integranova
Software Solutions

DEFINING SERVICE INTERACTION UNITS

SAMPLE: SHOPPING CART

Last Update: June 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 mechanisms of the presentation model to give tools for show the information in many ways.

Following with the tutorial series, we will now expand the Tutorial 10, with this new tutorial that will build on the previous one.

Let’s now expand the presentation model of our system adding new elements.

By the end of this tutorial, you will have learnt mechanisms to represent the information in a structured way, show complementary information for object valued arguments, preload information for users and so on.

2 Opening the initial model

In this tutorial, we will start from the model we created in the previous tutorial in the series: Tutorial_10_Final.oom.

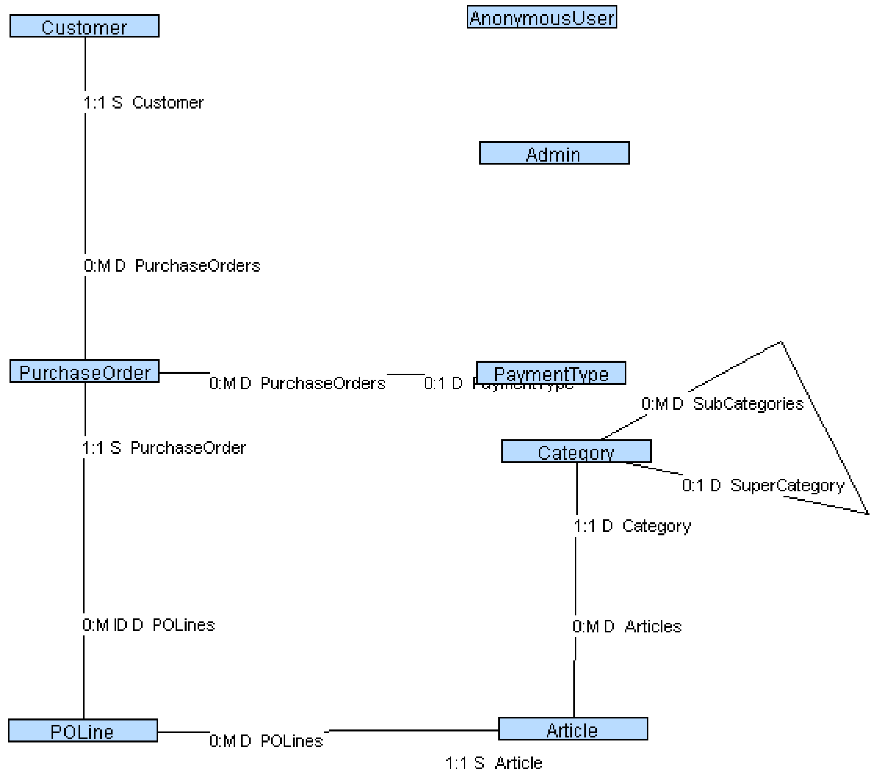


Figure 1 Tutorial_10_Final.oom

3 Introduction to the concept of Defined Selection Pattern

In this tutorial, we require to have more functionality in purchase orders. We want to enumerate the possible values of the 'Status' attribute in a single selection box for our own convenience. The way to do that, in Integranova, is through the definition of Defined Selection Pattern.

A Defined Selection pattern is the enumeration of the valid values for a selection range.

Defined Selection Patterns can be associated to an attribute, a data-valued argument or a data-valued filter variable. This pattern behaves as an enumerated type.

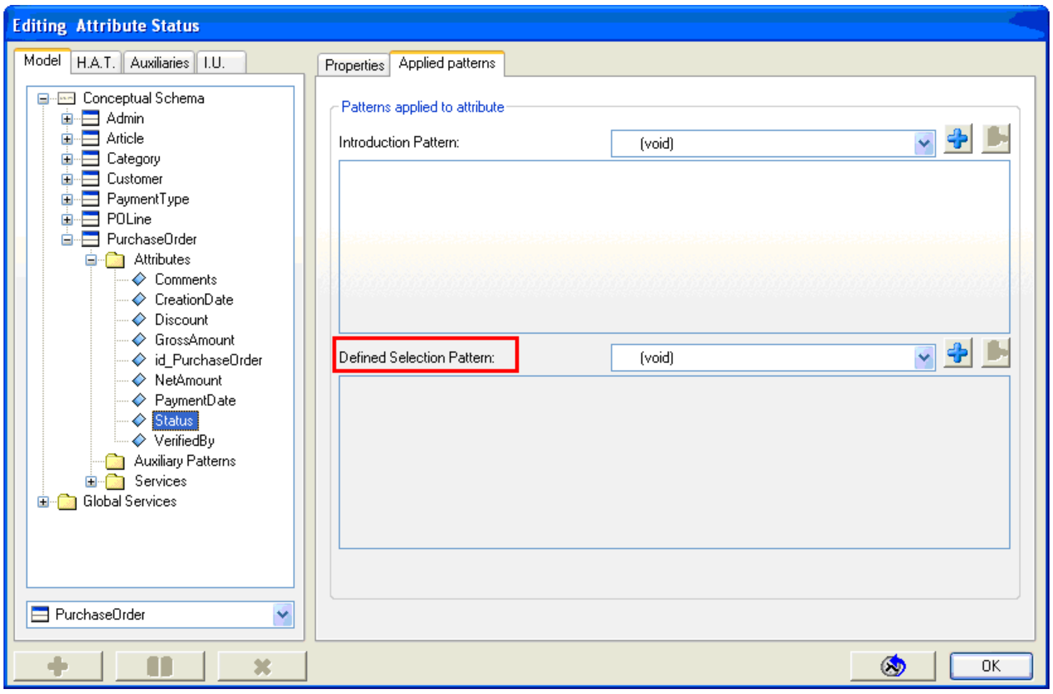
A Defined Selection pattern has two main goals:

- ✓ Enumerates the possible values for an argument restricting the selection to a specified set of values.
- ✓ Allows the definition of aliases for the possible values of the selection.

3.1 Defining a Defined Selection Pattern

As already said, we need to enumerate the valid values for the 'Status' attribute because we don't want to separate the internal value stored in this attribute from the displayed value. In order to reach this functionality, we will define a Defined Selection Pattern that will be assigned to the 'Status' attribute of the 'PurchaseOrder' class.

Let's go to the Presentation Model dialog and in the *Model* tab, select the 'Status' attribute from the *PurchaseOrder* class. At the right, in the *Applied patterns* tab, you can see that the current value of the Defined Selection Pattern is '(void)', that is, no pattern is applied at the moment. Press the *Add* ➕ button to create a new Defined Selection Pattern.

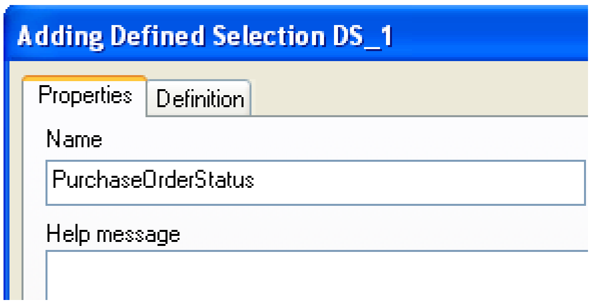


Introduction Pattern: (void)

Defined Selection Pattern: (void)

Figure 2 Applied patterns Defined Selection Window

First, we are going to give our Defined Selection Pattern a proper name, in this case 'PurchaseOrderStatus'.



Adding Defined Selection DS_1

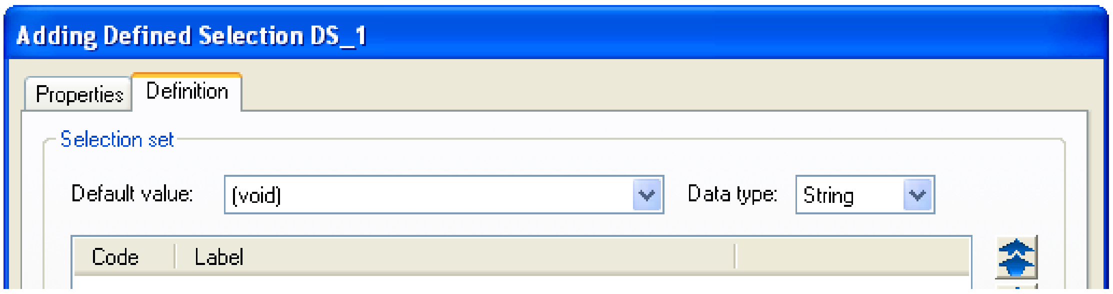
Properties | Definition

Name: PurchaseOrderStatus

Help message:

Figure 3 Choose a proper name for the Defined Selection Pattern

After that, we are going to define the Defined Selection Pattern. First we will have to select the Data type for the Defined Selection Pattern. The data type must be the same data type of the attribute we want to assign it.



Adding Defined Selection DS_1

Properties | Definition

Selection set

Default value: (void)

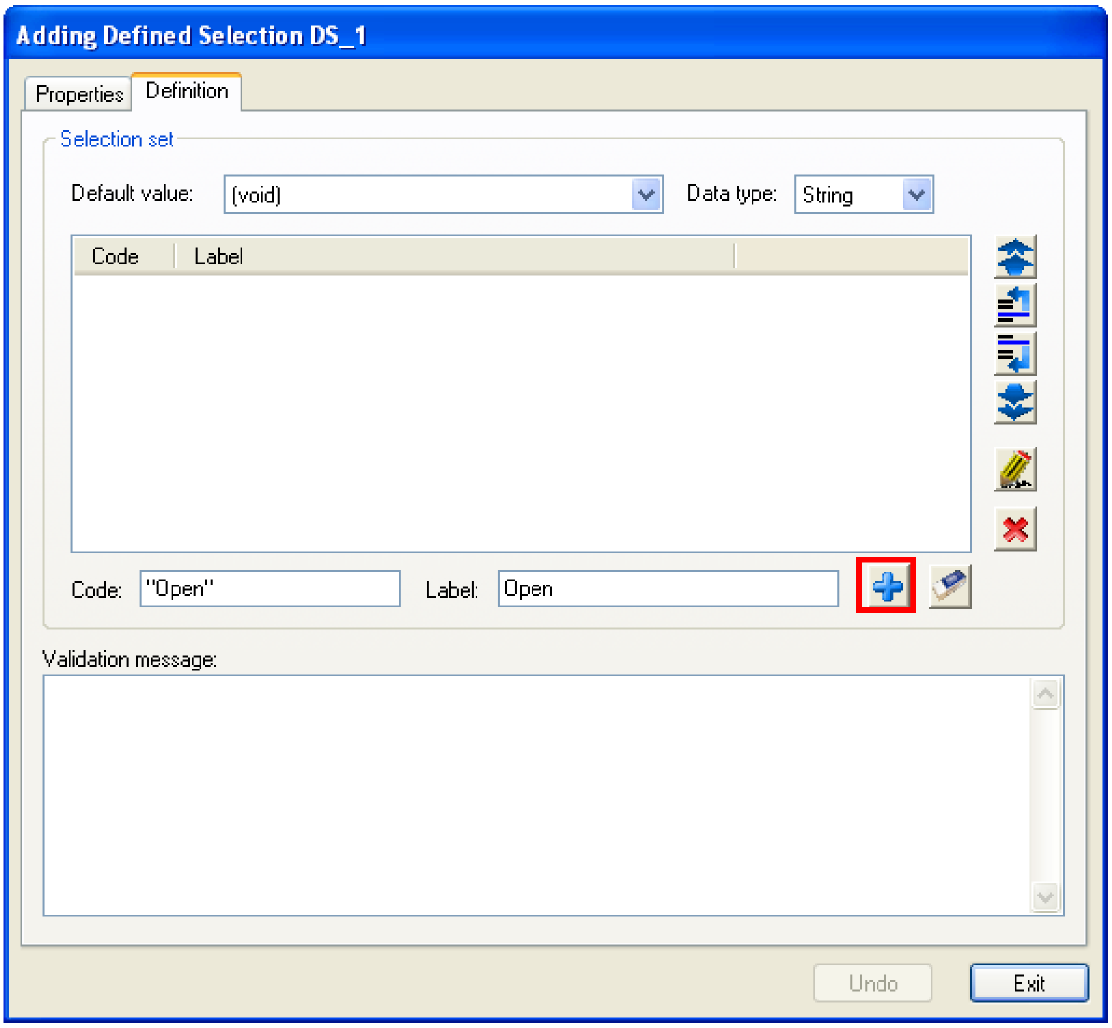
Data type: String

Code	Label

Figure 4 Setting the Data type of the Defined Selection pattern

Next, we will add the possible options that can be selected by the user application. For the *Code* we need to indicate the internal value the application will store, whereas in *Label,* we will enter the name that will be displayed to the user application.

Define the two possible values for the 'Status' attribute: *Open* and *Paid.* Don't forget to press the *Add* ➕ button to add the defined selection to the list.



Default value: (void)
Data type: String

Code	Label

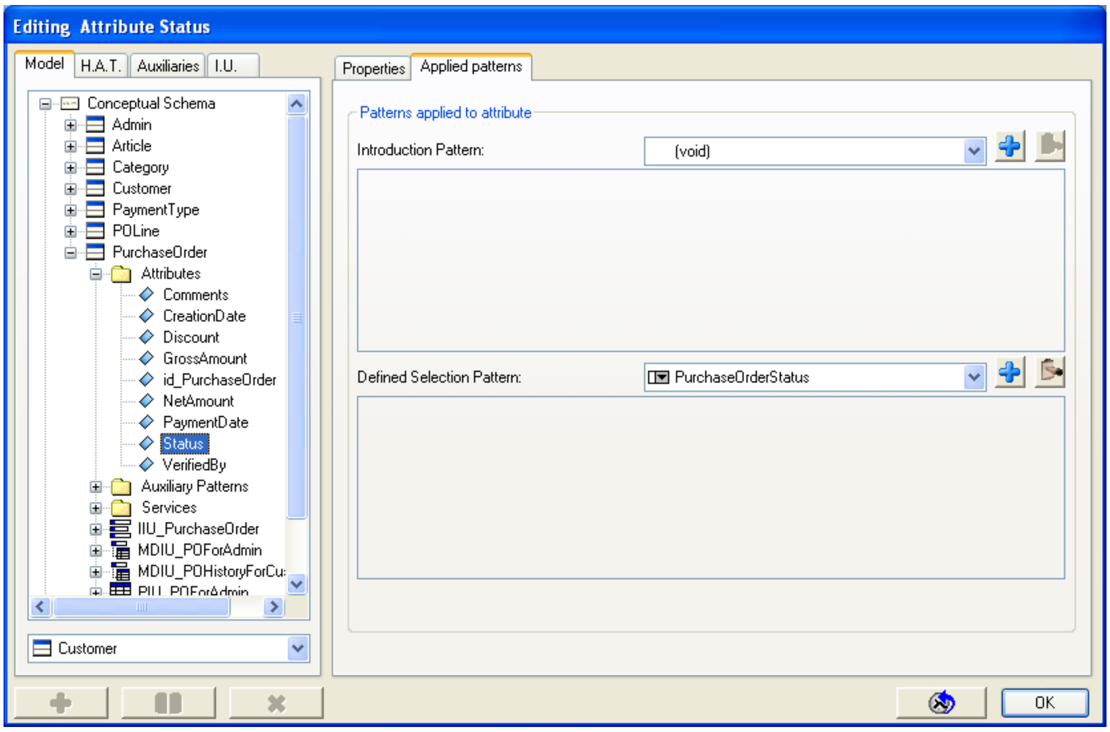
Code: "Open"
Label: Open
Validation message:

Figure 5 Adding elements to our Defined Selection Pattern

Note: The value entered in *Code* must be of the **same type** of the Defined Selection Pattern itself.

When you have finished press *Exit* button.

Finally, you can see the 'PurchaseOrderStatus' Defined Selection Pattern selected in the *AppliedPatterns* tab of the *Editing Attribute Status* window.


Editing Attribute Status

Patterns applied to attribute

Introduction Pattern: (void)

Defined Selection Pattern: PurchaseOrderStatus

Figure 6 adding a defined selection pattern

This is the mechanism to define Defined Selection Patterns.

4 Additional information when selecting an object

In the final application the way in which the user can select an object is using the OID (object-identifier) that is the attributes that compose the identification function.

Sometimes, the attributes used in the OID doesn't have a relevant semantic meaning. For example, in our model, the customer is identified by a number (id_Customer). Some examples could be:

Table 1 Customer data examples

Id_Customer	First Name	Surname	...
1	John	Smith	...
2	Caroline	Brown	...

It is impossible that the application user remembers all the identifiers of the customers. If the application shows a customer with the value equal to '1', how can the user be sure that ithe customer is the one the he wants?

The way in with Integranova helps the user in that case, is providing a mechanism to define a set of attributes that are shown to the application user, when he selects or enter an OID. This mechanism is called **Supplementary Information Pattern**.

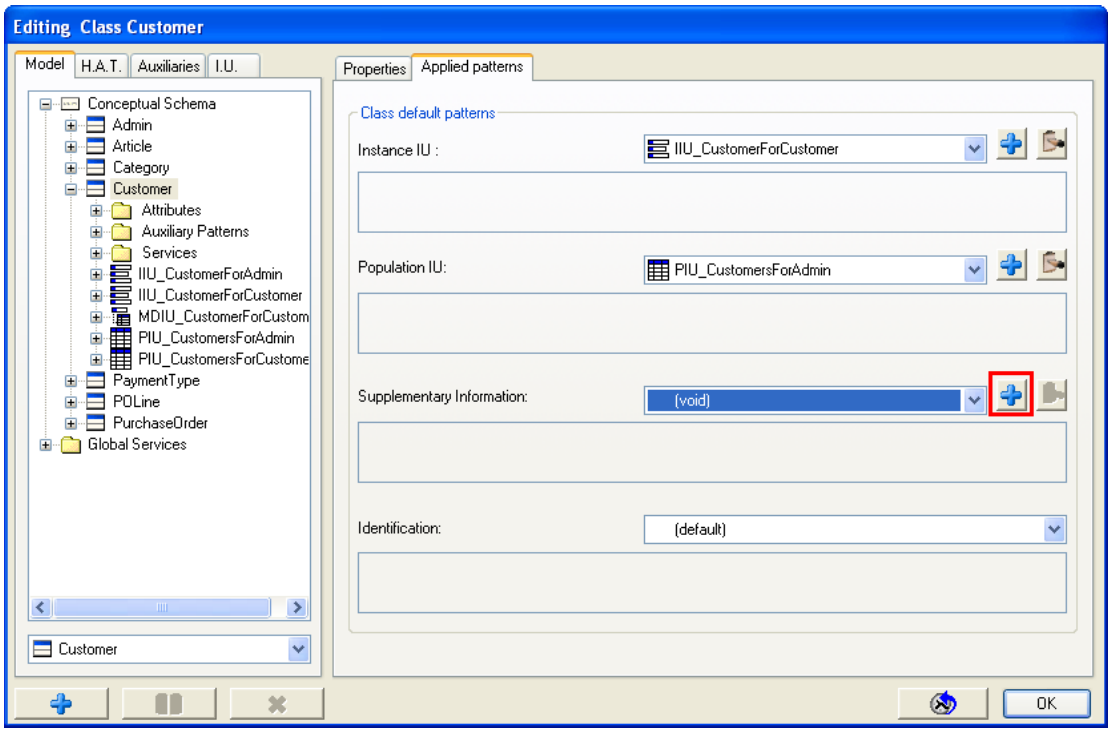
Supplementary Information Pattern captures the additional information that will be showed to a user when an object is selected. It helps the user to confirm the selection. The Supplementary Information Pattern is a Display Set pattern. It is possible to assign a Supplementary Information Pattern to any object (object-valued filter variable or object-valued inbound argument).

If you want to use the Supplementary Information Pattern each time that a 'Customer' object is selected (wherever you use it), then you has to apply this pattern to the 'Customer' class. All the customers will use this Supplementary Information Pattern, except that ones that have a specific pattern applied itself.

4.1 Defining a Supplementary Information Pattern

We are going to show additional information of our customers in all the services that participate. This information will help us to identify our final customer after a selection, let's go to see how to do that:

Press to *Presentation Model* button and in the Presentation Model dialog, select *Model* tab. Next, expand the *Conceptual Schema* item, then select the 'Customer' class and in the *Applied patterns* tab (look the right side of the window), press the *Add* button that belongs to the *Supplementary Information* combo box.



Instance IU : IIU_CustomerForCustomer

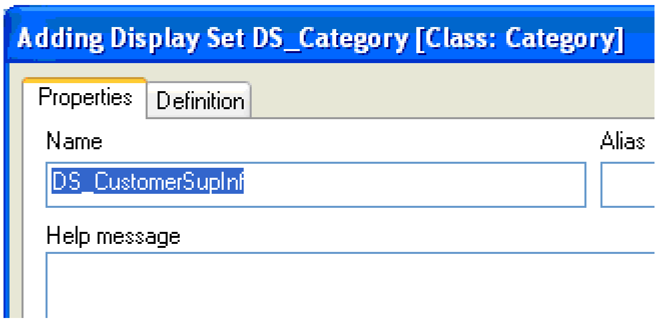
Population IU: PIU_CustomersForAdmin

Supplementary Information: (void)

Identification: (default)

Figure 7 Applied patterns supplementary information window

Now we are going to define a representative name ("DS_CustomerSupInf") for our Supplementary Information Pattern.



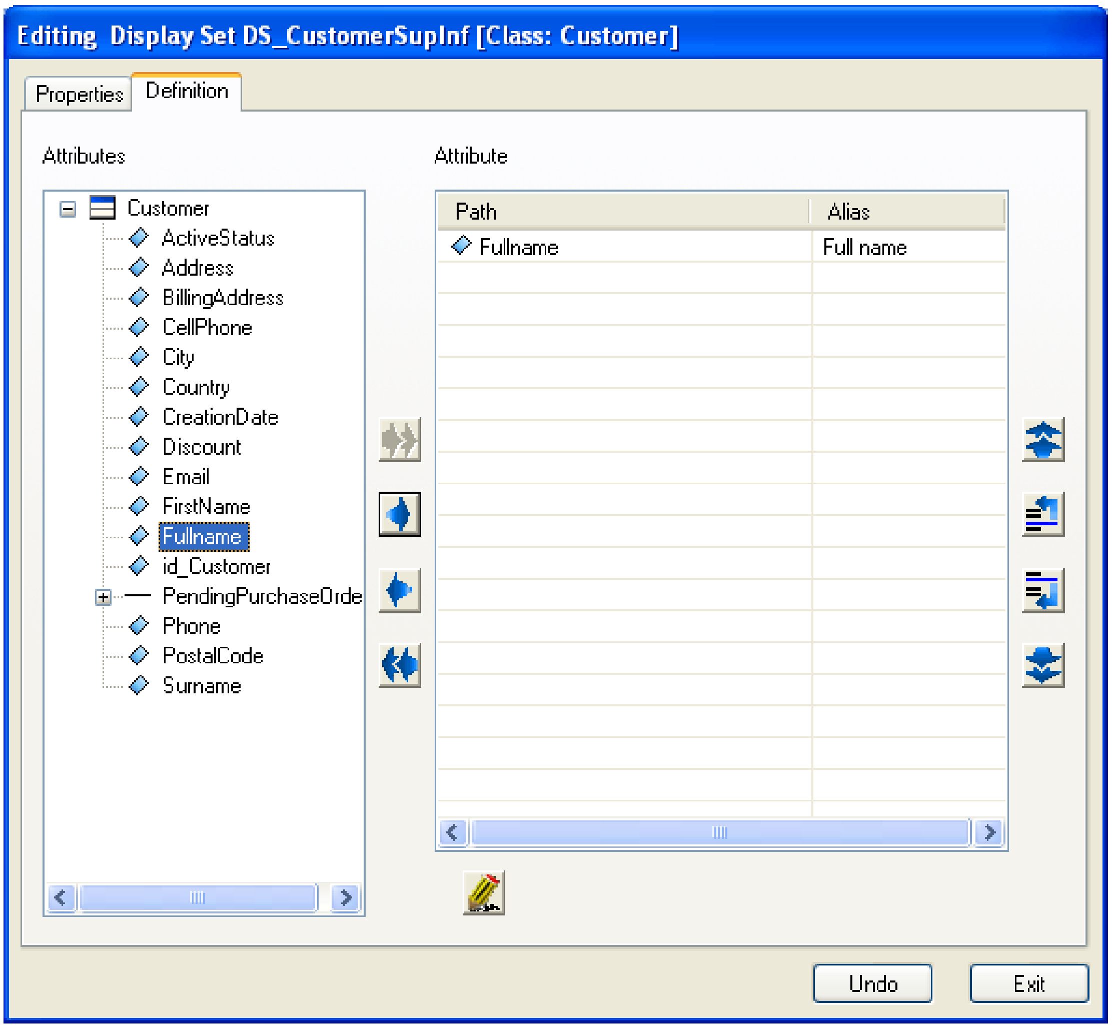
Name: DS_CustomerSupInf

Alias

Help message

Figure 8 Set a name to our Supplementary Information Pattern

After that, in the 'Definition' tab we are going to choose the attributes we want to show, when a customer and the order in which we want to show the supplementary information, you select the attributes with the add or delete arrows, to quit or put in the Attribute part, and with the move up all , move down all , move up and move down put the attributes in the order that you prefer.



Editing Display Set DS_CustomerSupInf [Class: Customer]

Properties | Definition

Attributes

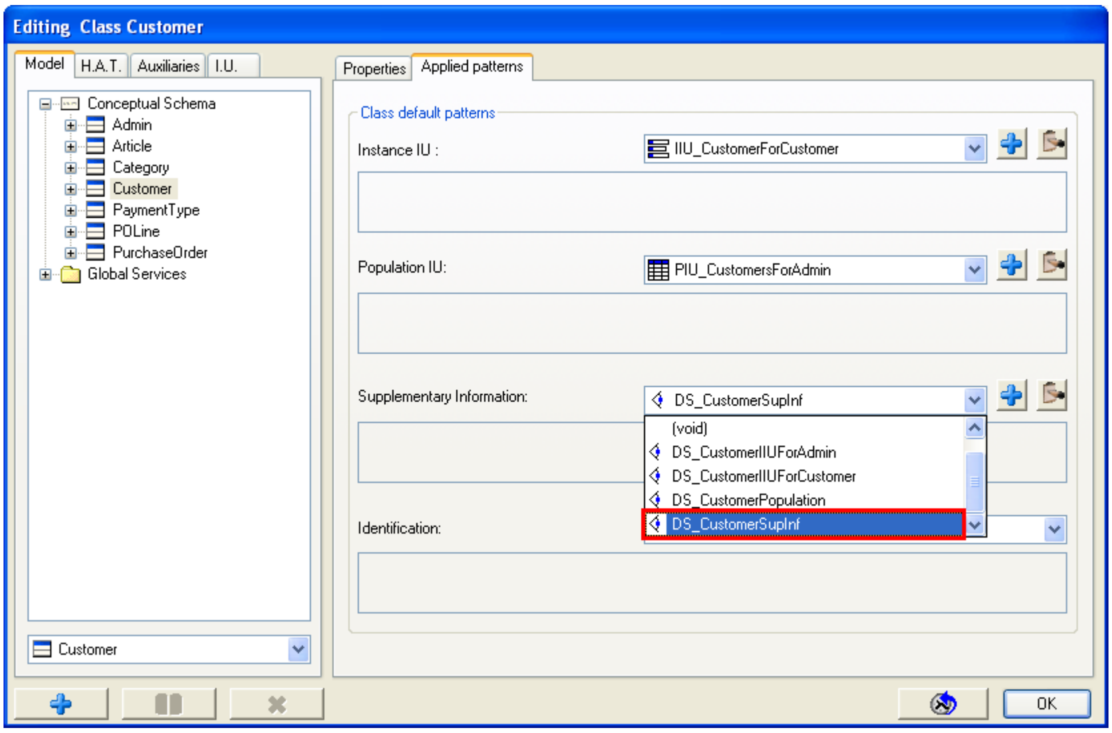
- Customer
  - ActiveStatus
  - Address
  - BillingAddress
  - CellPhone
  - City
  - Country
  - CreationDate
  - Discount
  - Email
  - FirstName
  - Fullname
  - id_Customer
  - PendingPurchaseOrde
  - Phone
  - PostalCode
  - Surname

Attribute

Path	Alias
Fullname	Full name

Figure 9 Adding attributes to our Supplementary Information Pattern

When you finish, press *Exit* button. Finally, you will be back to the *Editing Class Customer* window and you will see that in *Supplementary Information* combo box has a new Supplementary Information Pattern already assigned.



Instance IU : IIU_CustomerForCustomer

Population IU: PIU_CustomersForAdmin

Supplementary Information: DS_CustomerSupInf
- (void)
- DS_CustomerIIUForAdmin
- DS_CustomerIIUForCustomer
- DS_CustomerPopulation
- DS_CustomerSupInf

Identification:

Figure 10 Supplementary information

This is the mechanism to assign Supplementary Information Pattern to a class.

5 Selecting an object in the same scenario

In the applications we can build in this moment, if we want to introduce the value of an object, then we have two options:

✓ Introduce manually the value of the attributes that define the OID. If we can remember the identification of the object.

✓ Select the object we want, from another scenario (using a Population Interaction Unit). When we don't have learnt by heart the OID of each object, we can open a new scenario, and select one of the objects it shows.

Note: Scenario is a generic name that represents a form in windows, a page in web, etc…

But, Integranova offers a mechanism to show a list of objects in the *same scenario* we want to use an object. This mechanism is called **Population Preload**. Population Preload is a Boolean property that shows a list of instances in the same scenario where we are going to select one.

That suits very well when we know that a class has only few instances. For example, it is better to show the list of payment type in the same scenario when we want to pay a purchase order, instead of accessing to other scenario to pick one, and come back to the original scenario.

This pattern is very useful, but remember that the number of instances should be reduced enough in order that it turns out easy and comfortable. If we want the instances to be shown ordered following a specific criterion, then we can use an Order Criteria. If any order is not defined, the instances will be ordered by the identification function of the class. To see more about Order Criterion, go to Tutorial 12 in tutorial series.

Population Preload can be defined:

- ✓ For an object-value filter variable: for a specific object-valued filter variable, its population will be preloaded.

- ✓ For an object-value Argument: for a specific object-valued inbound argument, its population will be preloaded.

- ✓ For all object elements of a class: If you want that any time you can select an object of the class, you have a population preload, you must define it for the class.

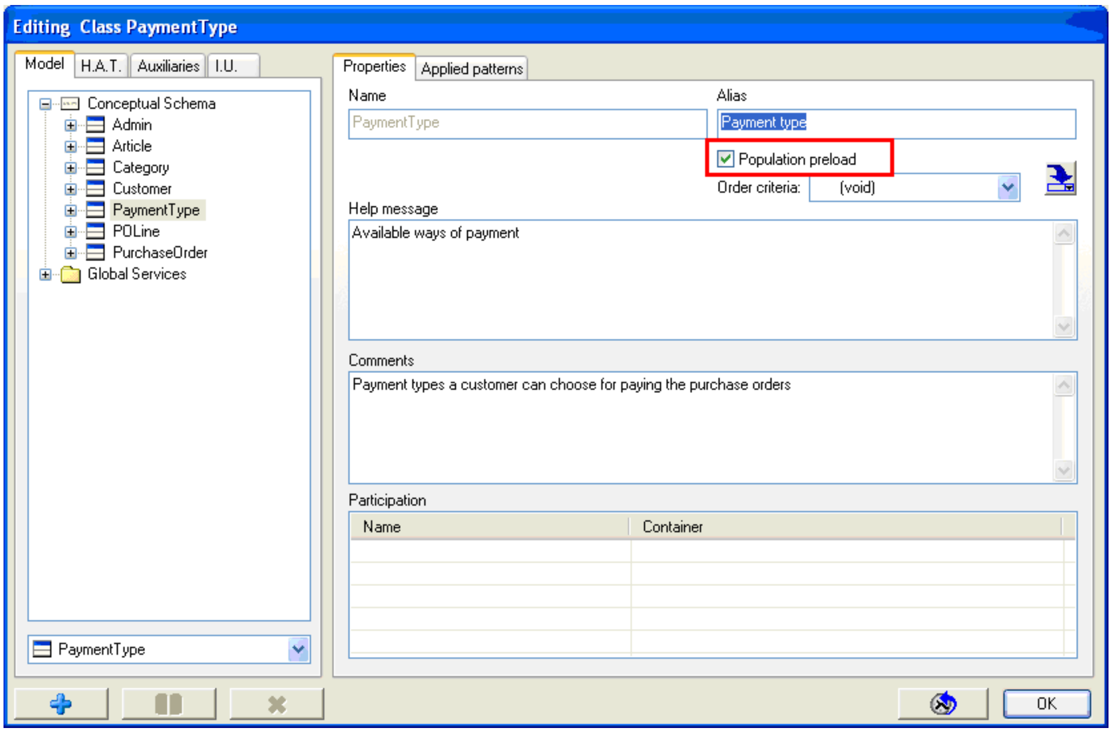
Note: The information, that allows identifying each instance shown in the Population Preload, is specified specifically by the analyst defining a Supplementary Information. The analyst must select representative attributes of delimited size.

5.1 Defining a Population Preload

As we said, in order to add more functionality to our model, we are going to select any object from 'PaymentType' class using Population Preload and also the value of the 'p_agrCategory' argument in the 'create_instance' service of the 'Article' class will selected using Population Preload.

To define a *Population Preload* in a class scope, press on the *Presentation Model* button and the Presentation Model dialog is open. After that, select *Model* tab and open the *Conceptual Schema* item. Next, select the 'Payment Type' class where the Population Preload property is going to be activated.

Then, select the *Properties* tab and enable the *Population preload* check, optionally, associate it to order criteria ('(void)' in our example).



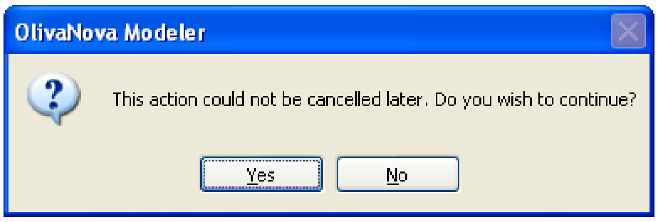
Name: PaymentType
Alias: Payment type
[✓] Population preload
Order criteria: (void)
Help message: Available ways of payment
Comments: Payment types a customer can choose for paying the purchase orders
Participation

Name	Container

Figure 11 Properties population preload window

In case the analyst would like to propagate the Population Preload property to all existing arguments and filter variables of the domain class, it must check on *Propagate Population Preload* button.

If you click on the *Propagate Population Preload* button, a warning dialog advices you that the propagation process cannot be cancelled later.


OlivaNova Modeler

This action could not be cancelled later. Do you wish to continue?

Yes | No

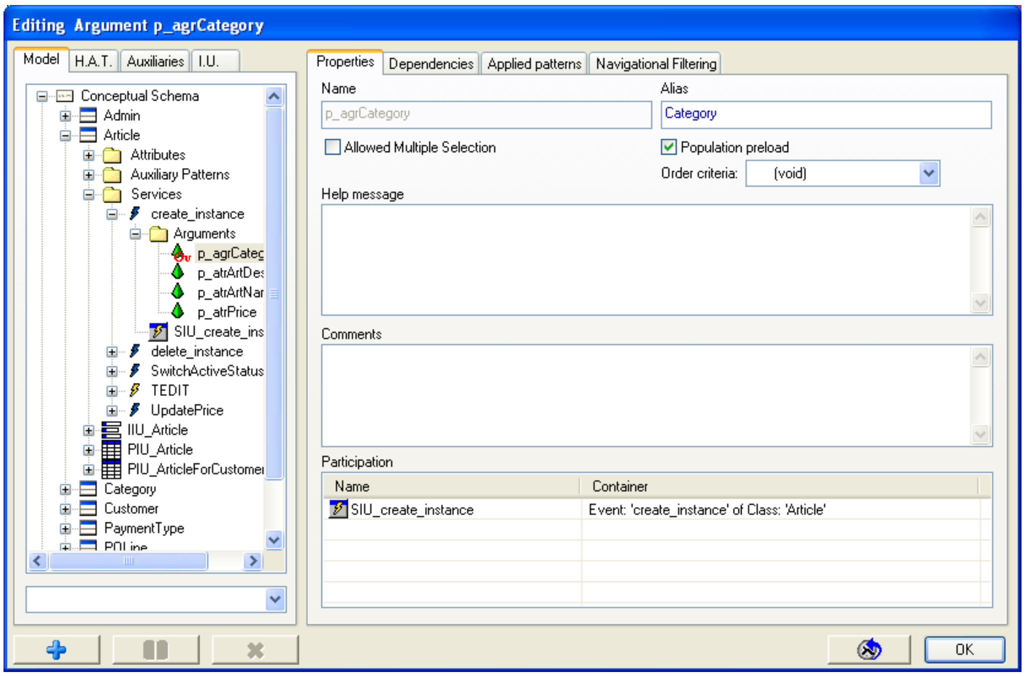
Figure 12 Dialog to warn the propagation action cannot be cancelled later

In case the Supplementary Information is '(void)', the identification function will be showed to represent the instances in the preloaded population.

With these steps, we'll have our first Population Preload defined in a class scope.

Let's go now to create the Population Preload for the 'p_agrCategory' argument in the 'create_instance' service for the 'Article' class. That means that, when we are going to create an article, the list of categories of an article will be shown in the same scenario. The mechanism is quite similar to assign the Population Preload property to a class, but instead a class, it is assigned in the argument of the service.

Press on the *Presentation Model* button and the Presentation Model dialog is opened. Select *Model* tab and open the *Conceptual Schema* folder, then a list of folders with the name of class appears. Next, select the class where the Population Preload property is going to be activated. Then, go to the *Services* folder and choose the service or transaction you will activate the Population Preload property (in this case 'create_instance' service). After that, select the 'Arguments' folder and the 'p_agrCategory' argument. Finally, in the *Properties* tab, enable the *Population preload* check, and optionally, associate it to one order criterion.


Editing Argument p_agrCategory

Model | H.A.T. | Auxiliaries | I.U.

- Conceptual Schema
  - Admin
  - Article
    - Attributes
    - Auxiliary Patterns
    - Services
      - create_instance
        - Arguments
          - p_agrCateg
          - p_atrArtDes
          - p_atrArtNar
          - p_atrPrice
        - SIU_create_ins
      - delete_instance
      - SwitchActiveStatus
      - TEDIT
      - UpdatePrice
    - IIU_Article
    - PIU_Article
    - PIU_ArticleForCustomer
  - Category
  - Customer
  - PaymentType
  - POLine

Properties | Dependencies | Applied patterns | Navigational Filtering

Name: p_agrCategory
Alias: Category
☐ Allowed Multiple Selection
☑ Population preload
Order criteria: (void)
Help message
Comments
Participation

Name	Container
SIU_create_instance	Event: 'create_instance' of Class: 'Article'

OK

Figure 13 Properties population preload window

Note: Don't forget to assign a Supplementary Information Pattern to the argument.

6 Selecting multiple objects

During the use of an application, there are situations where the user would like to execute a service several times over different objects.

The way in which we can do that is using **Multiple selection** (multiselection). **Multiselection** is the feature that **allows the user to select more than one instance for an object-valued inbound argument** of a service and execute this service as many times as instances had been selected.

Multiselection allows sending a service execution call from Client side of the application to the Server side as many times as instances has been selected. The rest of inbound arguments different from the object-valued inbound argument with several selected instance will be equal in all calls.

If there is **more than one object-valued inbound argument with multiselection** enable, then the service will be **executed for Cartesian product** of all selected instance of these inbound arguments

The different executions of the service will be completely isolate.

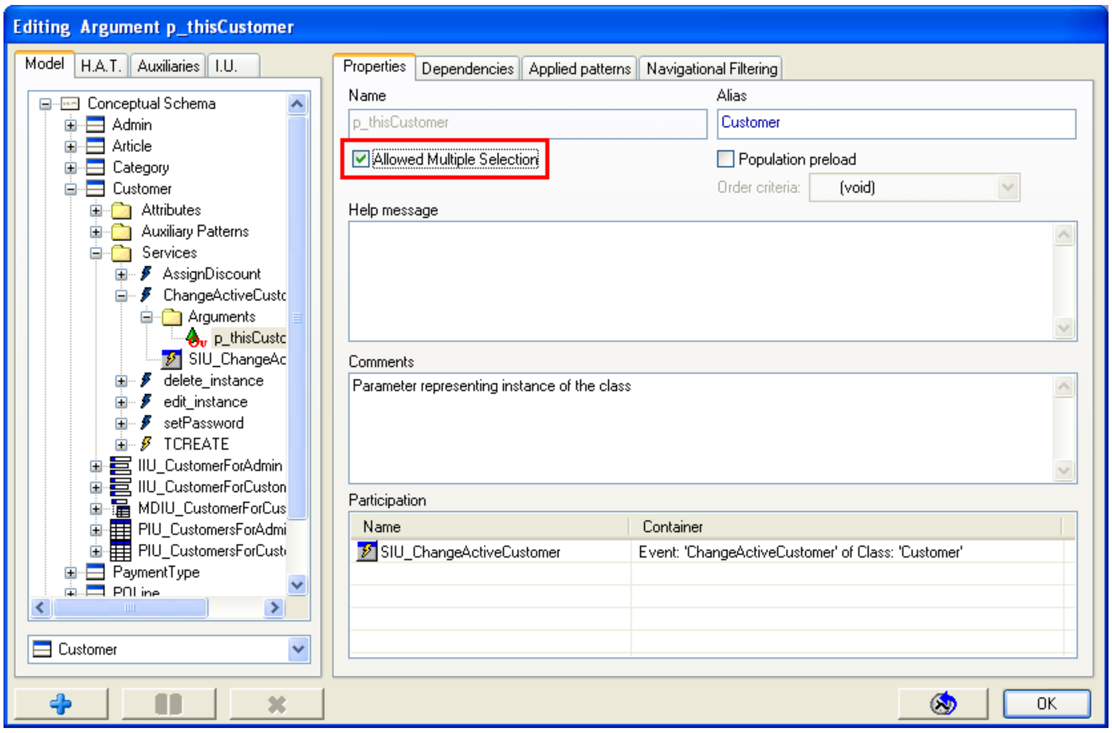
A typical example of using this feature is to delete several articles or enables or disabled many customers. But, there are other cases that multiselection has no sense. For example, for editing information about customer has no sense multiselection since each customer has its own personal data.

This feature is defined in Presentation Model as a property in object-valued inbound argument named as ***Allowed Multiple Selection***.

6.1 Defining Multiple Selection

We are going to mark the 'p_thisCustomer' argument as *Allowed Multiple Selection* in the 'ChangeActiveCustomer'service of the 'Customer' class.

Access to the Presentation Model dialog pressing on *Presentation Model* button. After that, select *Model* tab and open the *Conceptual Schema* folder. Then, the list of the classes appears. Select the 'Customer' class and go to the *Services* folder. Next, choose the 'ChangeActiveCustomer' event and open the *Arguments* folder. Finally select the 'p_thisCustomer' argument and pick the *Allowed Multiple Selection* check.



Editing Argument p_thisCustomer

Model | H.A.T. | Auxiliaries | I.U.

- Conceptual Schema
  - Admin
  - Article
  - Category
  - Customer
    - Attributes
    - Auxiliary Patterns
    - Services
      - AssignDiscount
      - ChangeActiveCustc
        - Arguments
          - p_thisCustc
        - SIU_ChangeAc
      - delete_instance
      - edit_instance
      - setPassword
      - TCREATE
    - IIU_CustomerForAdmin
    - IIU_CustomerForCuston
    - MDIU_CustomerForCus
    - PIU_CustomersForAdmi
    - PIU_CustomersForCust
  - PaymentType
  - POLine

Customer

Properties | Dependencies | Applied patterns | Navigational Filtering

Name: p_thisCustomer

Alias: Customer

[✓] Allowed Multiple Selection

[ ] Population preload

Order criteria: (void)

Help message:

Comments: Parameter representing instance of the class

Participation

Name	Container
SIU_ChangeActiveCustomer	Event: 'ChangeActiveCustomer' of Class: 'Customer'

OK

Figure 14 Properties allowed multiple selection window

Now, we have activated the multiselection in the 'ChangeActiveCustomer' service.

7 Showing the result after executing a service

When we launch an important service or transaction, normally we want to know if the execution of this service or transaction is OK.

In our tutorial, we have an important transaction called 'TPAY' that has defined an outbound argument called 'oa_message' that show a confirmation message after the execution of the transaction. When we define an outbound argument not ever we want to show its value to the user, perhaps you want use it for other purposes. For example, outbound arguments can be also used to initialize a service in a conditional navigation, that is a concept that we will see further in the Tutorial 15.

Don't worry about it for the moment, the only thing we want to do now is to choose whose arguments we want to show when the execution of 'TCHECKOUT' transaction is finished.

7.1 Defining how to show (or not) outbound arguments

Well, the mechanism that Integranova brings us to show or not the outbound arguments can be defined in the *Outbound Arguments* tab.

To access to *Outbound Arguments* tab, open to the Presentation Model dialog clicking on *Presentation Model* button. After that, select *Model* tab and open the *Conceptual Schema* folder. Next, select 'PurchaseOrder' class and after that, select the *Services* folder. Then, choose 'TCHECKOUT' service and select its Service Interaction Unit related named 'SIU_TCHECKOUT'. Finally, select the *Outbound Arguments* tab in the right area where you can see the outbound arguments that you've modeled previously, in the beginning all the outbound arguments are visible.

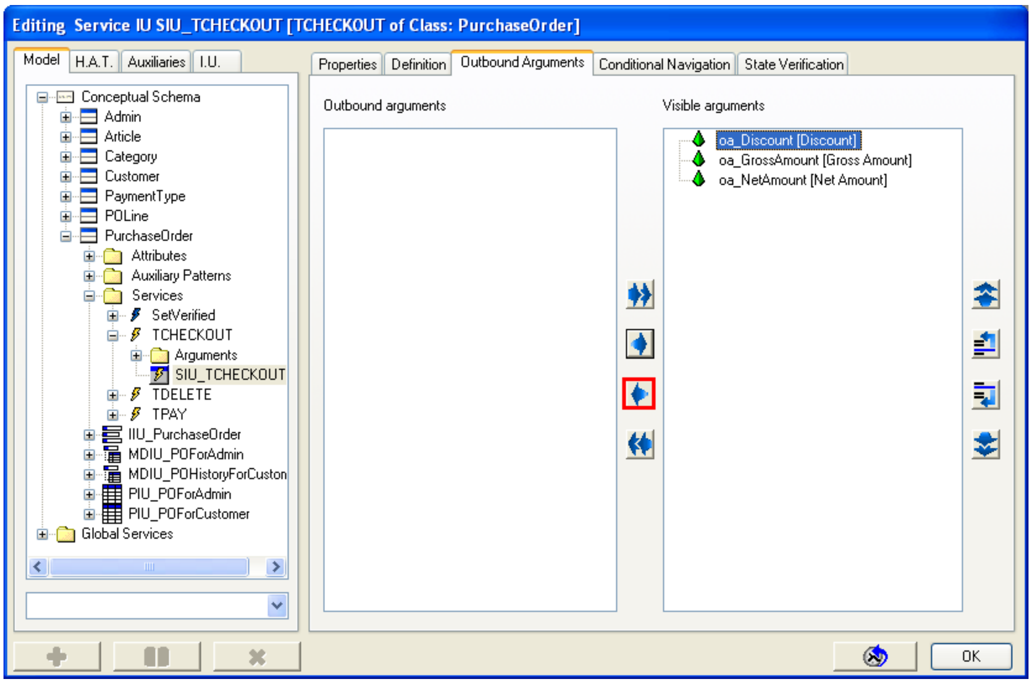


Figure 15 defining outbound arguments to show

In our tutorial, we want that the Discount will not be shown to the user application.

Select the ‘oa_Discount’ outbound argument and press the *left arrow* button. This argument will be not shown when the execution of the service finishes. Using the *move up all*, *move down all*, *move up* and *move down* buttons, you can order the outbound arguments that will be visible at the end of the execution in the order that you prefer.

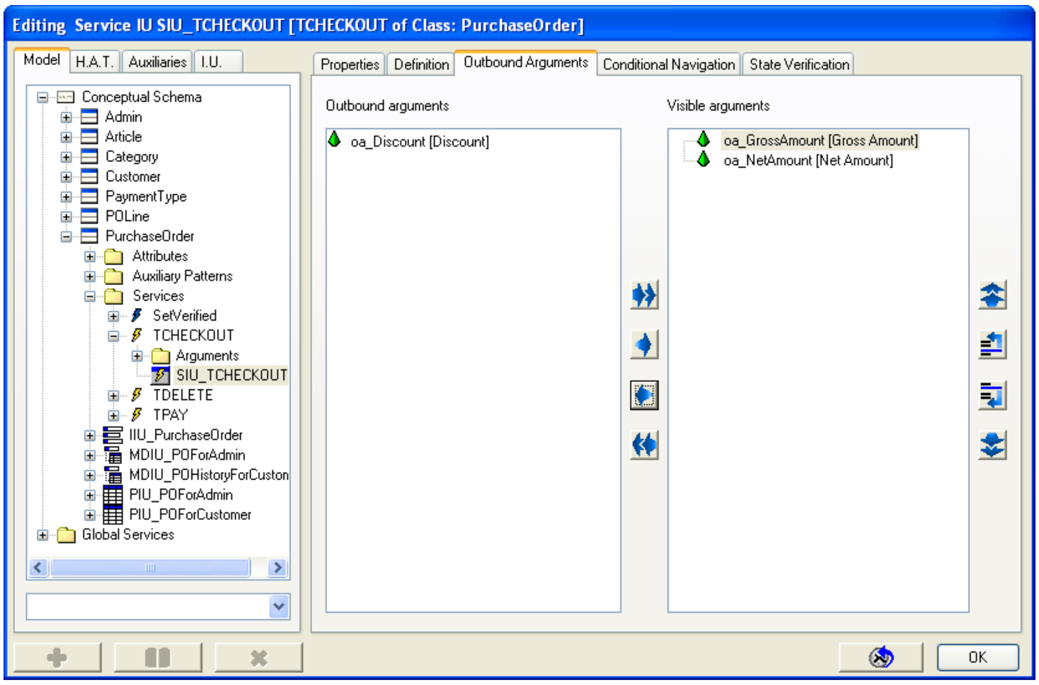


Figure 16 defining outbound arguments to show

Then, in run time, the application only will show two arguments: ‘Gross Amount’ and ‘Net Amount’.