” (maximum 1).



Just like with regular concepts, you have to include the attributes in the AttrRep of the record class, otherwise they will not be visible in the table. Write the following code in Ingredient Quantities’ AttrRep:

```
NOTEBOOK
CHAPTER "Description"
ATTR "Ingredient"
ATTR "Quantity"
```

Next, you have to include this type of table in the property set for CookingStep. Go back to the concept hierarchy, right click on CookingStep, select New attribute. Call the new attribute “requires ingredient quantities” and declare it of type RECORD. This will further open a new window where you can declare the table structure that will be used – i.e., the record class you’ve just created (“Ingredient Quantities”)



Finally, you need to make the table visible in the CookingStep property set, by including it in its AttrRep:

NOTEBOOK

CHAPTER "Description"

ATTR "Name"

ATTR "Cost"

ATTR "requires ingredients"

ATTR "requires instruments"

ATTR "requires ingredient quantities"

Pay attention to the difference between the attribute "requires ingredient quantities" and the class "Ingredient Quantities"! The attribute belongs to the CookingStep concept, and it takes its table structure from "Ingredient Quantities". However, the same table structure could be reused by other attributes, in other concepts! "Ingredient Quantities" is not an attribute, it's just a reusable group of attributes presented as "fields" in table! Any attribute of type RECORD could reuse this table structure. In other words, **record classes are an alternative to inheritance** – i.e., they allow the reuse of some properties without inheriting them from some parent class from the concept hierarchy (this is the reason why they are created outside the concept hierarchy!).

Now go to the modelling toolkit and double click a CookingStep. You will notice the table attribute and its first column allows you to create hyperlinks to cooking ingredients from other models (a "measurement unit" would also be necessary, but we keep the example simple. Notice the following:

- Now you can assign more than one ingredient to a cooking step, and also a quantity for that specific ingredient, as used in that specific step! This means that we actually established a m-to-n relation between cooking steps and ingredients, and this relation has its own attribute (the quantity)
- The ingredient name has too much information in it – it also includes its class, the name of the target model and the type of the target model. We will reduce this to display only the ingredient name. Another change that we will make is to display the entire table in the graphic notation, to make the hyperlinks available for direct clicks on the diagram elements.

Prepare dough (CookingStep)

Name:
Prepare dough

Cost:
0

requires ingredients: + X →
☐ Egg Cooking stuff

requires instruments: + X →
☐ Mixer Cooking stuff

requires ingredient quantities: + X □

	Ingredient	Quantity
1	Egg (CookingIngredient) - Cooking stuff (Cooking Resources)	3
2	Sugar (CookingIngredient) - Cooking stuff (Cooking Resources)	1

Warning: you cannot create hyperlinks (INTERREF) or table attributes (RECORD) in a relation class (connector)

Go back to the Development toolkit. First let's make sure that the ingredient table displays strictly the ingredient name in the first column. For this, go to your record class and modify the AttrRep of the "Ingredient Quantities" table definition:

NOTEBOOK

CHAPTER "Description"

ATTR "Ingredient" **format:"%o"**

ATTR "Quantity"

The format parameter controls how much of the target of a hyperlink is displayed. Several codes can be used: %o is for the object name, %c is for the concept name, %m is for the model name, %t is for the model type name.

Now go to the concept hierarchy and change the GraphRep of the CookingStep as follows:

GRAPHREP

FILL color:royalblue

RECTANGLE x:-1.5cm y:-1cm w:3cm h:2cm

FONT "Arial" h:14pt color:red

ATTR "Name" y:1.5cm w:c

AVAL set-count-rows rowcount:"requires ingredient quantities"

IF (rowcount>0)

FONT h:9pt

TEXT "INGR." x:1.6cm y:-0.5cm

TEXT "QUANT." x:3cm y:-0.5cm

FOR i from:1 to:(rowcount)

{

ATTR "requires ingredient quantities" row:(i) col:"Ingredient" format:"%o" x:1.6cm y:(CM (i-1)/2)

ATTR "requires ingredient quantities" row:(i) col:"Quantity" format:"%o" x:3cm y:(CM (i-1)/2)

LINE x1:1.6cm y1:(CM (i-1)/2) x2:4cm y2:(CM (i-1)/2)

```

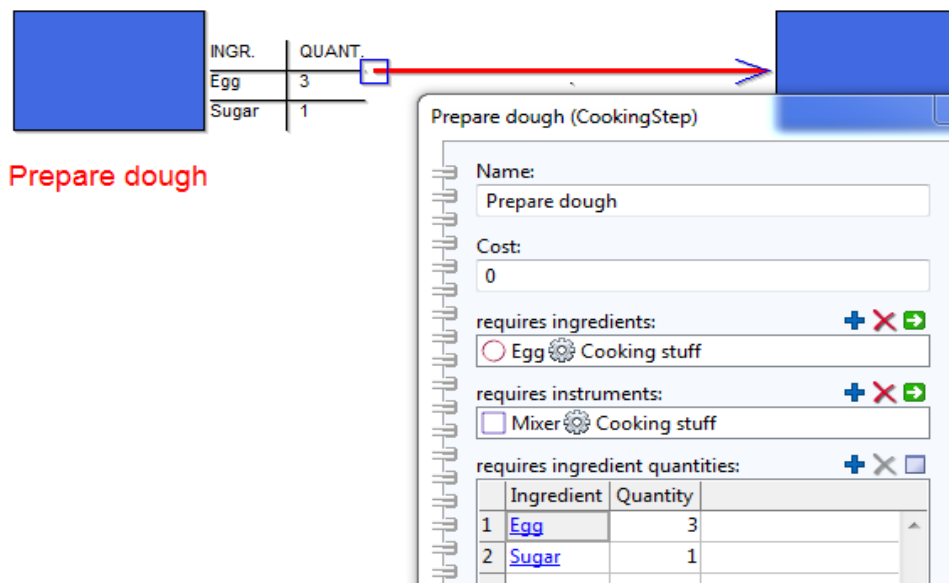
}
LINE x1:2.8cm y1:-0.5cm x2:2.8cm y2:(CM rowcount/2)
ENDIF

```

Notice:

- We kept the previous blue rectangle, but removed the hotspots
- AVAL now reads the number of rows in “requires ingredient quantities” and stores it in the rowcount internal variable
- If this rowcount is higher than zero, it builds a table with the following elements:
 - A table header, with static labels (the TEXT commands) properly positioned on the right side of the blue rectangle
 - A FOR loop generates a table row with two values collected from the columns Ingredient and Quantity (from row(i), the current row of the table). Again the format parameter will make sure that only the link target name is displayed here. The position is aligned with the table header, but the y axis coordinate is proportional with the row number, to make sure that each row is displayed below the previous one. A simple line is also drawn under the values, to delimit the rows
 - After the FOR loop, a vertical line is drawn between the two columns

Go back to the modelling toolkit and notice that now the table is also reflected in the notation and that each ingredient name acts as a hyperlink not only in the property window, but also directly in the notation. Also notice that if the table is empty, it will not be created at all in the notation (that’s why we included it in an IF structure).



In the following, we will create a calculated attribute, whose values will be read-only because they will be computed based on some other information available in models.

Go to the Development Toolkit and create for CookingStep a new attribute named “Calculated”, of type EXPRESSION. In the Standard value box for this attribute write:

```
EXPR expr:fixed:(aval("Cost")*2)
```

With expr:fixed, the expression will be fixed here in the Development toolkit and its values will change according to the calculation formula. Here the formula doubles the value of the Cost attribute (accessed with the aval() function).

An expression that is not fixed will generate a special attribute that acts like an “expression builder”, allowing the modeller to build any formula directly in the model, by using common operators and a set of predefined functions provided by ADOxx. *Look in the Help for “Expressions” to obtain the list of functions.*

Create another attribute in CookingStep, named “Free formula”, also of type EXPRESSION. This type in the Standard value write only:

EXPR

This will leave the formula completely unconstrained (a parameter type could constrain the value type returned by such formulas). With both expression attributes created, make them visible in the AttrRep of CookingStep:

NOTEBOOK

CHAPTER "Description"

ATTR "Name"

ATTR "Cost"

ATTR "requires ingredients"

ATTR "requires instruments"

ATTR "requires ingredient quantities"

ATTR "Calculated"

ATTR “Free formula”

Back in the modelling toolkit, notice:

- The read-only Calculated property slot, which displays the double of the Cost attribute
- The Free formula property slot, which has an fx button at the right-end. This provides an expression editor where ADOxx functions and operators may be used to freely build a computation. We used the same formula as before, but this time it’s a formula defined by the modeller, therefore it does not have well-defined semantics (the name “Free formula” also suggests the generic character of this expression). For free formulae, the modeller should consult the Help to become familiar with all the operators and functions that can be used.

Prepare dough (CookingStep)

Name: Prepare dough

Cost: 2

requires ingredients: ☐ Egg ☒ Cooking stuff

requires instruments: ☐ Mixer ☒ Cooking stuff

requires ingredient quantities:

	Ingredient	Quantity
1	Egg	3
2	Sugar	1

Calculated: 4

Free formula: 4

Free formula - Prepare dough (CookingStep)

☒ Expression: ☐ String constant:

aval("Cost") * 2

Apply Default value Cancel Help

Go to the Development toolkit and replace the EXPR value of “Calculated” with the following

EXPR expr:fixed:(rasum("requires ingredient quantities","Quantity"))

The rasum() function calculates the total of a column (“Quantity”) from a table attribute (“requires ingredient quantities”). Go to the modelling toolkit and check.

requires ingredient quantities:

	Ingredient	Quantity
1	Egg	3
2	Sugar	10

Calculated: 13

Go to the Development toolkit and replace the EXPR value of “Calculated” with the following

EXPR expr:fixed:(rcount("requires ingredient quantities"))

The rcount() function calculates the number of rows in a table attribute. Go to the modelling toolkit and check.

requires ingredient quantities:

	Ingredient	Quantity
1	Egg	3
2	Sugar	10

Calculated:
2

Go to the Development toolkit and replace the EXPR value of “Calculated” with the following

EXPR expr:fixed:(asum(allobjs(modelid,"CookingStep"),"Cost"))

The asum() function calculates the *sum of the values for an attribute (“Cost”) from all instances of a certain concept (“CookingStep) from a model*. Notice that it relies on the function allobjs() which returns an array of *all elements of a certain type from a specified model*. The current model can be specified with the built-in constant *modelid*. Other similar constants are available: *objid* is the current model element (including connectors); *classid* the current concept/class; *attrid* is the current attribute; *val* is the current value of the current expression. For a complete list of constants and functions that can be used, look in the Help for “Expressions”.

In the modelling toolkit, you can specify multiple costs for multiple cooking steps, then you can notice the calculated sum over the entire model. Of course, such an aggregate property does not make much sense to be stored in each element – typically it is stored either in the Start concept, or as a model-level attribute!



Prepare dough (CookingStep)	Prepare Glazing (CookingStep)												
Name: Prepare dough	Name: Prepare Glazing												
Cost: 8	Cost: 10												
requires ingredients: <input type="radio"/> Egg <input checked="" type="radio"/> Cooking stuff	requires ingredients:												
requires instruments: <input type="checkbox"/> Mixer <input checked="" type="checkbox"/> Cooking stuff	requires instruments:												
requires ingredient quantities:	requires ingredient quantities:												
<table border="1"> <thead> <tr> <th>Ingredient</th> <th>Quantity</th> </tr> </thead> <tbody> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Ingredient	Quantity					<table border="1"> <thead> <tr> <th>Ingredient</th> <th>Quantity</th> </tr> </thead> <tbody> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Ingredient	Quantity				
Ingredient	Quantity												
Ingredient	Quantity												
Calculated: 18	Calculated: 18												

In the following, we will extend the CookingStep concept with a new type of hyperlink, which can open either a file from the disk, or an executable program, or both (a specific file opened with a specific program).

Go to the Development toolkit and create a new attributes for CookingStep, “run program or file”, of type PROGRAMCALL. This link will be used to attach to each cooking step some relevant documentation – e.g., PDF files, movies describing the cooking process or even a web resource that can be accessed through a URL.

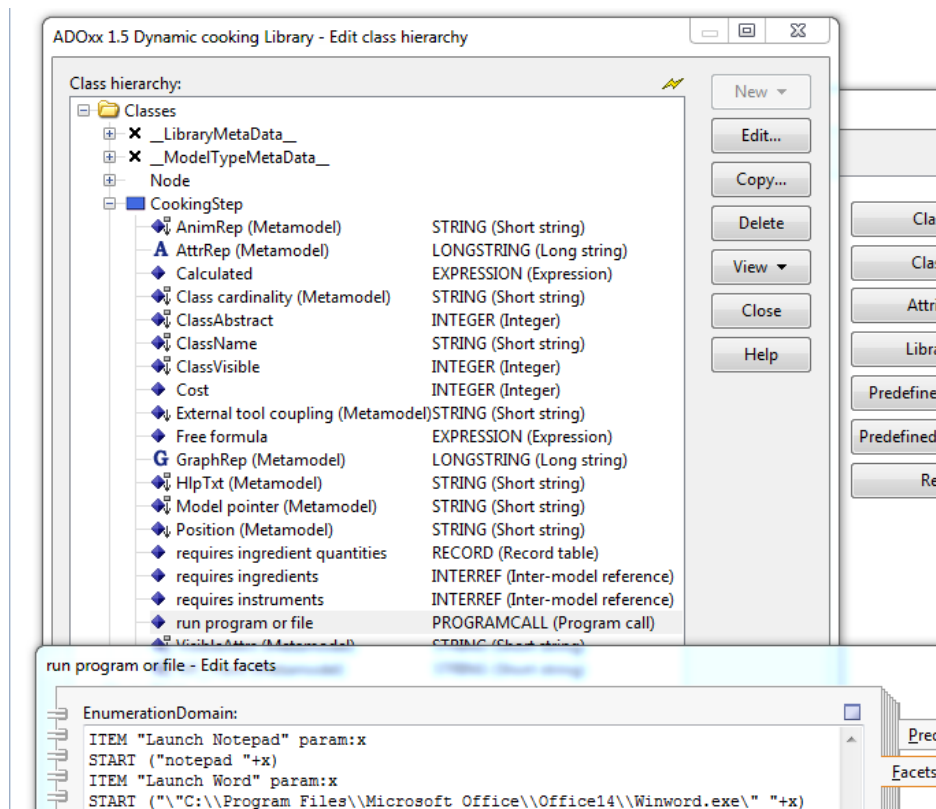
After the attribute is created, its EnumerationDomain must be defined:

ITEM "Launch Notepad" param:x

START ("notepad "+x)

ITEM "Launch Word" param:x

START ("\"C:\\Program Files\\Microsoft Office\\Office14\\Winword.exe\" "+x)



This piece of code will generate a drop-down menu of programs that can be launched from a CookingStep:

- The menu item “Launch Notepad” will run Notepad. It accepts a file as argument (the file will be selected through a dedicated file selection slot)
- The menu item “Launch Word” will run Word, and it also accepts a file as argument.
- In addition to the two programs, a default menu item will also be included (labelled “automatically”). This will let the system to decide what program should be run, based on the type of the selected file.

Notice that each menu item has a START command attached to it, which is actually the command to be passed to the Windows command line. For Notepad, the direct name should be enough to launch it (since it is included in the PATH environment variable). For Word however, if we don’t want to also

include it in PATH, we need to specify its full file path, with the following adjustment: backslashes and quotation marks must be escaped, to avoid conflict with the ADOxx string delimiters.

Next, you must add the new attribute to the AttrRep. Since we already have many attributes, we define a new chapter (a new page in the property sheet):

NOTEBOOK

CHAPTER "Description"

ATTR "Name"

ATTR "Cost"

ATTR "requires ingredients"

ATTR "requires instruments"

ATTR "requires ingredient quantities"

ATTR "Calculated"

ATTR "Free formula"

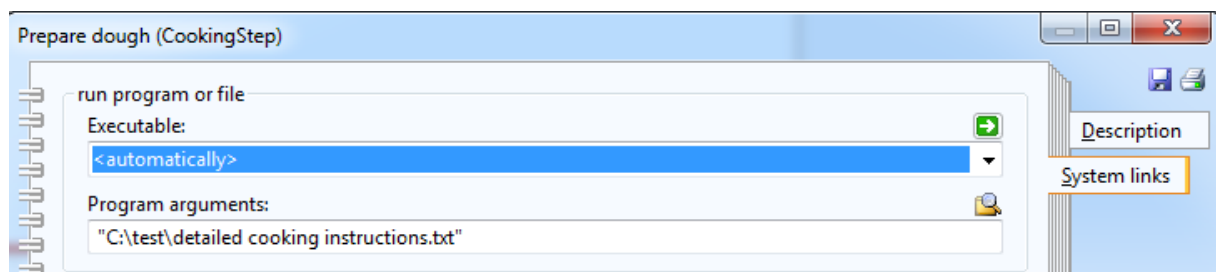
CHAPTER "System links"

ATTR "run program or file"

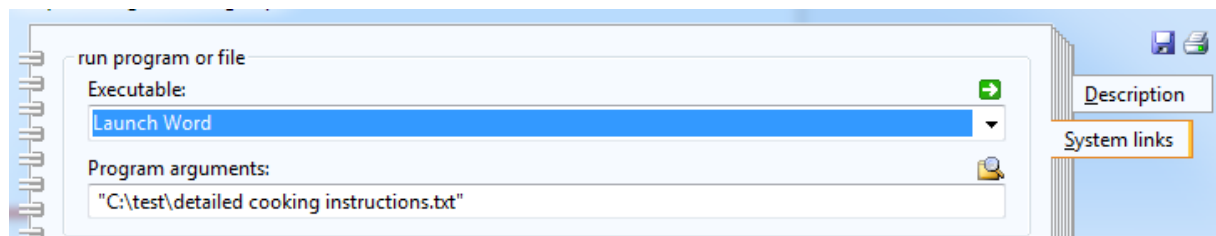
Go to the modelling toolkit, find the new attribute in the System links chapter of a CookingStep. Notice the two parts of the attribute: a drop-down list ("Executable") and a file selection slot ("Program arguments"). The green arrow button at the right-end will launch the selected program/file.

Several combinations are possible, illustrated in the following screenshots:

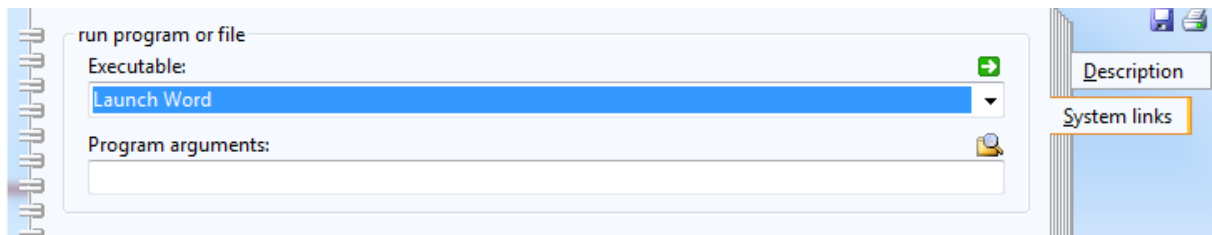
Leave the drop-down on its default value ("automatically") and open a file. The execution button will open the file with whatever program is defined in Windows for that type of file.



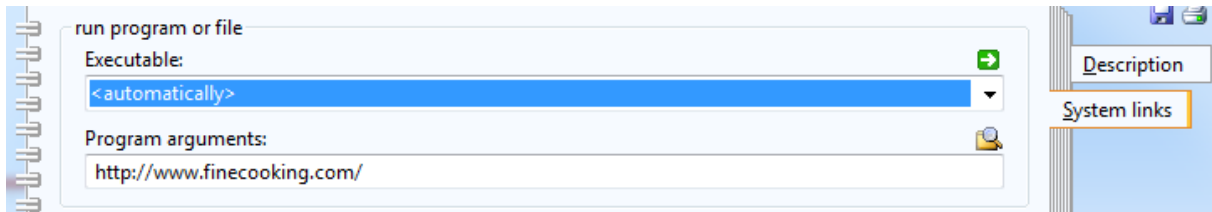
Select a program from the drop-down list and select a file. The execution button will try to open the file with the selected program (according to the START command defined in the Dev. Toolkit for that program).



Select a program from the drop-down list and leave the file slot empty. The program will be launched without any program.



Leave the drop-down on its default value and write a URL in the file slot. The default browser will be launched and will take you to that URL! This means that you can link relevant websites, movies etc. to any model element.



Remember that hyperlinks should be accessible with a direct click on the graphical symbol. Therefore we will extend the GraphRep of CookingStep to include a hotspot for the program call link:

GRAPHREP

FILL color:royalblue

RECTANGLE x:-1.5cm y:-1cm w:3cm h:2cm

FONT "Arial" h:14pt color:red

ATTR "Name" y:1.5cm w:c

AVAL pcall:"run program or file"

SET length:(LEN pcall)

SET pos:(search(pcall,"@",0))

IF ((pos) OR (length-pos-1))

 FILL color:white

 ELLIPSE x:1.5cm y:1cm rx:0.35cm ry:0.25cm

 HOTSPOT "run program or file" x:1.25cm y:0.65cm w:0.8cm h:0.4cm text:"launch application"

ENDIF

(you may keep the rest of the code for the ingredient quantities table)

Notice the two variables, pos and length. They are not attribute values, therefore they are not initialized with AVAL, but with SET. In order to understand the logic, you need to understand how the value of the "run program or file" attribute looks (you can check it by including an ATTR in the GraphRep to show you the full value of the attribute):

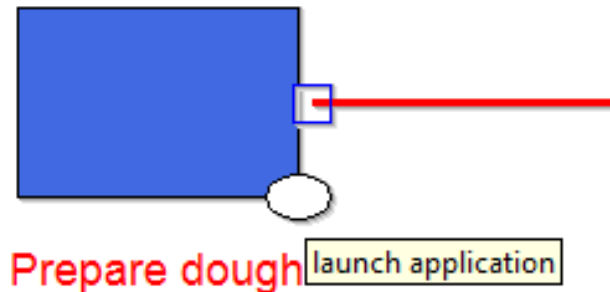
The value of a PROGRAMCALL attribute value has the form x@y, where x is the ITEM label selected in the drop-down list and y is the file path string included in the file selection slot. If the default („automatically") is selected in the drop-down list, x will be an empty string. If quotation marks appear in the file path (for example when the file path includes spaces), then they are also part of the y string.

Therefore the processing of the variables is as following:

- length will store the number of characters for the attribute as a whole

- pos will store the position of the @ delimiter in the full value

The hotspot will be shown only if pos is different than zero (i.e., if there is at least one additional character in front of @) or if length-pos-1 is different than zero (i.e., if there is at least one character after @). The hotspot is an ellipse in the bottom-right corner of the rectangle, which will show a hint text and will allow the user to launch the program (or to visit a Web link) by a direct click on it.



In the following, we will move to the `CookingObject` concept in the second type of diagram. Currently, its symbol is a simple blue rectangle. We will allow the modeller to select any picture he wants as a graphical symbol, also with the possibility of resizing it conveniently. This is different from what we did in the beginning, when we imposed a specific graphics file for all `CookingSteps`. Here, each element's picture will be customized by the modeller.

Go to the Development toolkit. First, create in the `CookingObject` an attribute „Preferred picture“ of type `PROGRAMCALL`. Then, make it visible in `AttrRep`:

```
NOTEBOOK
CHAPTER "Description"
ATTR "Name"
ATTR "Type" ctrltype:dropdown
AVAL t:"Type"
ATTR "Equipment usage" enabled:(t="Equipment")
ATTR "Preferred picture"
```

Replace the `GraphRep` for the `CookingObject` with the following:

```
GRAPHREP sizing:asymmetrical

AVAL picattr:"Preferred picture"

SET found:(search(picattr,"\\",0))
IF (found>=0)
  SET picpath:(copy(picattr,2,(LEN picattr)-3))
ELSE
  SET picpath:(copy(picattr,1,(LEN picattr)-1))
ENDIF

IF (LEN picpath)
  BITMAP (picpath) x:-1cm y:-1cm w:2cm h:2cm
ELSE
  PEN color:blue
  RECTANGLE x:-1cm y:-1cm w:2cm h:2cm
ENDIF
```

ATTR "Name" y:1cm w:c

Notice the sizing parameter on the first line. This will allow the modeller to resize the symbol conveniently, to avoid the distortion of images.

Then, the "Preferred picture" attribute is stored in picattr. This attribute will allow the modeller to select any file from the disk, as shown before. The program selection drop-down list will remain unused, only the file path matters, since it is used by the BITMAP command to display the selected picture file. However, BITMAP needs strictly the full path so we need to clean it up. Remember that the value of a PROGRAMCALL attribute includes:

- the delimiter @ between the program name and the file path; since the program name will not be used, the character will be the first in the string value
- quotation marks, if the selected file path has spaces in it

First, we check with search() if a quotation mark (escaped with \) appears in the attribute value. The function returns the position, or -1 if not found.

- If found (any value ≥ 0) then the full path can be extracted starting with position 2 (skipping the @ and the opening quotation). The number of extracted characters will be the length minus 3 (the 2 quotation marks and @)
- If not found, then the full path starts at position 1 and the number of extracted characters is length minus 1

Then, if the path is not empty, the BITMAP command is executed with some initial sizing. If the path is empty, the old rectangle will be displayed.

Go to the modelling toolkit and try all possibilities: open a graphics file whose path has spaces (hence quotation marks), one without spaces or leave the attribute empty to see only the default rectangle.

We mentioned earlier that table attributes (of type RECORD) provide an alternative to inheritance, by allowing the reuse of some properties defined outside the concept hierarchy.

Another way of doing this are attribute profiles (attributes of type PROFREF). These are also reusable groups of attributes, with the following differences:

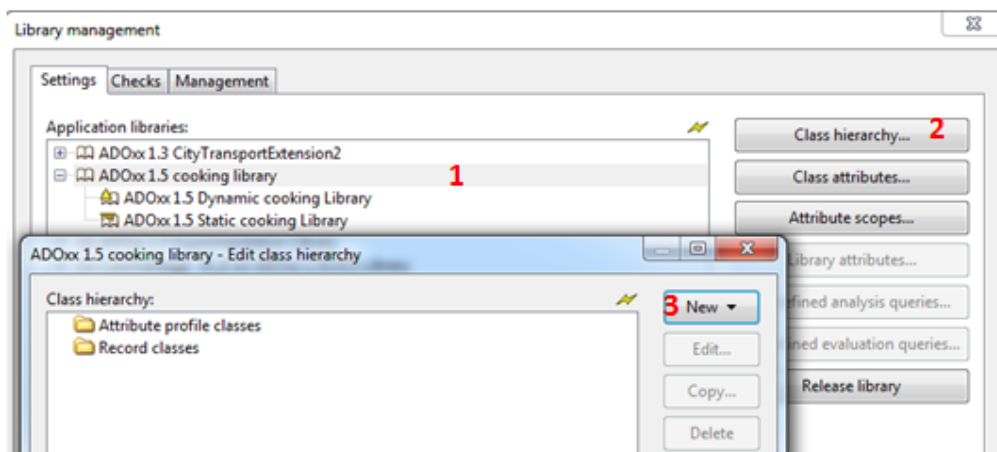
- they are not presented as tables, but as groups of properties included in the property sheet; this means that we can also include tables in an attribute profile, whereas it is not possible to have a table inside a table;
- not only the attributes are reused, but also their values; the modeller will be able to define a combination of values for all the attributes in the profile and to reuse it conveniently.

In the following we will create a pair of attributes to be used together in all ingredient elements – both in CookingIngredients and ComplexIngredients (swimlanes). The pair will be formed of:

- the attribute "usage", providing a selection list to indicate if an ingredient can be used raw or should always be processed.
- the attribute "keep in the fridge", displayed as a checkbox, to indicate if an ingredient has some temperature requirements.

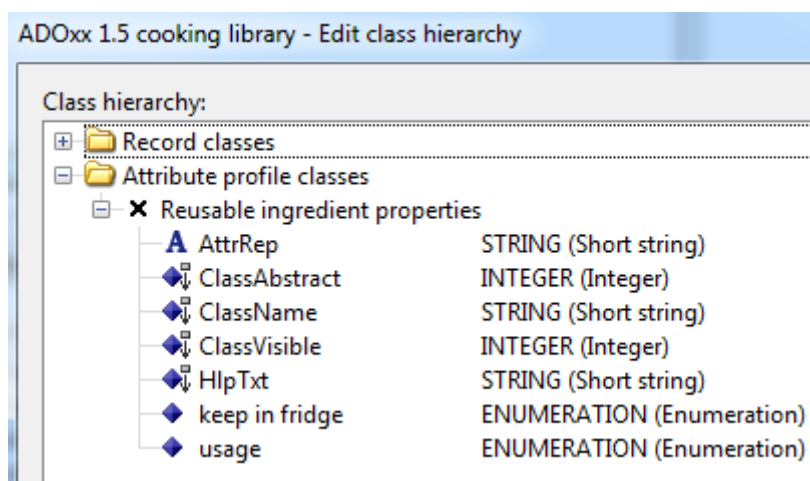
The modeller will be able to assign to different ingredients different combinations of values for these attributes.

In the Development toolkit, select the entire library before pressing Class hierarchy. Just like with the Record class before, an Attribute Profile does not belong to a specific concept, it can be reused anywhere throughout the library.



Select New – attribute profile class and give it the name "Reusable ingredient properties". Inside it

- Create the attribute "keep in fridge" of type ENUMERATION (checkboxes are also enumerations); define for it the options "yes" and "no"
- Create the attribute "usage" also of type ENUMERATION; define for it the options "can be used raw" and "must be processed"



Include in the AttrRep of the profile the two attributes, also specifying how they should be presented (the first will be default, as a radio button list; the second as a checkbox):

NOTEBOOK

CHAPTER "Description"

ATTR "usage"

ATTR "keep in fridge" ctrltype:check checked-value:"yes" unchecked-value:"no"

Go to the concept hierarchy, select CookingIngredient and create an attribute "handling properties" of type PROFREF. Just like with tables, it will ask you to select the class where you stored the structure – this is "Reusable ingredient properties". Make the new attribute visible:

NOTEBOOK

CHAPTER "Description"

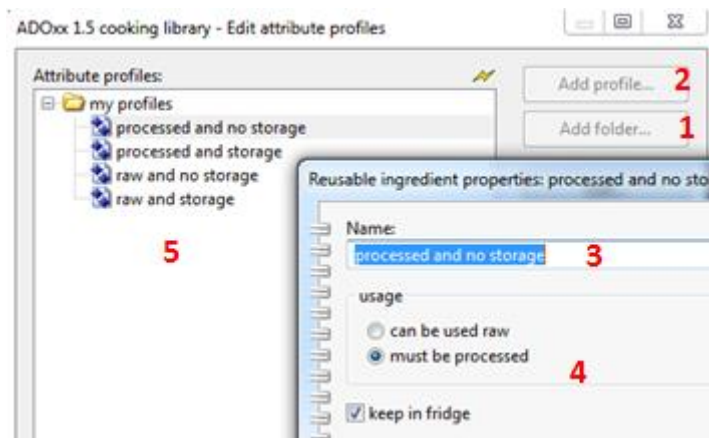
ATTR "Name"

ATTR "handling properties"

Do the same for ComplexIngredient. You may call the attribute differently, although you may also use the same name "handling properties". Declare the same type PROFREF, link it to the same structure "Reusable ingredient properties" and make it visible in the AttrRep of ComplexIngredient.

Go to the modelling toolkit and select Edit-Attribute profiles.

1. Use Add folder to define a folder to organize the profiles ("my profiles")
2. Use Add profile to create a new profile
3. Give the profile a Name
4. Define the combination of values to be saved under this name
5. The combination will be listed under the profile folder. Repeat these steps to define different combinations that you might want to reuse often (e.g., raw usage selected in combination with the checkbox active).



In a CookingRecipes diagram, you will find the "handling properties" attribute in both types of ingredients. They cannot be edited directly. Instead, use the + sign at the right-end to open the list of previously defined combinations of values ("profiles") and select the one you like to apply. Notice that you can do the same for the swimlanes (ComplexIngredients).

