integranova
Software Solutions

FILTERING AND SORTING INFORMATION

SAMPLE: SHOPPING CART

Last Update: Aug 2010

Table of Contents

1 Goal

In this tutorial we will continue walking through the development of a very simple application using Integranova. The tutorial covers the basic tools and steps involved in the development of applications from scratch as a way of introducing the reader to Integranova.

Following with the tutorial series, we will now expand the Tutorial 11, with this new tutorial (Tutorial 12) that will build on the previous one.

So far, we have learnt how to work with interaction units, the auxiliary patterns that can take part of them.

In this tutorial series we are going to work on Filters and Order criteria, both patterns briefly introduced in the Tutorial 9 of the series.

By the end of this tutorial, we will be able to define our own searching patterns by means of Filters, and also we will be able to define the order in which a population of instances will be shown to the application users.

2 Helping to search and filter data

In previous series, we saw how to define display sets and put them into different interaction units. This leads the application to show instances based on the information defined in these display sets. However, if the amount of these instances grows, it would be interesting to provide a way of filtering that information, wouldn't it?

In Integranova, as we briefly saw when introducing some interaction units, we can optionally use filter patterns as to achieve this filtering aim.

Let's see some examples that will illustrate this.

In 'Customer' class we have already defined a Population Interaction Unit whose aim is to show the customers in the system. We should remember that we defined two Population Interaction Units, one for Customers and one for Administrators. The one for Customers, as customers have Horizontal Visibility defined in order to only see themselves, does not need any filter. However, for the administrator's case, we could define a filter for this population interaction unit in order to allow the administrators to search customers by their names. So let's open the model and start modeling this.

We open the 'Tutorial_11_Final.oom' model file and go to the Presentation Model of the 'Customer' class. We select there the 'PIU_CustomersForAdmin' Population Interaction Unit.

Initially this Population Interaction Unit only has defined its display set (which as we already know is mandatory). See the Figure 1:

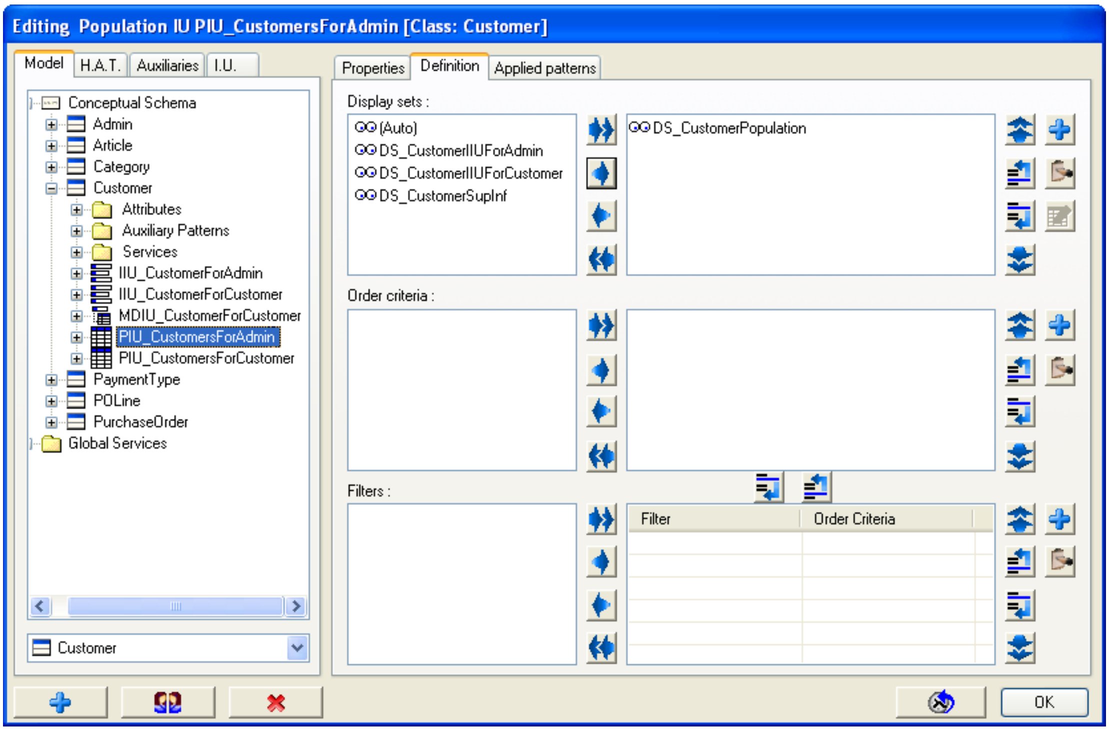
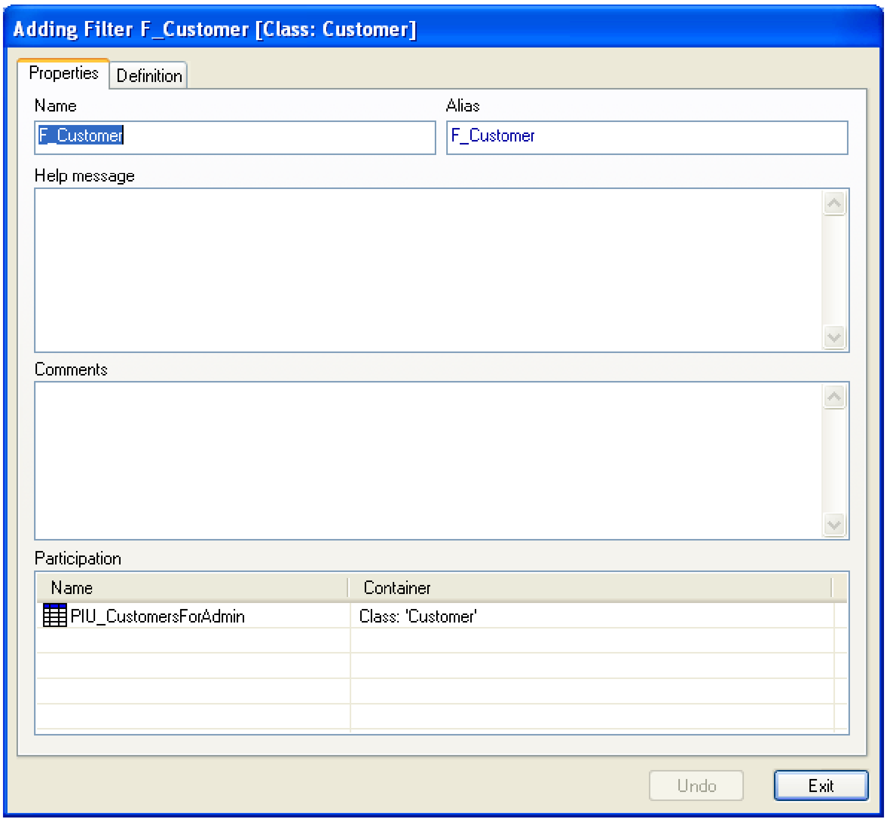


Figure 1 PIU_CustomersForAdmin Definition

As we can see, there are no filters already created. That's exactly what we are going to do now. So we look at the bottom right side area and we click on the *Add* + button as to define our first filter:



Name: F_Customer

Alias: F_Customer

Participation

Name	Container
PIU_CustomersForAdmin	Class: 'Customer'

Figure 2 *Properties* tab of Filter dialog

As you can see in Figure 2, *Properties* tab shows the properties of the filter. A *Name* and an *Alias* are offered by default. We can change these values anytime so we will leave them as they are. We can introduce a *Help message* as well and we can also see there is a *Participation* box, at the bottom of the window, which indicates the interaction unit/s in which this filter is taking part.

However, the clue of the filter definition is in the *Definition* tab. Let's click on it:

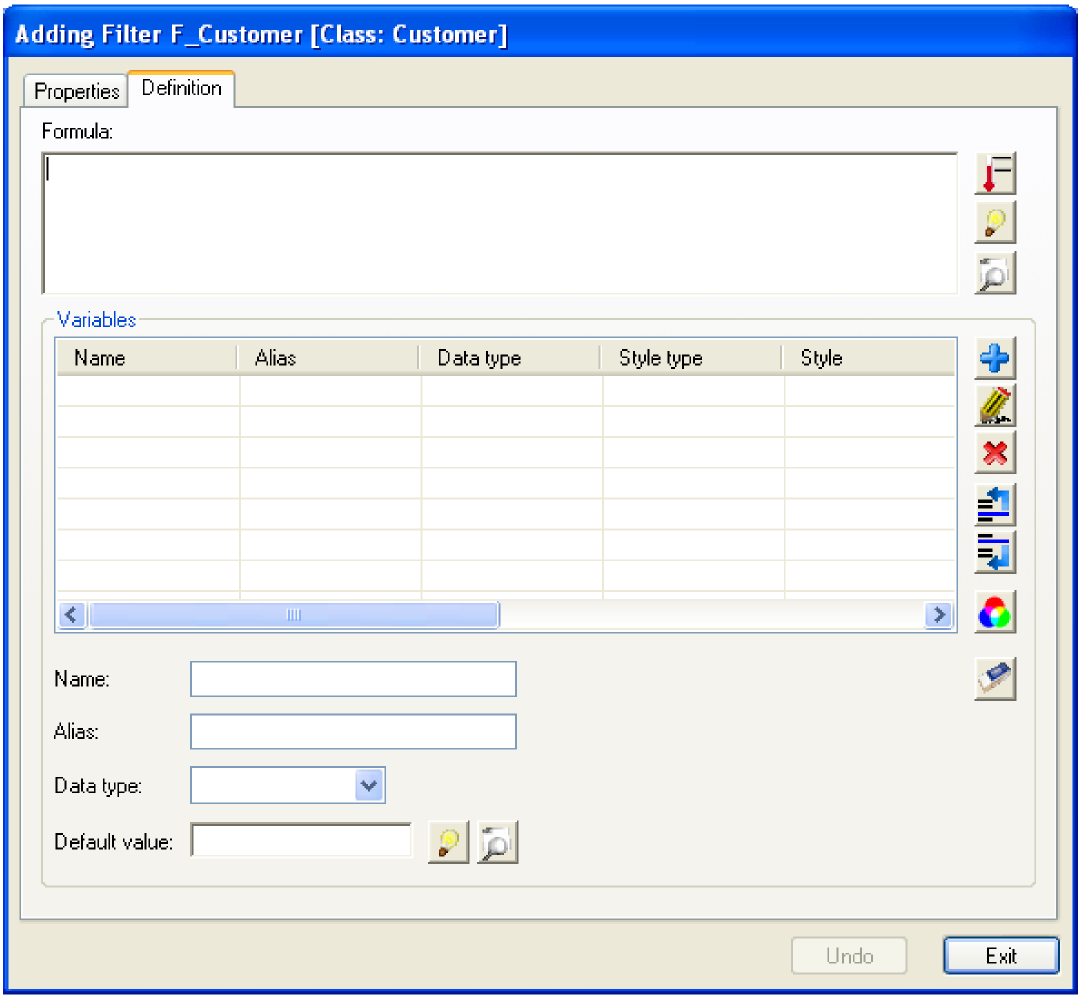


Figure 3 Defining a filter

In the *Definition* tab of a filter, starting from the top, we can see we have a textbox area in which to define a **filter formula**. A filter formula is a Boolean well defined formula that specifies the condition that every instance must fulfill in order to be retrieved. Secondly, we have an area in which we can define filter variables. These filter variables are the values from which we would like to filter, and are requested to the application user.

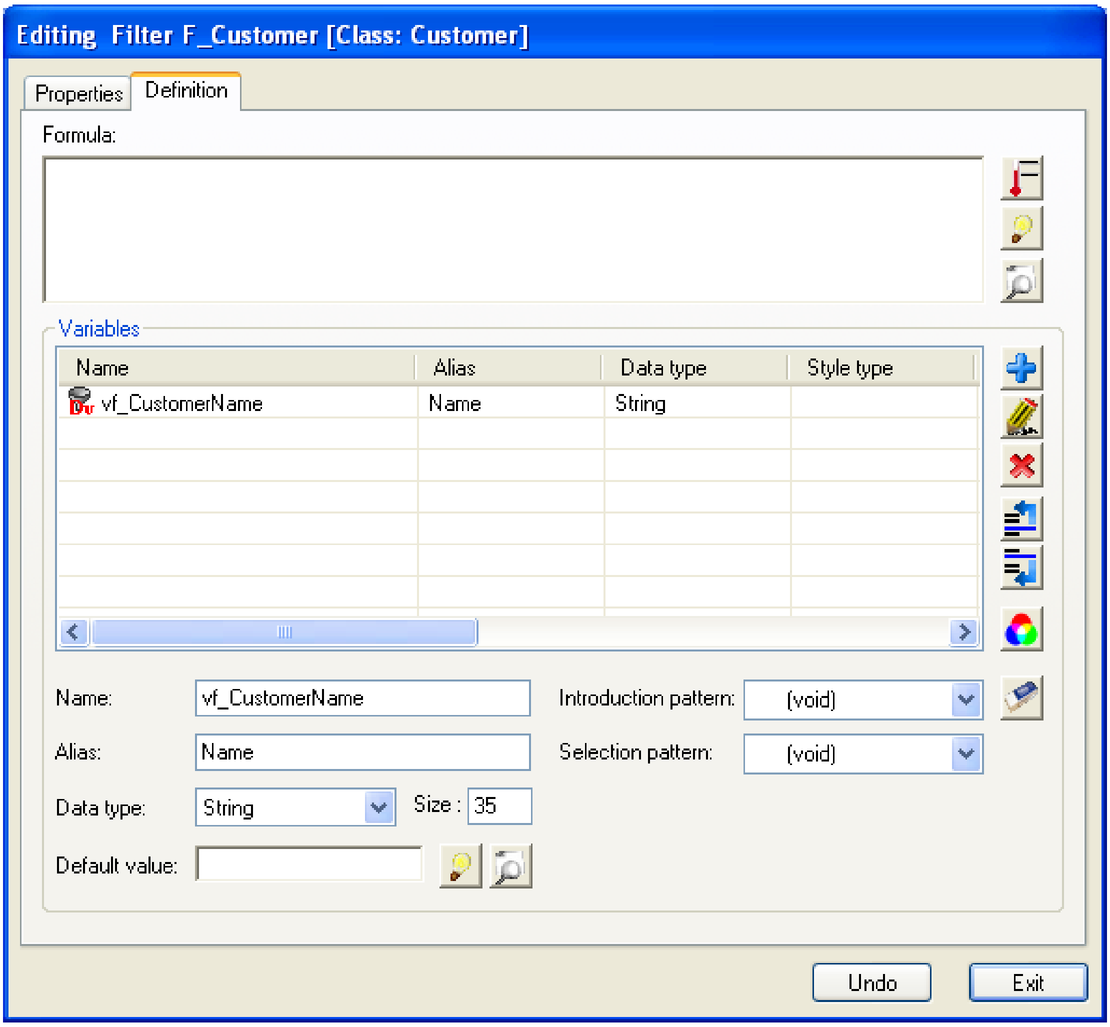
Bearing all this in mind, let's define our filter. In this filter, we would like to recover customers based on this information:

- ✓ Name of the customer starting by "characters given by the user"
- ✓ Customers who are still active on our system (or not).

Well, our condition is based in two variables, so we will define two filter variables as to cover this, and then we will define the filter formula.

Let's create our first filter variable which will be responsible for asking the name of the customer. Then, we start filling in the values of the requested fields.

We enter a name, for instance, ‘vf_CustomerName’. We enter an alias which could be ‘Name’. We enter the filter variable data type, in this case a String type of size 35, and once this is done we press the *Add* button as to add the filter variable. Let’s see the result:



Name	Alias	Data type	Style type
vf_CustomerName	Name	String	

Name: vf_CustomerName

Alias: Name

Data type: String

Size : 35

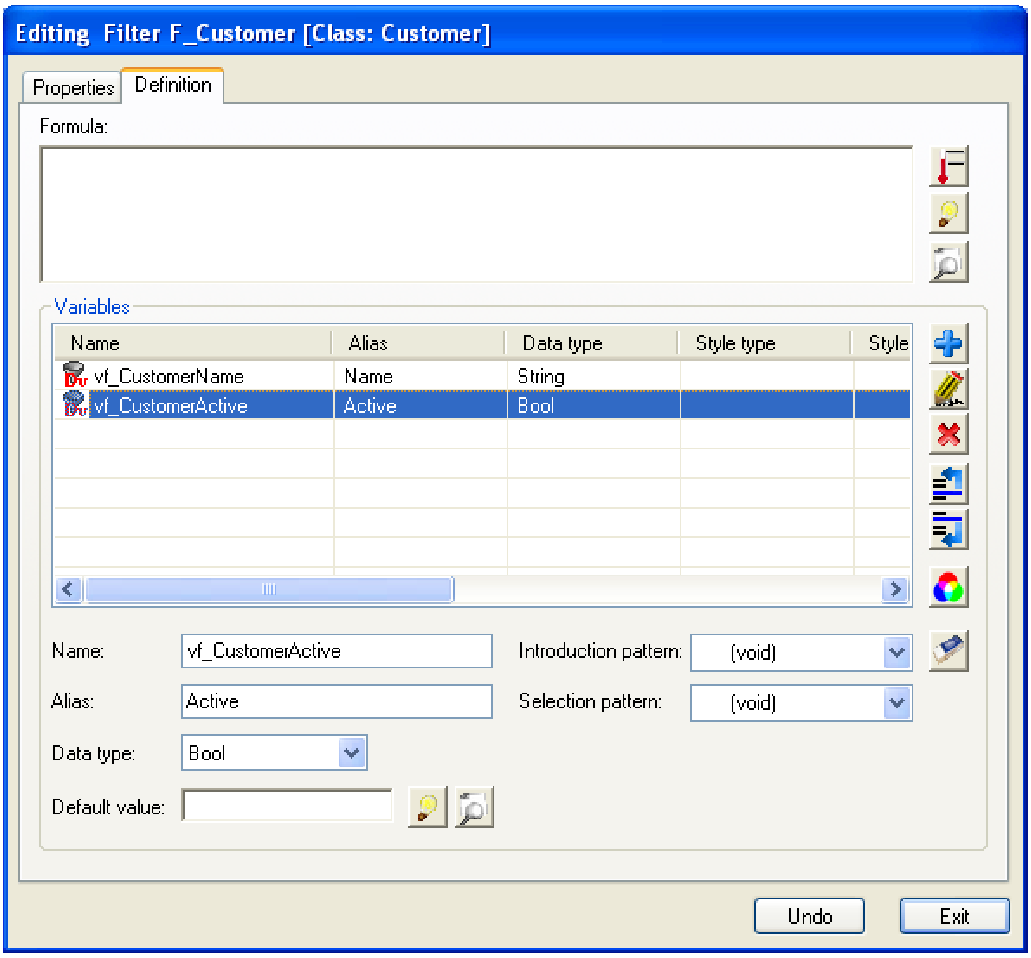
Default value:

Introduction pattern: (void)

Selection pattern: (void)

Figure 4 Filter variable vf_CustomerName of data type String

We carry on with the second filter variable that we need. In this case we would like to see whether the customer in the system is still active or not. Then, we create a new filter variable. We fill in the name ‘vf_CustomerActive’ and we enter an alias, ‘Active’ for instance. Finally we enter the data type that in this case will be Boolean:



Editing Filter F_Customer [Class: Customer]

Name	Alias	Data type	Style type	Style
vf_CustomerName	Name	String		
vf_CustomerActive	Active	Bool		

Name: vf_CustomerActive
Introduction pattern: (void)
Alias: Active
Selection pattern: (void)
Data type: Bool
Default value:

Figure 5 Filter variable vf_CustomerActive of boolean data type

Next, when explaining the formula we will see why we have selected these data types, string first and Boolean next for our two filter variables.

Now it's time for defining the filter formula. We already know that an instance is composed of some attributes each of them storing values. For this particular example, the instance belongs to 'Customer' class. Then if we want to look for some customer name, we will have to use the 'FirstName' attribute of that 'Customer' class, in the formula. This attribute is of String data type, so that's why we have defined a filter variable of String data type (vf_CustomerName) as to match it with the attribute. The same reason for the 'vf_CustomerActive' with the 'ActiveStatus' attribute.

Eventually, having identified the attributes that will take part in our filter formula, and having also previously defined the filter variables the users will use to enter their searching parameters, we are ready to define our first filter variable formula. So we go to the textbox area at the top of the dialog and we enter this:

```
FirstName LIKE vf_CustomerName AND ActiveStatus = vf_ActivedCustomer
```

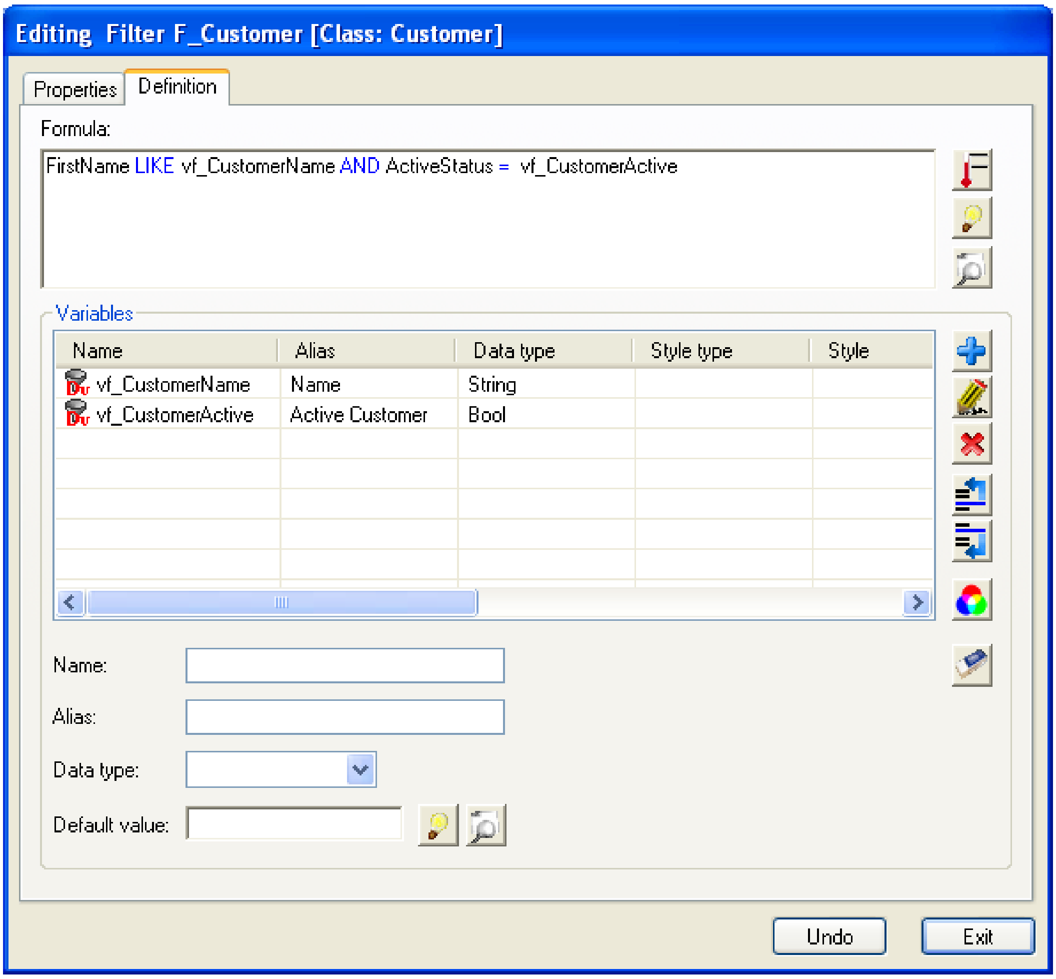
Let's see what this formula means.

We have two requirements separated by an AND conjunction. In the left side of the conjunction, we are using the **LIKE operator**, which is an operator that can be used in filter formulas and which basically means: "return the instances whose 'FirstName' attribute *starts by* the same characters given in the 'vf_CustomerName' filter variable". Of course, as you can imagine, this operator can only be used for Strings and Text data types. The result of this would be: if the user enters an "A" the system will return all the instances whose Customer first name starts by "A". For instance: Albert, Alexander, Anthony…

The left side of the conjunction just equals the 'ActiveStatus' attribute to the filter variable, which means: "return those Customers whose 'ActiveStatus' attribute equals to True or False, depending on what the user specified".

Both conjunctions put together with an AND, allows the system to cover the filter aim, which was allowing the user to search by the name of the customers and their status.

Let's see the final result of our filter:


Editing Filter F_Customer [Class: Customer]

Formula:

```
FirstName LIKE vf_CustomerName AND ActiveStatus = vf_CustomerActive
```

Variables

Name	Alias	Data type	Style type	Style
vf_CustomerName	Name	String		
vf_CustomerActive	Active Customer	Bool		

Name:

Alias:

Data type:

Default value:

Figure 6 'F_Customer' filter in 'Customer' class

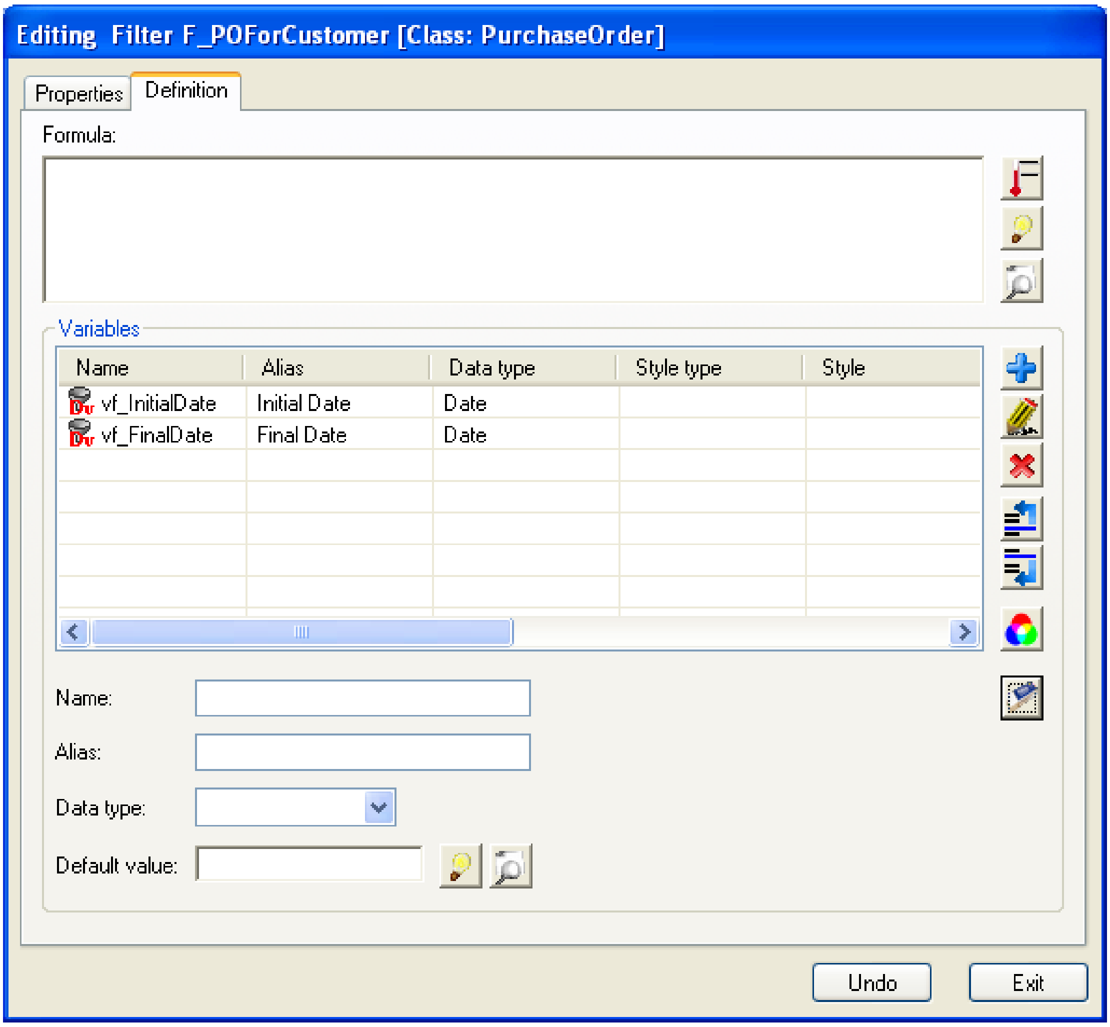
The same way we have learnt along the series, we can make use of the *Help* button, as to help us using attributes, operators and so on. And also we can make use of the *Variables* button as to add the filter variables already defined in the formula.

Let's see another filtering example. In this case we are going to define two filters for purchase orders, one for the Population Interaction Unit created for 'Customer' agents and another for 'Administrator' agents. For the first case, we will define a formula in which Customers will be able to filter purchase orders between dates (an initial date and final date) and in the second case, Administrators in addition to filter between these dates will also be able to filter taking into account whether the purchase orders are already paid or not (still open).

Then, the same way we did previously, we go to the *Presentation Model* dialog and expand the 'PurchaseOrder' class. We are going to model first the filter that will be placed in the population referred to Customers, so we choose the 'PIU_POForCustomer', and the same way we just learnt we create a new filter.

Once in the *Definition* tab, after entering a name, let's say 'F_POForCustomer' and an alias, 'By dates', we start defining the filter variables. We said we would like to filter between dates so somehow we need a date from and a date to, therefore two filter variables of Date type. Let's create them. We will first create the initial date filter variable. We may call it 'vf_InitialDate' giving to it an alias such as 'Initial date'. Of course its data type will be Date. The same way we create the date from filter variable. We enter a name let's say 'vf_FinalDate' and an alias, 'Final Date'. Again, obviously, its data type will be Date.

Let's see the result in the next figure:



Editing Filter F_POForCustomer [Class: PurchaseOrder]

Name	Alias	Data type	Style type	Style
vf_InitialDate	Initial Date	Date		
vf_FinalDate	Final Date	Date		

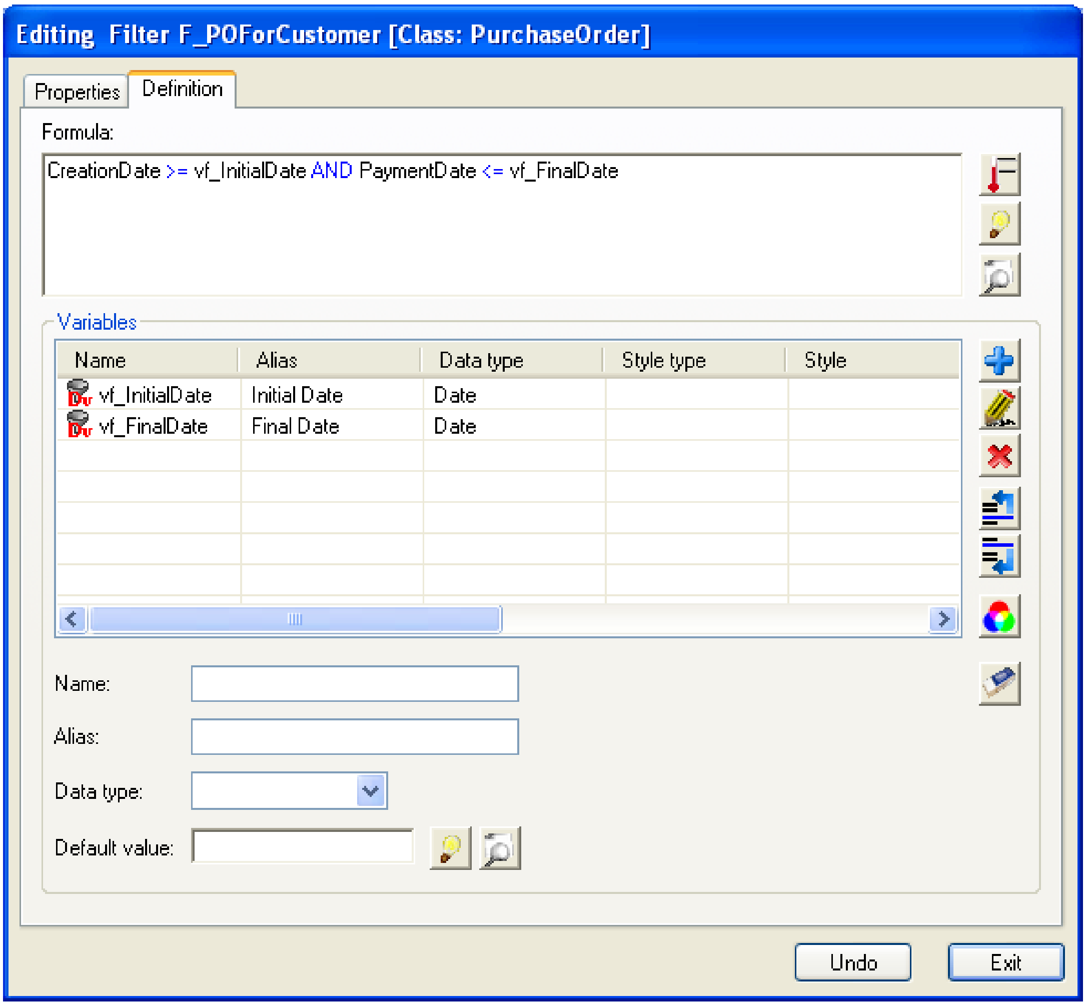
Figure 7 Filtering Purchase orders between dates

Now, we need to enter the filter formula. Again, the same way we did in the first filter example, we must think about the attributes we will be using in this formula. It is clear that we have an attribute named 'CreationDate' of Date data type, which gets its value once the purchase order is created. Therefore, this is the attribute we will be using as to search purchase orders from. However, we do not have any sort of *finishDate* attribute we could use for the final date. But, instead we have the attribute that saves the date in which a purchase order is paid (PaymentDate attribute of data type Date). Thus, when a purchase order is paid we consider it as finished, right? Then we will use the PaymentDate attribute as to a boundary of the final date.

Ok, with all the elements needed in the formula identified, we just need to use our analyst skills to enter the filter formula:

```
CreationDate >= vf_InitialDate AND PaymentDate <= vf_FinalDate
```

That's the result of our second filter:



Editing Filter F_POForCustomer [Class: PurchaseOrder]

Formula:

```
CreationDate >= vf_InitialDate AND PaymentDate <= vf_FinalDate
```

Variables

Name	Alias	Data type	Style type	Style
vf_InitialDate	Initial Date	Date		
vf_FinalDate	Final Date	Date		

Name:

Alias:

Data type:

Default value:

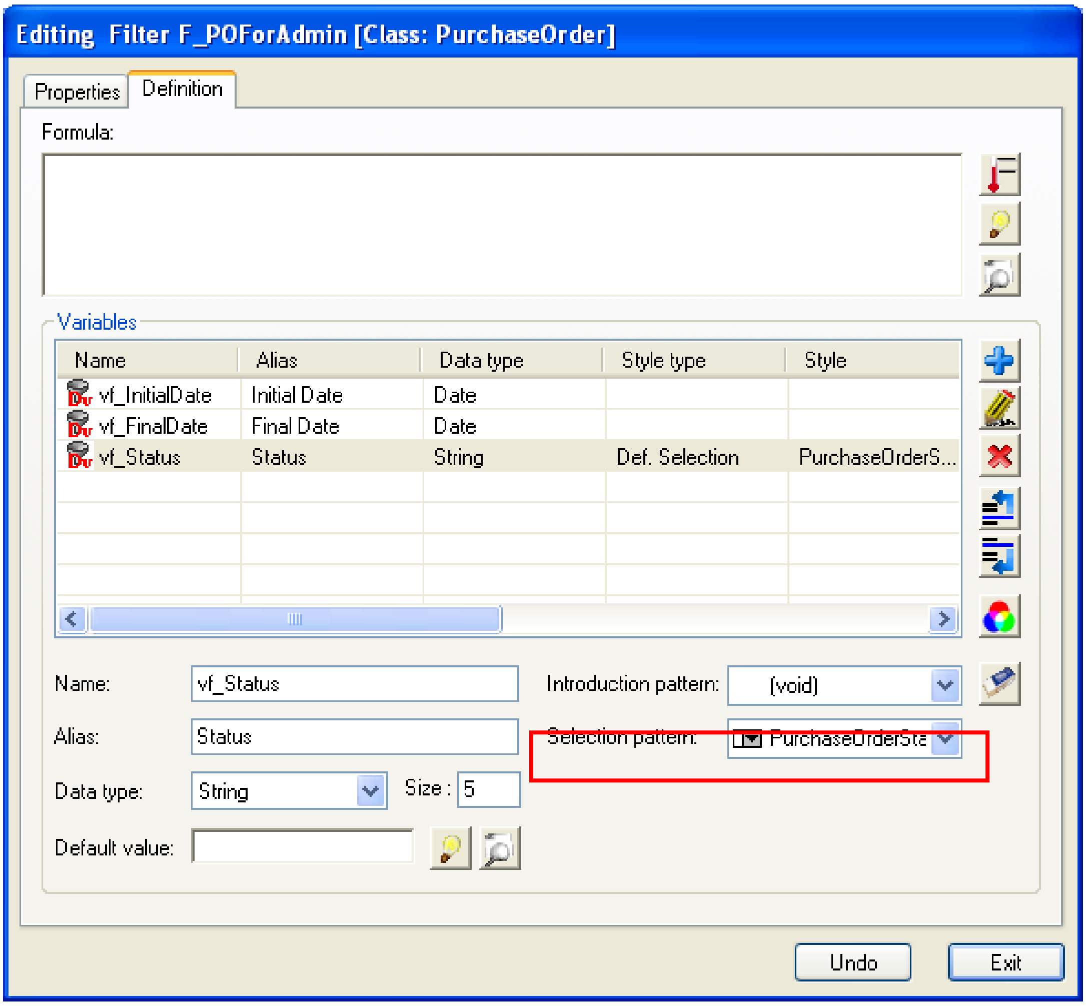
Figure 8 Filtering purchase orders between dates

Eventually, this filter fulfills the first aim of the second filter example. When a customer agent will use this scenario, it will be able to enter an initial date and a final date and in addition to the fact that the horizontal visibility (we saw it in previous series) is applied for customer as to restrict them to only see its own purchase orders, then the customers will see their purchase orders between two given dates.

Let's see now the second case, for 'Administrator' agents. The Administrators can see all the purchase orders in the system, there is no horizontal visibility applied for them. So filtering between dates could also be helpful for them, as the amount of purchase order will be greater. But, they would also like to filter taking into account whether the purchase orders are still in process or already paid. So we will create a new filter for the 'PIU_POForAdmin'. We go to the *Presentation Model* dialog, and open 'PurchaseOrder' class. Then, select the 'PIU_POForAdmin' Population Interaction Unit. As this filter formula will have the same variables than the one we just did for the Customers, we take for granted this part is already done.

Therefore, we add the third filter variable that will allow the user application (in this case an Administrator) to choose whether the status of a purchase order is paid or not. We name it as 'vf_Status', enter an alias and let's say 'Status' and we choose the data type as String of size 5. The filter variable 'vf_Status' is ready. However, let's see another property that can be applied for filter variables. We have seen in previous series that we can create **Defined Selection Patterns** and what they are. Then, we could reuse one of these patterns and apply it to filter variables. In our case, this can be done due to we defined a Defined Selection Pattern for the 'Status' attribute of 'PurchaseOrder' class, and in this filter we have defined an filter variable referring to the same, the status of a purchase order.

Therefore, in the *Selection Pattern* dropdown list of the filter variable definition area, we choose the offered item: 'PurchaseOrderStatus'. We add the filter variable and we are ready to define the filter formula:



Editing Filter F_POForAdmin [Class: PurchaseOrder]

Name	Alias	Data type	Style type	Style
vf_InitialDate	Initial Date	Date		
vf_FinalDate	Final Date	Date		
vf_Status	Status	String	Def. Selection	PurchaseOrderS...

Name: vf_Status

Alias: Status

Data type: String

Size : 5

Default value:

Introduction pattern: (void)

Selection pattern: PurchaseOrderSta

Figure 9 Filter variable with a selection pattern applied

This selection pattern will allow the application user to select from a dropdown list, the filtering status value offered by the Defined Selection Pattern. In this case, two possible values:

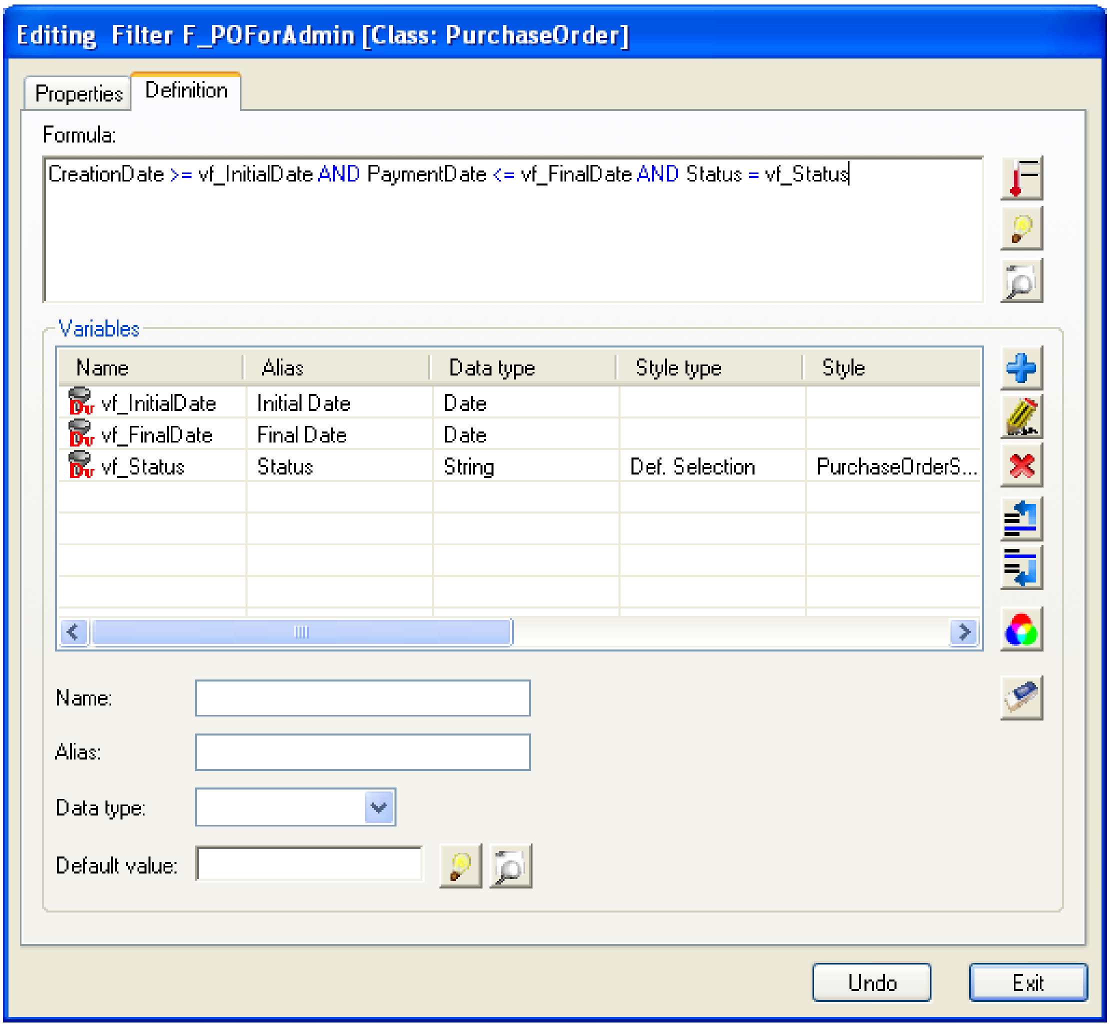
- ✓ Open
- ✓ Paid

Finally, we just have to enter the filtering formula making use of this three filtering variables. The part of the dates is clear, we did it previously. However, for the status we have to think which attribute we will use for the formula. Obviously as the name clues we will use the Status attribute.

Eventually the formula entered is:

```
CreationDate >= vf_InitialDate AND PaymentDate <= vf_FinalDate AND Status =
vf_Status
```

See the figure below:



Editing Filter F_POForAdmin [Class: PurchaseOrder]

Formula:

```
CreationDate >= vf_InitialDate AND PaymentDate <= vf_FinalDate AND Status = vf_Status
```

Variables

Name	Alias	Data type	Style type	Style
vf_InitialDate	Initial Date	Date		
vf_FinalDate	Final Date	Date		
vf_Status	Status	String	Def. Selection	PurchaseOrderS...

Name:

Alias:

Data type:

Default value:

Figure 10 Filtering purchase orders between dates and status

Of course these filter examples we have seen are very simple examples of filtering information although these cases are very common. Anyway, a filter formula can be much more complicated, making use of operators, and son on…

We will see further explanations about filters and other properties that can be applied to filter variables in later on series.

3 Sorting the retrieved information

Often, apart from giving the chance of filtering information through filters, it is also required to sort the information presented. For instance, it is commonly used, when accessing to our e-mail accounts, to have the inbound mail ordered by the date in which they were sent, or sorted by those still not read, and so on…

In this section we will learn how to achieve this using Integranova.

In **Integranova Modeler,** we have an auxiliary pattern named *Order Criteria* that allow defining order to the information retrieved, using attributes as to define that order. In addition, the sort sense can be indicated as well, in terms of ascending or descending.

We will introduce this concept with some illustrative examples.

In the first example, we are going to define a sorting criterion for the 'Customer' class. This order will be applied through the 'FirstName' attribute of 'Customer' class so that the population of customers is ordered by their first name.

Let's start modelling this. We open the *Presentation Model* dialog and go to *I.U.* tab, we expand the *Auxiliary Patterns* folder, we expand again the *Order Criteria* folder, and then right click on 'Customer' class:

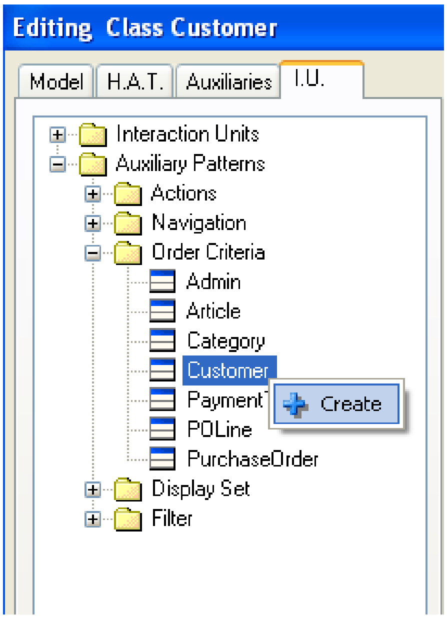
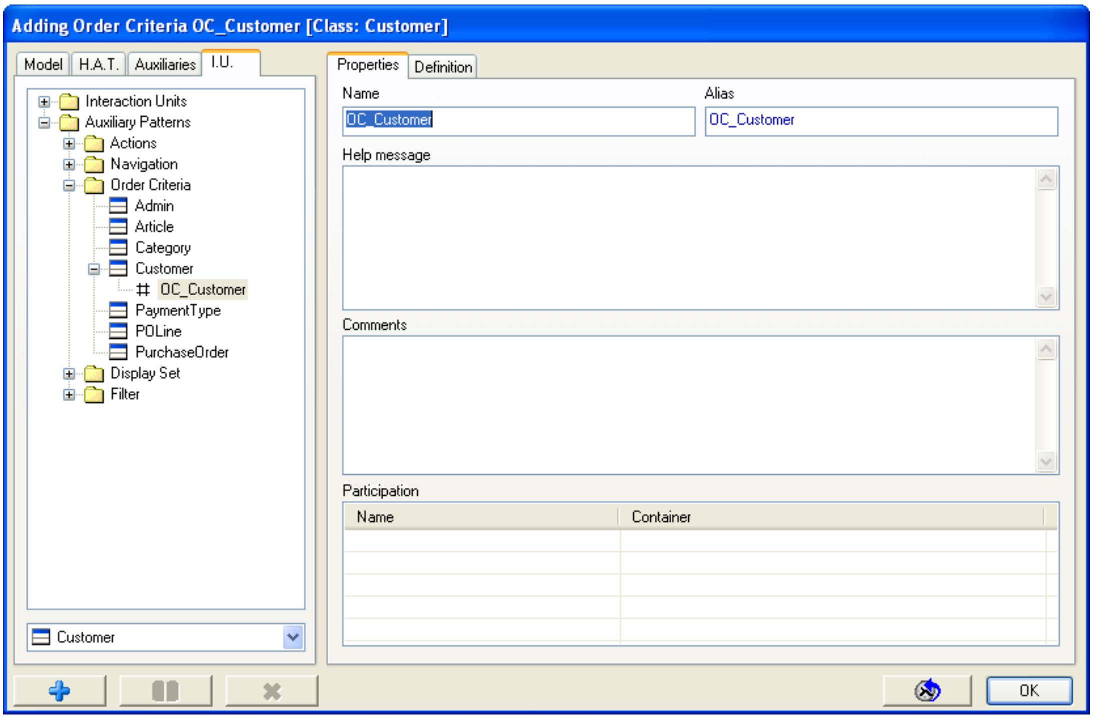


Figure 11 Creating an Order Criterion for 'Customer' class

After clicking on the *Create* option, the order criterion is created:



Name: OC_Customer
Alias: OC_Customer

Figure 12 Order criteria properties

As we can see in the Figure 12, a name and an alias are provided by default. We can change them to more accurate ones. Let's move to the *Definition* tab:

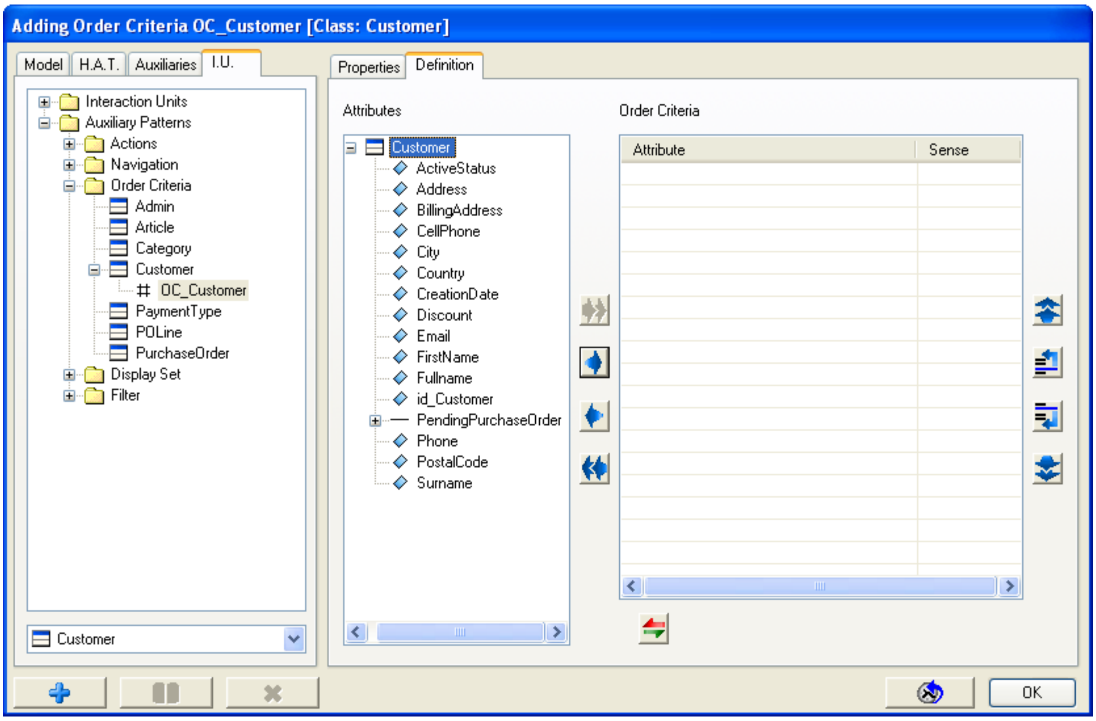
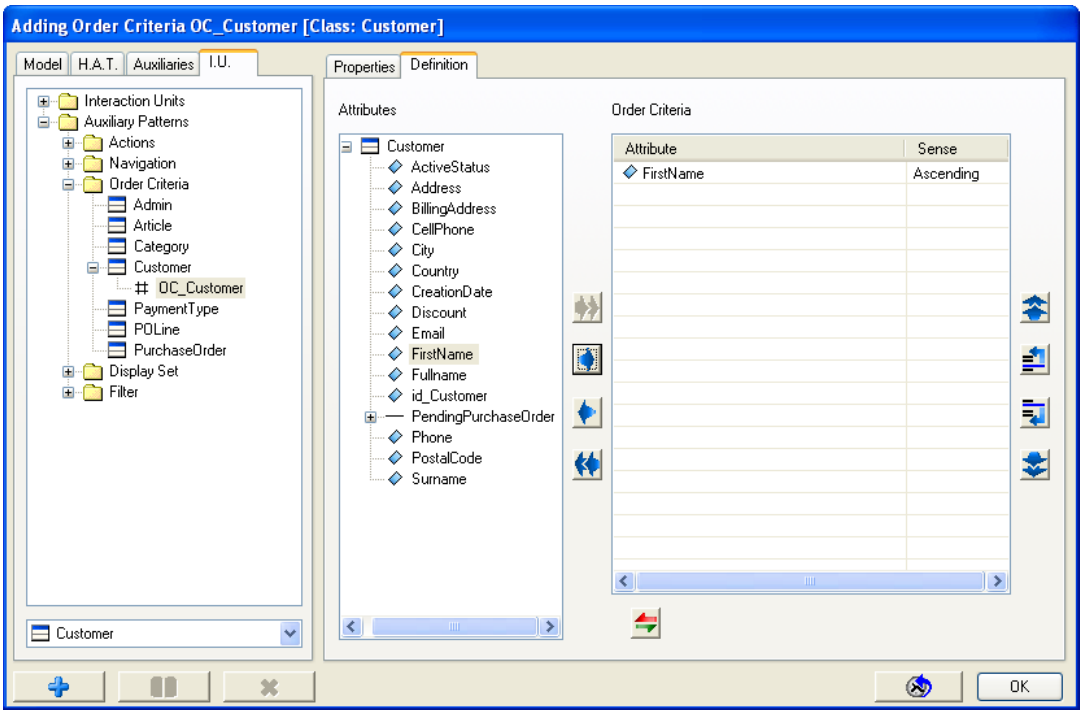


Figure 13 Order Criteria Definition tab

In the *Definition* tab, you can see the attributes of the class to which the order criterion is being defined, and also the attributes of the classes related through a univaluated role path. Therefore, the attribute/s that can be used for defining the order can be chosen from the 'Customer' class or the attributes accessed through the 'PendingPurchaseOrder' role path.

In this case we said we would like to sort the instances of 'Customer' class by their first name attribute. Then, we select the 'FirstName' attribute and we move it to the right side:



Attribute	Sense
FirstName	Ascending

Figure 14 Order criteria. Customers ordered by first name

Notice that, by default, the sorting sense is ascending. We can change that order if needed to descending, through the *Change sense* button.

And the order criterion is done. Afterwards, we can make this order criteria participate is the interaction units we need it, just adding it to them. For instance, imaging the 'PIU_CustomersForAdmin' of 'Customer' class we used in the section 2 when talking about filters. We could make that Population Interaction Unit to take into account that order criterion:

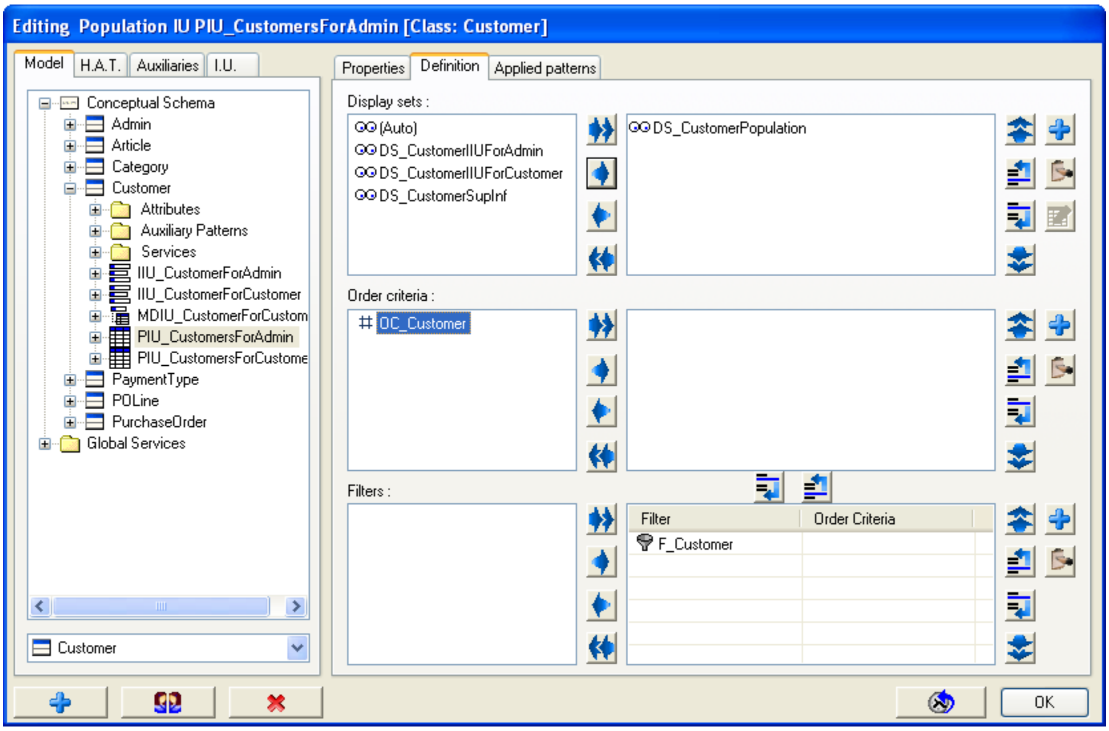


Figure 15 Population Interaction Unit without order criteria assigned

As you can see, when opening that Population Interaction Unit in the Presentation Model dialog, we see we have an order criterion already defined. However, it is not being applied to that Population Interaction Unit. So select it and move it to the right side. Furthermore, if we want this order criterion to be applied automatically with the filtering mechanism, so the filtered instances come with that specific order, then add it to the filter too, making use of the buttons.

Therefore, let's add it and see the result in the next figure:

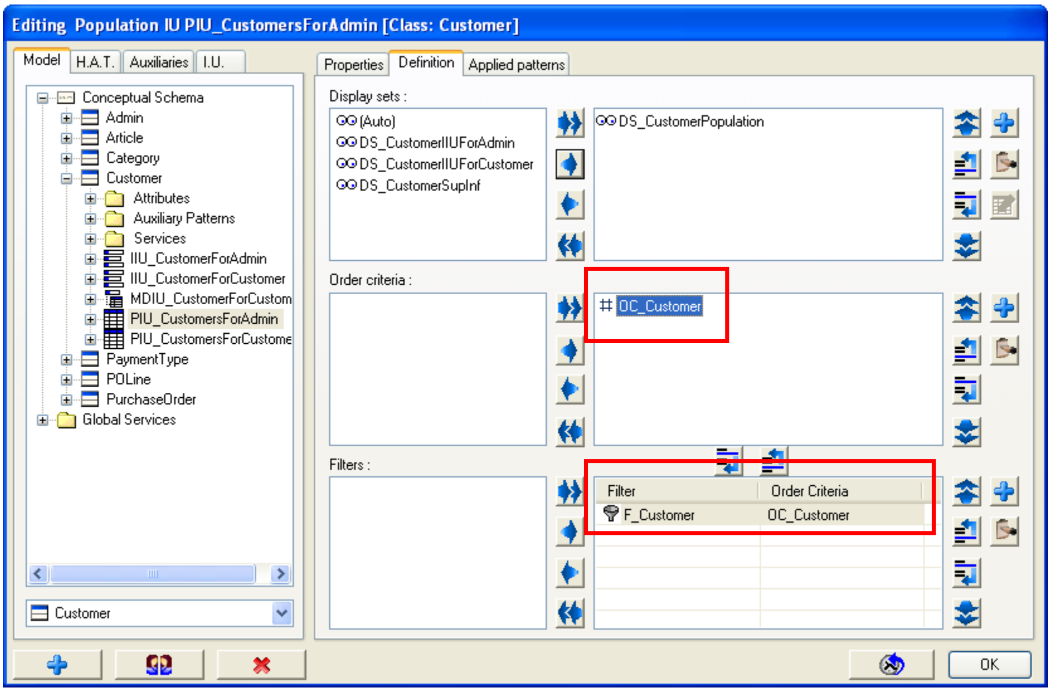


Figure 16 Order criterion applied to the Population Interaction Unit and the filter

Let’s see now a second example that will show how we can use attributes of related classes that can be accessed through a univaluated role path.

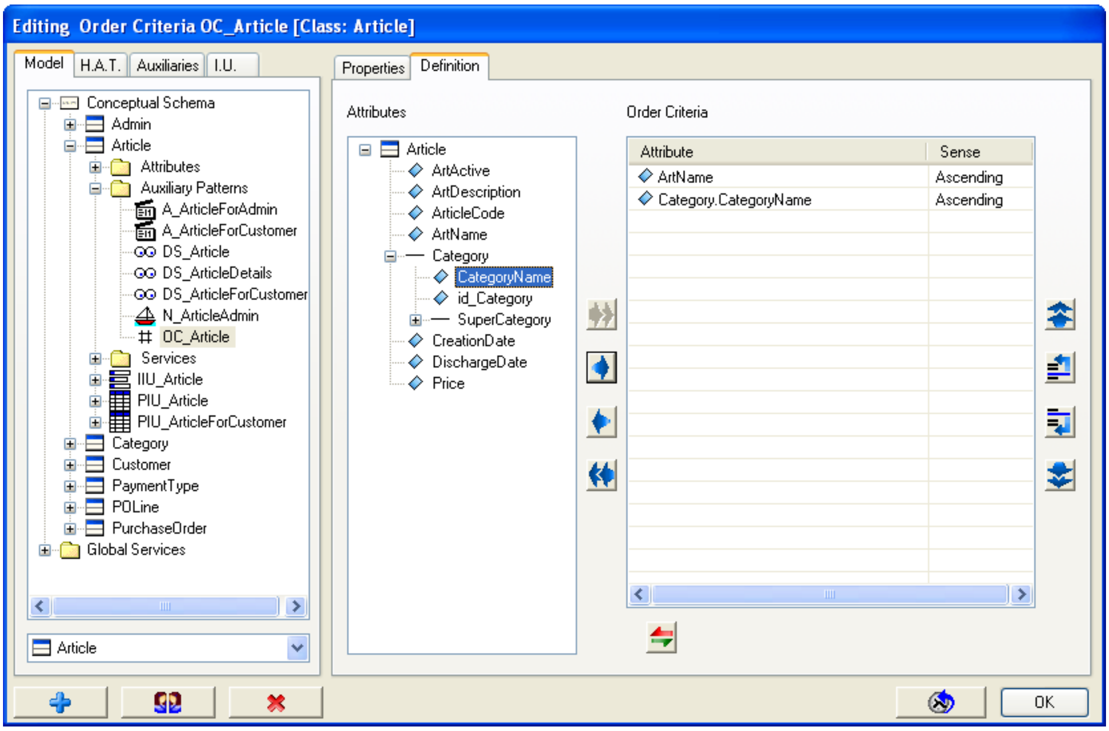
We will define an order criterion for the ‘Article’ class. We know that Articles are related to a specific Category, right? Then, we are going to apply an order criterion over the Articles, in a way that they could be ordered by their article’s name and also their category name.

The same way we did before, we go to the Presentation Model dialog, select *Model* tab, expand the *Auxiliary patterns* folder, expand the *Order Criteria* folder, right click on ‘Article’ class and create a new order criterion. Once you are in the *Definition* tab, we have the attributes of the ‘Article’ class offered as to take part potentially in the order criterion and also we have offered the attributes of the ‘Category’ class, due to this class can be reached from ‘Article’ class through a univaluated role path. Then, as we need to order by the article’s name attribute and the category name that the articles belong to we have to select two attributes:

✓ ‘ArtName’

✓ ‘Category.CategoryName’ (where Category is the name of the rolepath)

Eventually, our second order criterion is defined as follows:



Editing Order Criteria OC_Article [Class: Article]

Order Criteria

Attribute	Sense
ArtName	Ascending
Category.CategoryName	Ascending

Figure 17 Order criterion defined using attributes of a related class

Following these rules, both for filters and order criteria we can define as many of these auxiliary patterns as we need in our model requirements.