DEFINING THE USER INTERFACE

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 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 08, with this new tutorial that will build on the previous one.

So far, we have learnt how to model some of the functionality that offers Integranova. We are now going to focus on presentation matters, so introducing the reader to the **Presentation Model**. In this tutorial, the basic concepts for defining the User Interface as well as how to model some interaction unit scenarios will be introduced.

By the end of this tutorial you will have learnt how to define which elements will take part of your system (will be the final application) and also which are the basic interaction units provided by default.

In this tutorial we will start from the model we have done in Tutorial 08.

2 Introduction to the concept of View and Interface

In previous series we learnt the concept of Agent classes. We saw how agent permissions can be granted over other classes. These permissions were related to the services from a server class that an Agent class may execute, the attributes of that server class that can be seen by the Agent and also the roles that can be accessed to.

The set of all these permissions established between a server class and an agent class is called **Interface**.

Therefore, a **View** is just a set of Interfaces.

A **View** defines, first of all, the collection of agents that can connect to the system (can login) and their visibility. Since a view is a union of interfaces, the visibility of a view at run-time is restricted to the connected agent class interfaces, not to the rest of the interfaces of other agent classes.

Having all of this in mind, we can conclude that a View will be the final application. We can also have more than one View defined for the same model, but we will see this further on the next series.

3 Creating a View for the Administrator Agent profile

In previous series, we have been identifying some agent profiles: anonymous user, customers and of course, the administrator. We have also been defining agent permissions for these agent classes. Now, we are going to create a View for the 'Administrator' agent class. The same View could be used at the same time by different agents, but in this case, we are going to define a View exclusively for the Administrator.

The 'Administrator' agent is almost agent of everything in our system. This makes sense as its main function is to manage and maintain the background of the system (being in charge of article's tasks, purchases, payments, categories and so on…).

With all this in mind, that's the reason why we are going to define our first View which will be used by Administrator's users.

Let's go to do that in the model. We open the model and go to the *Edit* menu and select *Views*. As you can see, initially there is no View created yet:

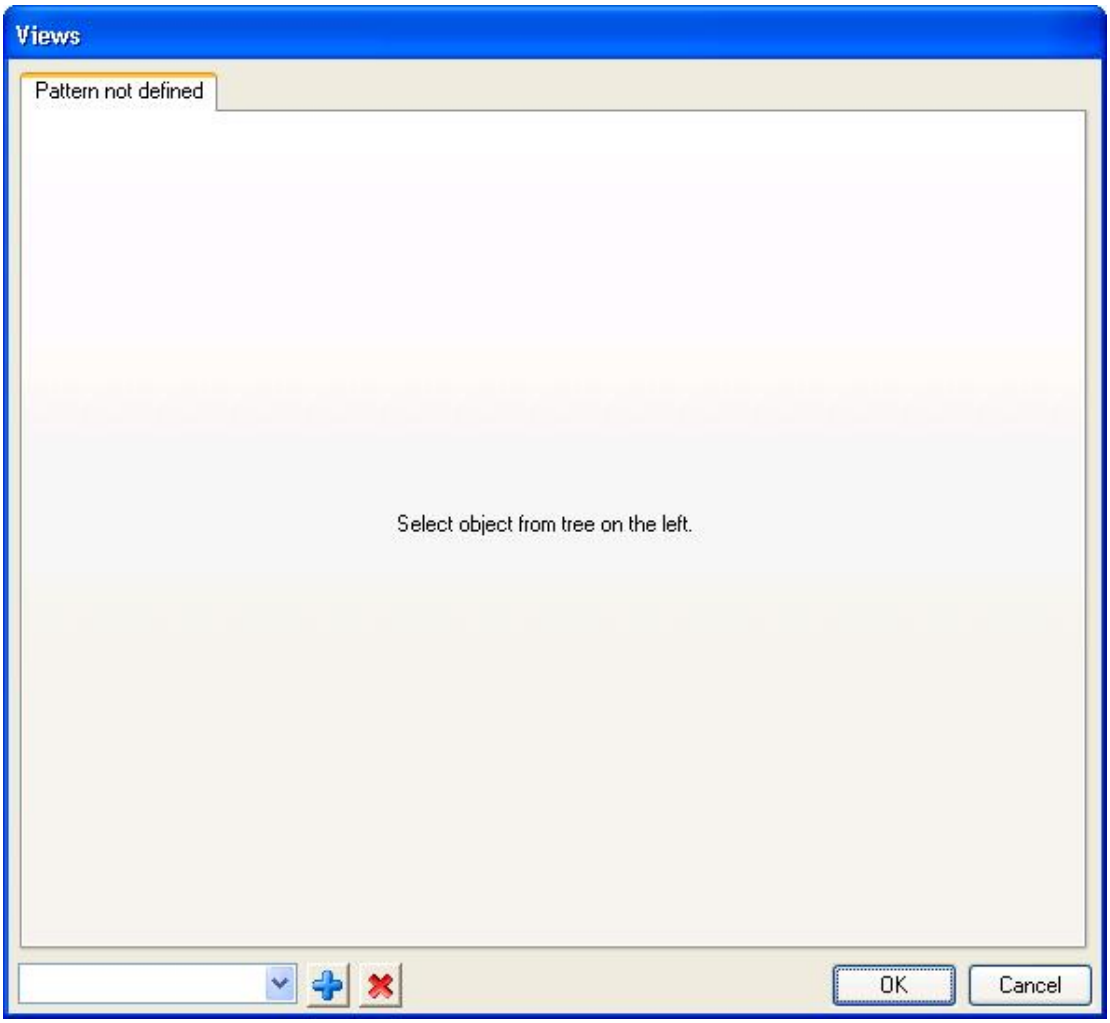

Figure 1 View's dialog

Then, we press on the *Add*  button as to add our first View:

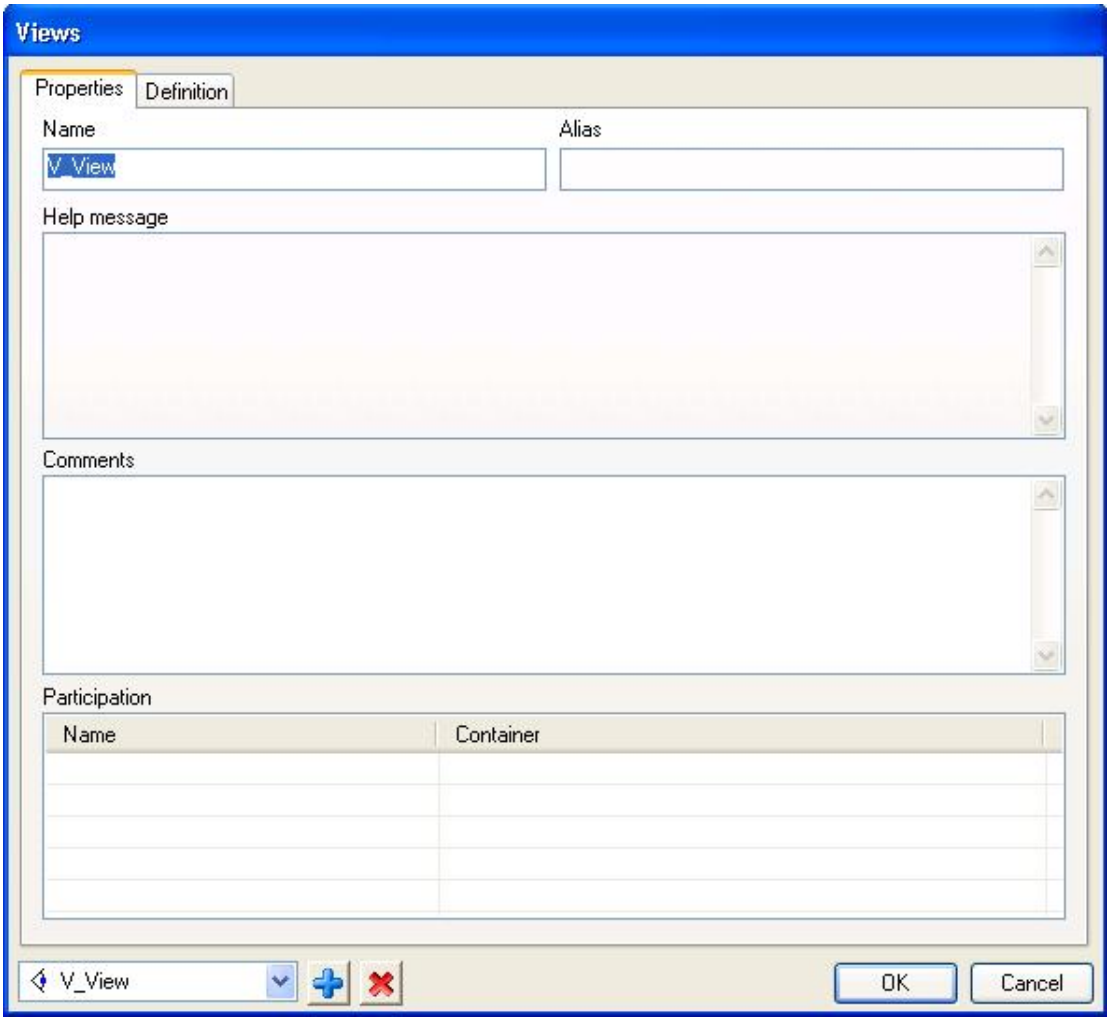

Figure 2 Creation View dialog

In the dialog above, we can see we are offered a name by default, 'V_View', we can change it to 'V_Admin', for instance, so as to better identify it later on in the model.

After changing the name, we go to the *Definition* tab. That's where we are going to select which agent interfaces will be included to the View. So, click on the *Definition* tab:

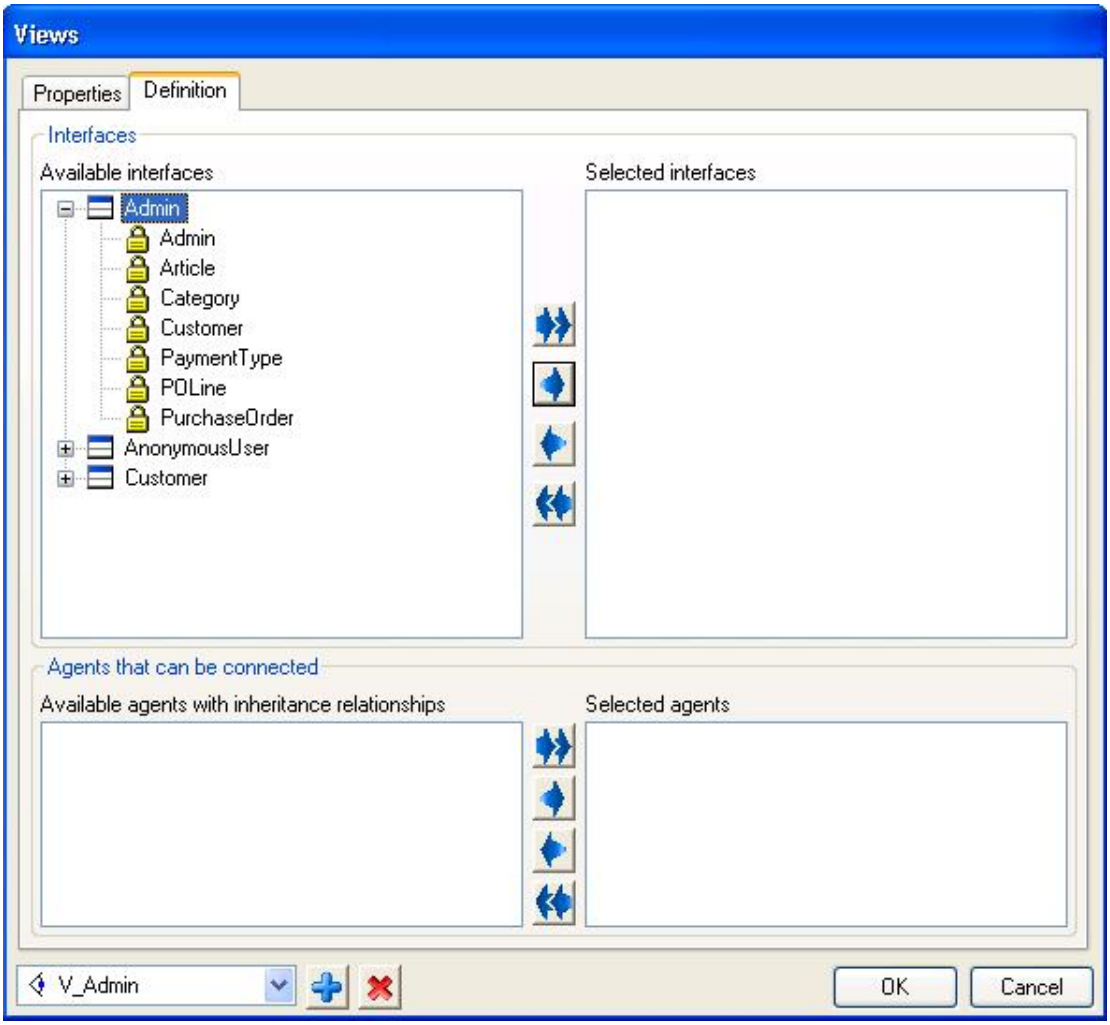

Figure 3 Creation View dialog. Definition tab

Let's comment the screenshot above. Initially we can see the Agent classes we have defined in the model, in a sort of tree graphical representation in the *Available interfaces* list box. If you expand one of these agent classes, it will show its interfaces (set of agent permissions over classes of the model). In this case, we just want to define a View for the 'Admin' agent class, so once expanded the 'Admin' class, we can select which interfaces we will incorporate in the View, adding them to the right side. As we want the Administrator to be able to do almost anything, we are going to add all its interfaces.

Then, move all the 'Admin' interfaces to the right side, and our first view 'V_Admin' will be ready:

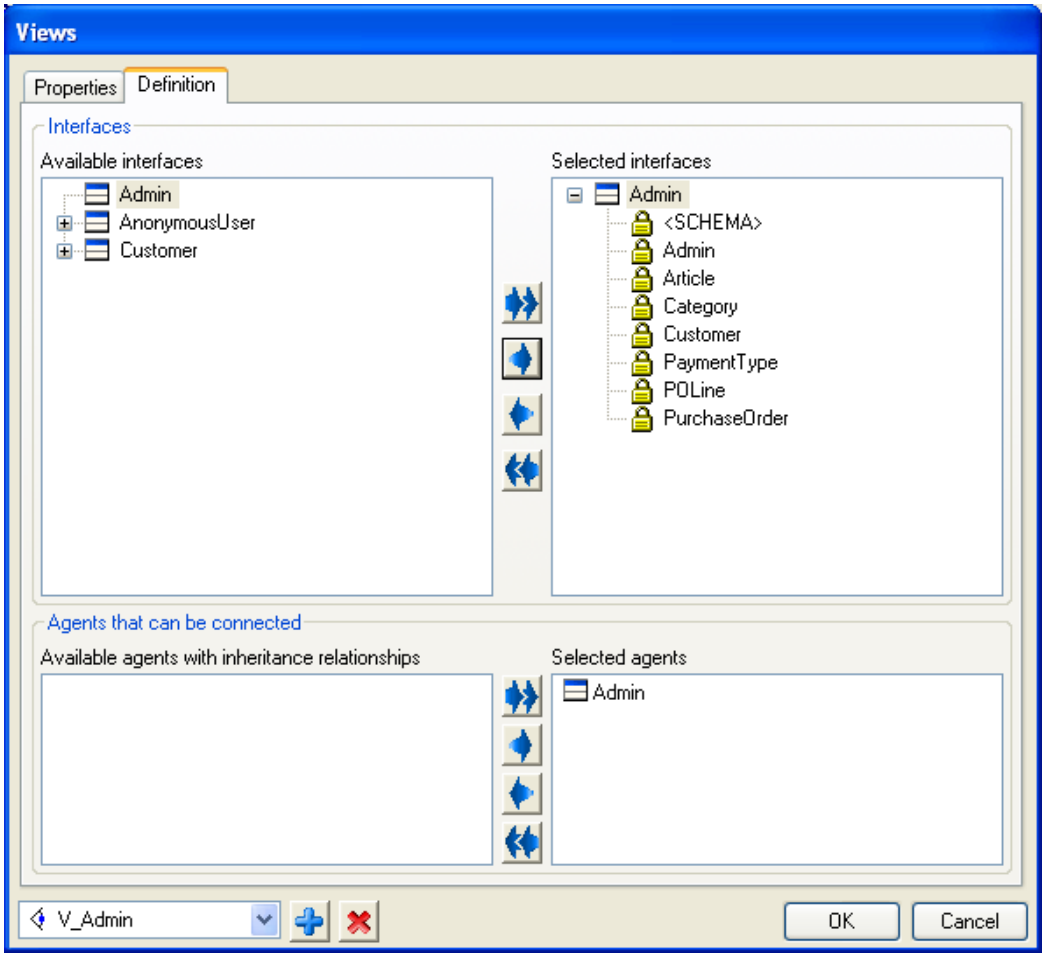

Figure 4 Adding Interfaces to the View

Now the 'V_Admin' is ready.

Note: At the time of transforming the model, when doing so through the *STAR Client* tool, you have to choose this view in the star option client profile. You could go back to the Tutorial 0 of the series as to remember how to do so.

4 Introduction to the Interaction Units

The concept of Interaction Unit comprises the basic specification element for the interaction between the user and the software system. An Interaction Unit describes a particular scenario of the User Interface through which users are able to carry out specific tasks (such as executing services or searching for objects).

Integranova provides four different types of interaction units, which represent four different basic kinds of interaction scenarios:

- ✓ Execution of a service –> **Service Interaction Unit**
- ✓ Manipulation of one object -> **Instance Interaction Unit**
- ✓ Manipulation of a collection of objects -> **Population Interaction Unit**
- ✓ Manipulation of multiple related collections of objects -> **Master/Detail Interaction Unit**

4.1 Interactions through services' execution

We are going to briefly see an example of a Service Interaction Unit. These types of interaction units as well as the rest of interaction unit types mentioned above are defined in the Presentation Model.

We can access to the Presentation Model both by:

✓ Clicking on the toolbar *Presentation Model*  button:

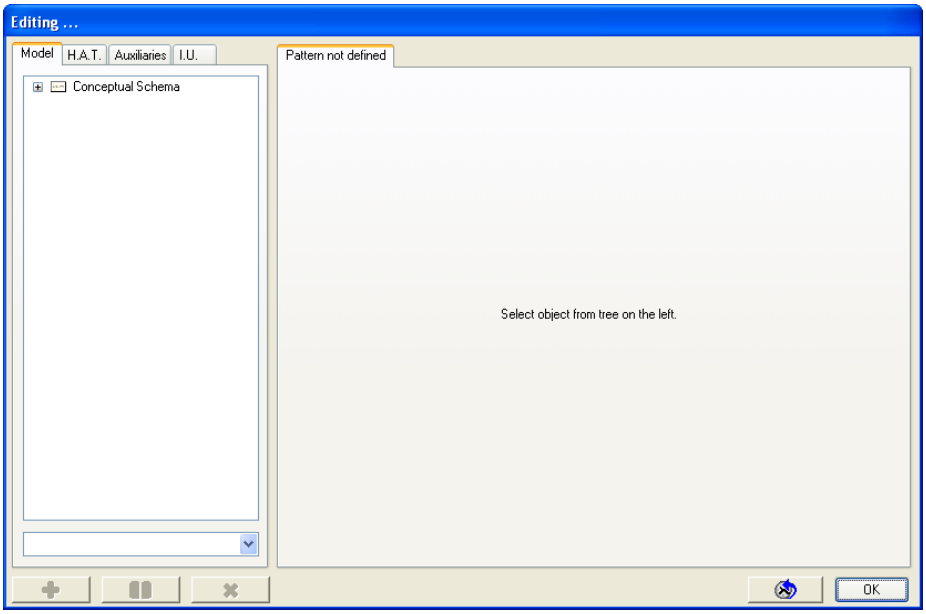

Figure 5 Presentation model opened first time

✓ Or right clicking on a class and selecting *Presentation Model* option

So let's see for example the Presentation Model accessing from 'Customer' class. We right click on 'Customer' class and select *Presentation Model* option:

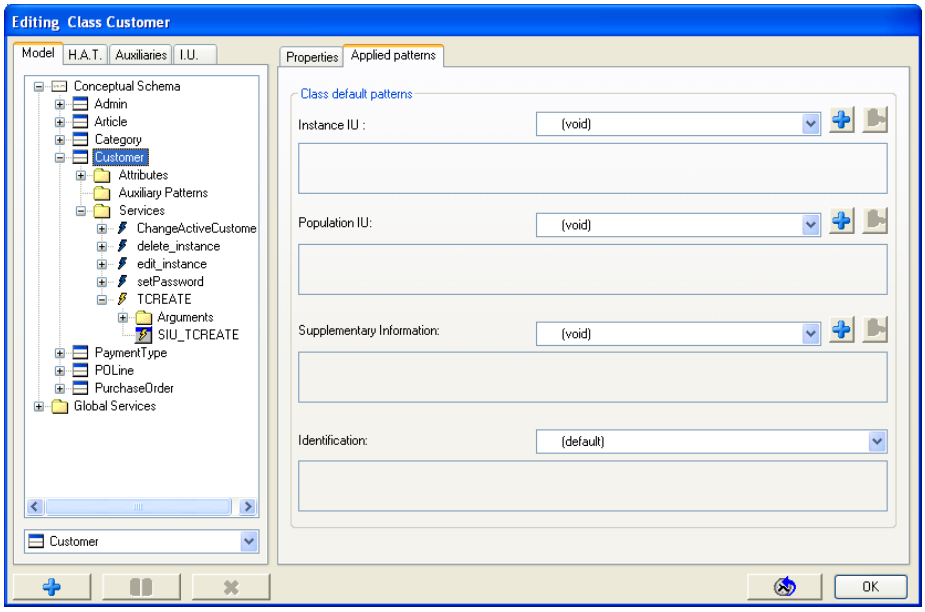

Figure 6 Editing 'Customer' class in Presentation Model window

As we can see, the Presentation Model for 'Customer' class is empty due to we have not defined anything yet. So the text "(void)" appears in the *Applied patterns* tab on the right side.

However, if we expand the *Services* folder, we can see there are some services inside. These services are those services we have been modeling so far and they are not marked as internal. So for each service, there is a Service IU associated and automatically created by **Integranova Modeler**. In the Figure 6, we can see the corresponding Service Interaction Unit for the *TCREATE* service.

If we click on the 'SIU_TCREATE' interaction unit, we will access to the Service Interaction Unit associated to the 'TCREATE' service of 'Customer' class. There, we can see some information which is related to a Service Interaction Unit:

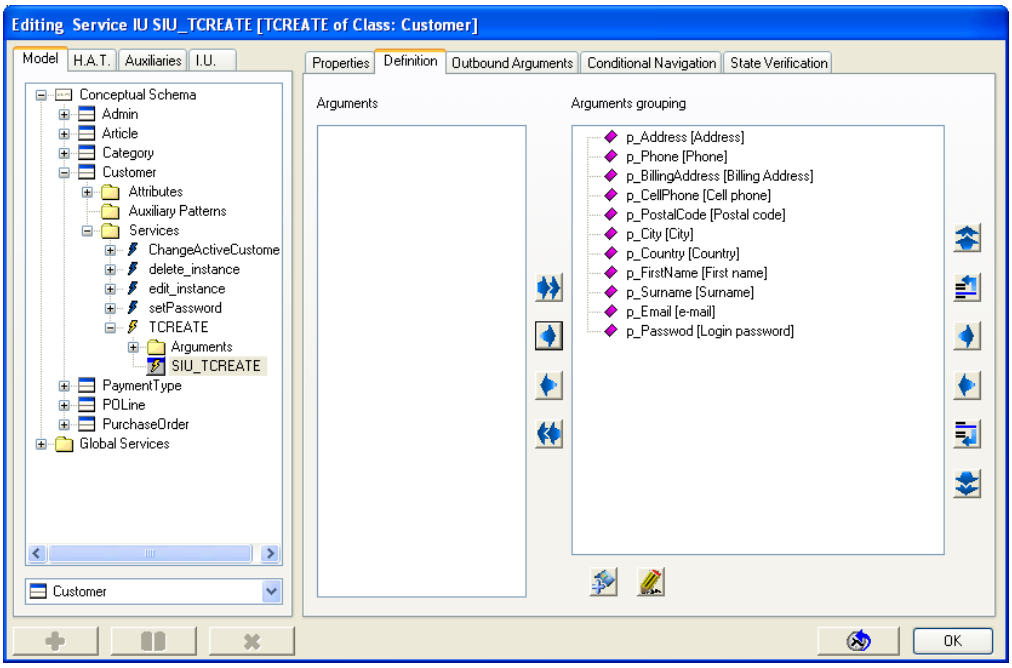

Figure 7 *Definition* tab of SIU_TCREATE Service Interaction Unit from 'Customer' class

You can see the Service Interaction Unit's related scenario, which have some tabs. The *Properties* tab just shows the name of the service and allows changing the name, the alias of the Service Interaction Unit and adding some comments and so on… The rest of the tabs: *Outbound Arguments*, *Conditional Navigation* and *State Verification* will be further explained on the next series.

Let's then focus on the *Definition* tab, which is shown in the Figure 7. In this tab we can see all the inbound arguments defined in the Service Interaction Unit with their alias (shown between brackets). For now, we'll just bear in mind this. In the next series, we will learn how to change the order of these inbound arguments, change the aliases and also group them.

Taking advantage that we are in the Presentation Model, seeing what Service Interaction Units are and some of their properties, let's introduce the concept of *Status Recovery*.

The *Status Recovery* is an option offered to edition type services (neither creation nor destruction) that infers values of the instance to be executed initializing the service. In other words, the values are gotten from the attributes of the instance and initialize the arguments of the service. This is a kind of wizard functionality that **Integranova Modeler** offers on the analyst demand.

That way, if the final application user needs to edit any instance, when launching that edition service, the information is initially presented in the Service IU, with the current values of the instance attributes that will afterwards get the value from those inbound arguments of the interaction unit.

Let' see how we can do it. In the Presentation Model, you just select the edition service required, right click on it and select *Status Recovery* option:

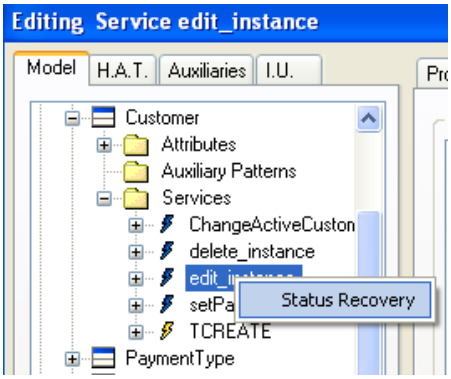

Figure 8 Status Recovery

5 Interactions through Population Interaction Units

When starting to define the interactions with the system, a good starting point could be defining a Population Interaction Unit. This kind of scenarios are thought to show the final application user the instances of the current class in which the Population Interaction Unit is defined.

Note: The set of instances are restricted to the ones that the logged agent can see (see Horizontal Visibility in Tutorial 03).

Following our tutorial, we could define such interaction unit over 'Customer' class. At the time we do so, we will see some auxiliary patterns that can be defined for these types of interaction units.

Let's then create our first Population Interaction Unit. Being in the Presentation Model window, the same way we did previously, now we go to *I.U.* tab:

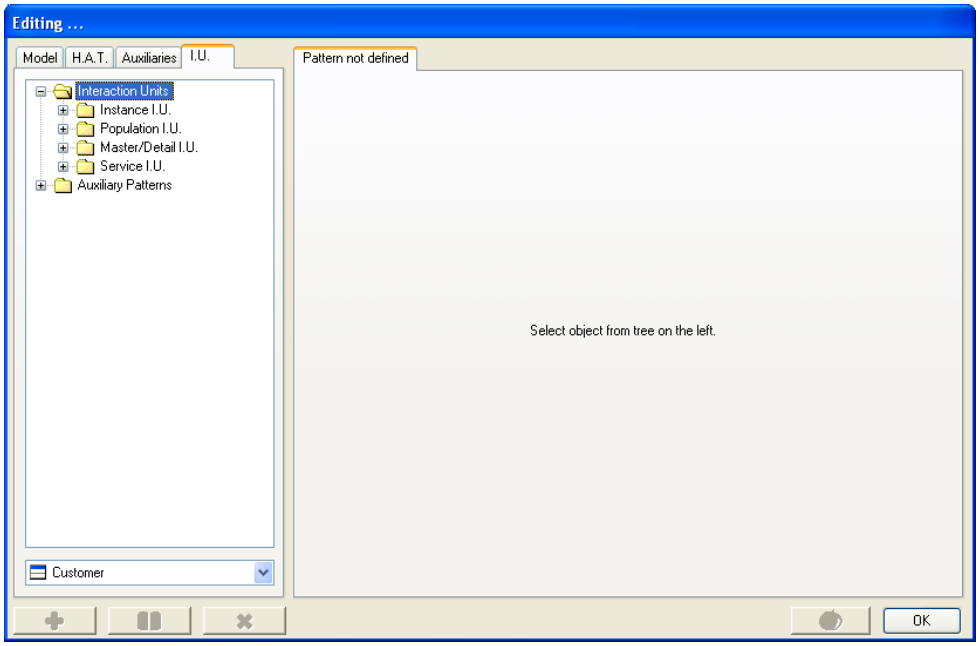

Figure 9 *I.U.* tab in Presentation Model window

As you can see, we can choose from different kind of interaction units. For each folder, if we expand it, we can see all the classes of our model. Let's expand the 'Population I.U' folder and click on 'Customer' class:

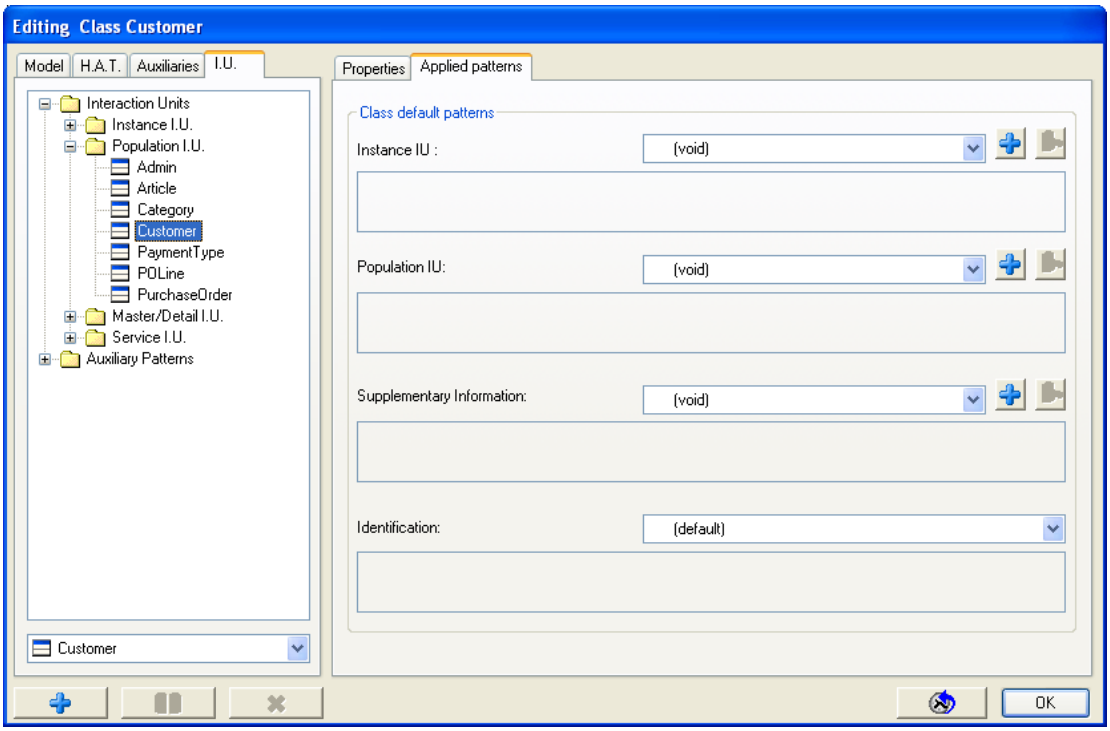

Figure 10 Presentation Model I.U tabs

Initially the 'Customer' class is empty, it doesn't contain any interaction unit; the items (Applied Patterns) in the right side are "(void)".

We would like to create a Population Interaction Unit, then, having selected the 'Customer' class, we go to the *Population Interaction Unit* combo box in the right side and press the *Add*  button:

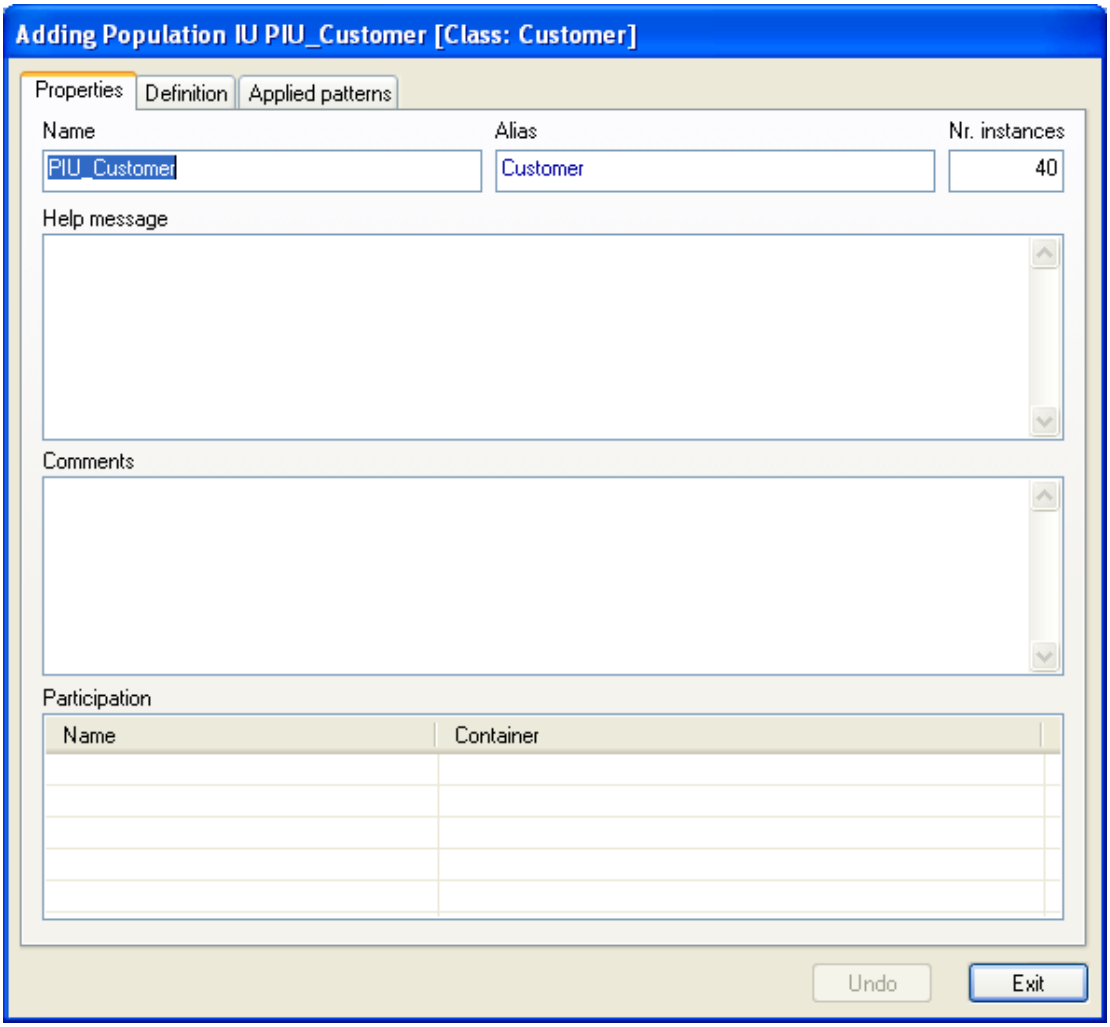

Figure 11 Adding a Population Interaction Unit

We have a Population Interaction Unit name by default 'PIU_Customer' that we can change anytime for a more accurate one. Furthermore, the alias is inferred from the alias of the class. We can also choose the number of instances that will be shown (per pager block). By default are 40. Comments and help message are also defined here.

Now, move to the *Definition* tab:

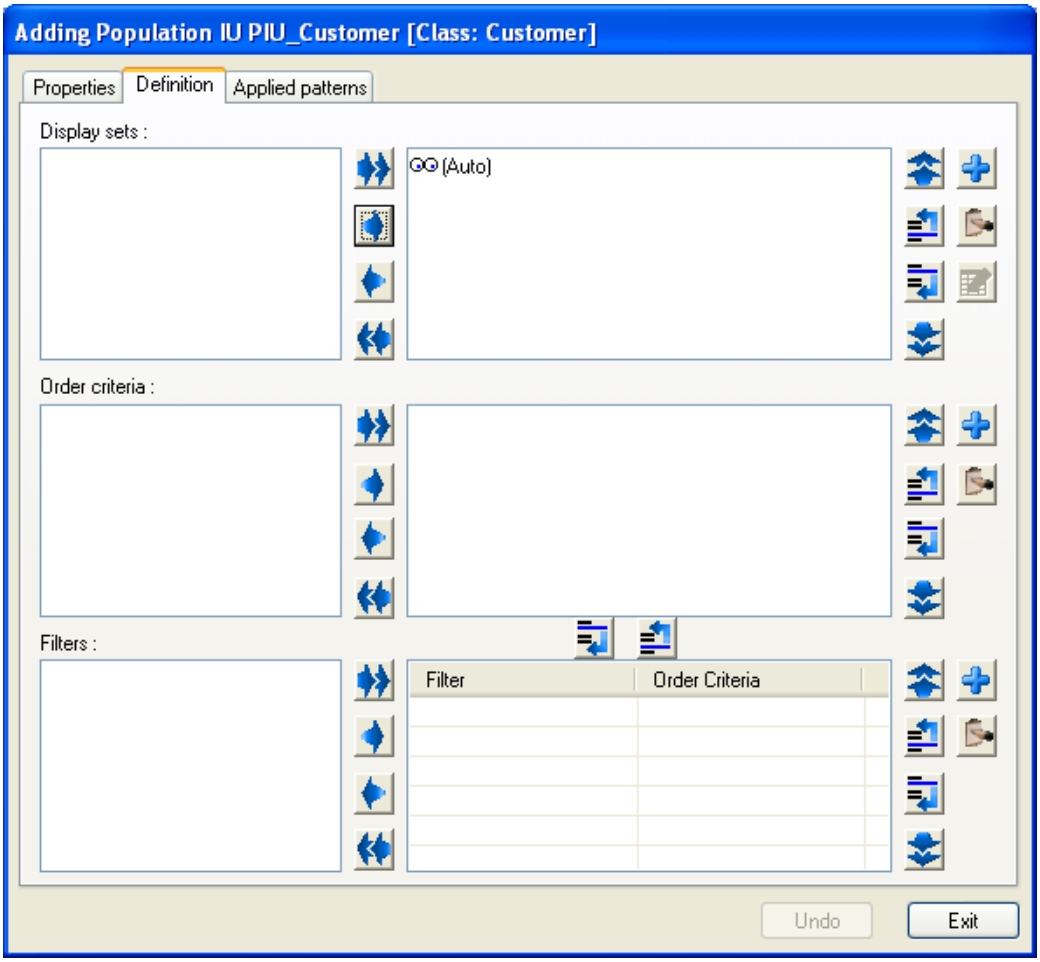

Figure 12 Add a Population Interaction Unit, Definition tab

In this dialog tab, it is possible to add a *Display set* item, *Order criteria* and *Filters*. A *Display set* allows specifying which attributes will be shown in the Population Interaction Unit. For instance, talking about customers, we could say: Name, address, phone, etc.

An *order criterion* defines the attribute or attributes from which these instances shown in the Population Interaction Unit will be ordered. And a *Filter*, as its name states allows filtering the information through concrete filtering items.

For now, we are going to leave this as it is by default. As you can see we are offered with a Display set in an (Auto) mode, which means that all the attributes defined in that class will be shown. In the next series we will further see how to define display sets as required. *Order criteria* and *Filters* are left empty that means that the instances will be shown following the identification order and no filters will appear and all the instances will be shown.

Well, let's now move to the *Applied patterns* tab:

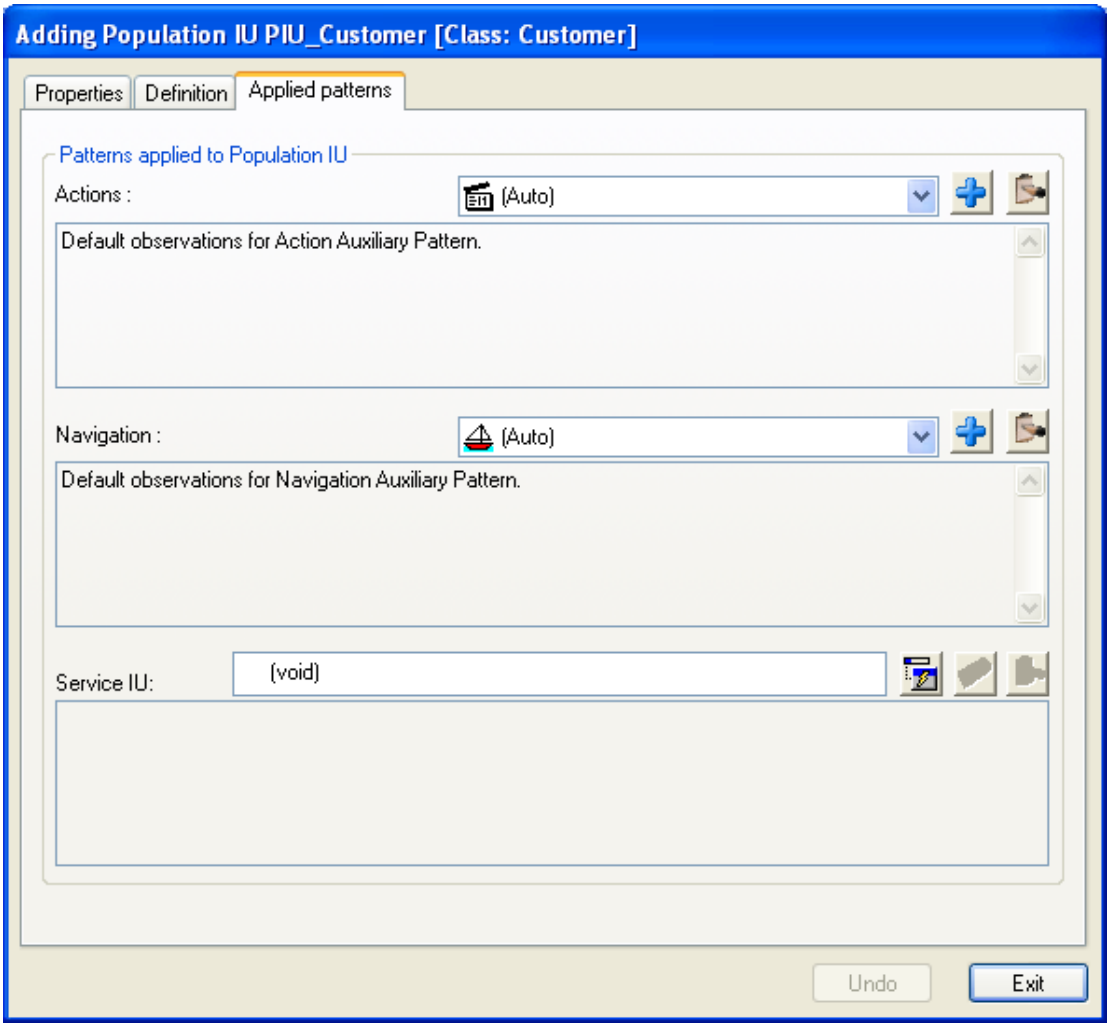

Figure 13 Adding a Population Interaction Unit, Applied patterns tab

Following with the definition of the Population Interaction Unit, we see in the *Applied patterns* tab that we can define *Actions*, *Navigations* and associate a *Service Interaction Unit* to that Population Interaction Unit we are about creating.

Actions are patterns that gather all the services we want this Population Interaction Unit to offer. For example, an edition service that allows changing some personal data of a customer. In the same way, *Navigations* gather these destination interaction units that can be accessed from this Population Interaction Unit. For instance, navigation to the purchase orders of a customer.

Again, alike in the *Display set*, we are offered with an (Auto) option for these patterns. In *Actions*, (Auto) option will show all the non-internal services defined in the class that the Population Interaction Unit belongs to. In *Navigations*, (Auto) option will allow accessing to each one of the related instances through a role path. We will leave them as they are and in the next series we will further learn how to define them as required instead of as default.

6 Defining a menu for the application

We can define a **menu** for our View 'V_Admin' through the hierarchical action tree (HAT). Following with the possibilities offered by Integranova, we can also create a HAT by means of a wizard. If we use that wizard, we would have a menu automatically created.

In the next series, we will further see and learn how to customize a HAT (menu) so we can design it as required or wanted. In the meantime, and finalizing with this tutorial of the series, we will learn how to create a menu by means of the wizard. Then, we go back to the Presentation Model window of our model and go to the *H.A.T.* tab:

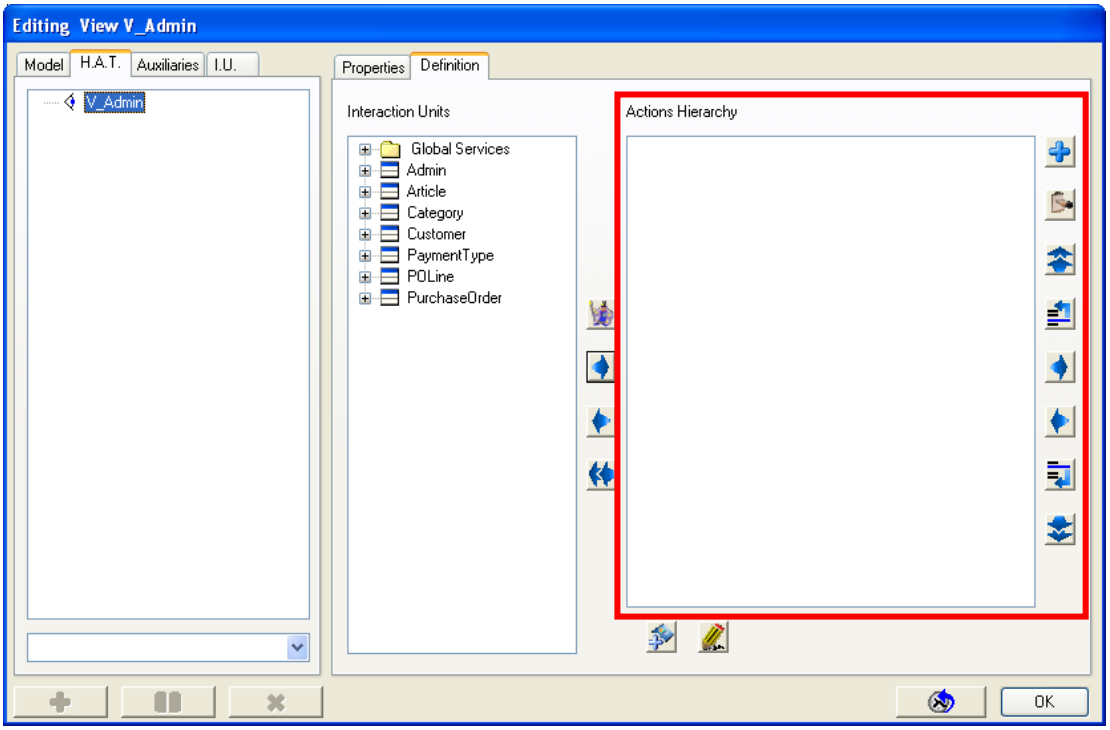

Figure 14 Hierarchical Action Tree. A menu for the application

As we can see in the Figure 14, the 'V_Admin' view has not any HAT defined yet. In the middle area we can see the classes of the Interfaces added in the View. We did that at the beginning of this tutorial series.

So the easiest way to define a HAT would be to use the wizard as to create a menu automatically. Then, just click on the *Wizard*  button and let's see what happens:

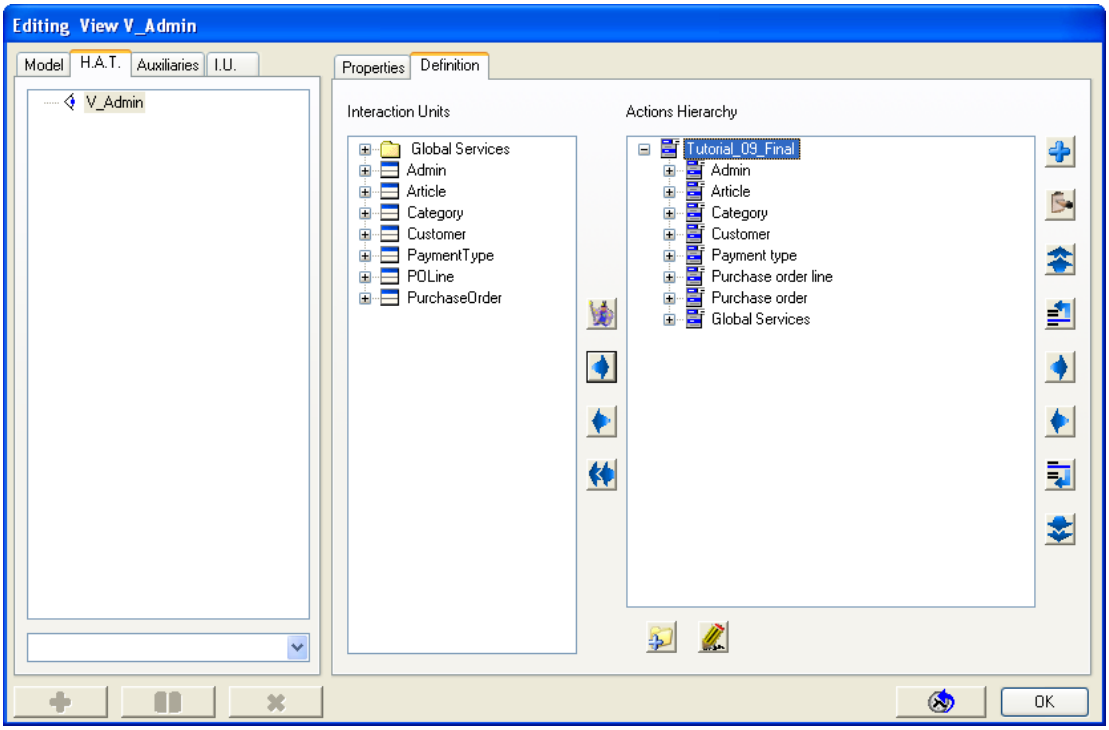

Figure 15 HAT automatically created using the wizard button

The result of clicking on the *Wizard* button is showed in the Figure 15. We have provided with a menu for our application, with all the elements added to it. It is built using all the Unit Interaction defined in the Presentation Model.

The offered name for the root of the menu (Tutorial_09_Final) can obviously be changed by a more accurate one. Having the root selected, we click on the *Modifying hierarchy node*  button and we change the HAT root name to 'Admin'.

As previously said, this can be changed as required, adding or removing elements as well as creating new nodes. We will see how to do it in the next series.