CREATING A CLASS AND ITS ATTRIBUTES

SAMPLE: SHOPPING CART

Last Update: June 2010

Table of Contents

1 Goal

In this tutorial we walk 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.

This is the first in a series of tutorials that walk through the capabilities of Integranova. Each tutorial builds on the previous one by adding more requirements to the sample application to be developed and showing how to model said requirements.

By the end of this tutorial you will have developed an application to manage a catalogue of articles. Said catalogue will store information about each article in the company (article id, name, creation date, price and availability) and will also provide the basic services for the creation, edition and deletion of articles.

Given the simplicity of this first sample, we will not care about who can log on to the application right now, although this will be addressed later in the series.

2 Starting up Integranova Modeler

When you start **Integranova Modeler**, the Class Diagram is shown by default with an empty work area.

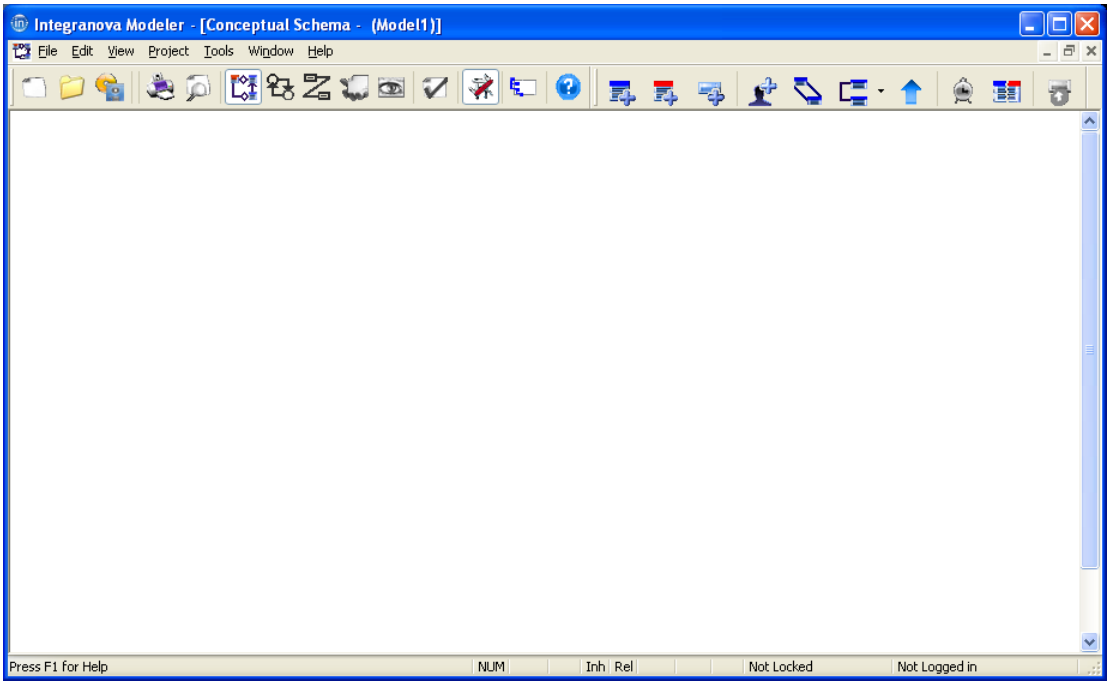

Figure 1 Integranova Modeler with an empty work area

If the toolbar does not include the Drawing toolbar, which you will need to start creating elements on the work area, click the *Drawing toolbar*  icon to augment the main toolbar with the elements of the Drawing toolbar.

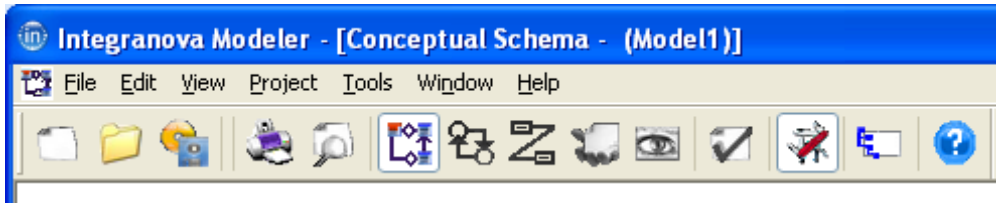

Figure 2 Integranova Modeler toolbar

The Drawing toolbar is added to the main toolbar with these elements:

Figure 3 Drawing toolbar elements

3 Creating the 'Article' class

The first thing we need to do is create something in our model that will represent the articles in our catalogue: the 'Article' class.

A class is the basic modeling unit. We use classes to represent the entities in our system as well as to define the logic of the services we will be using to manipulate them.

To create the new 'Article' class, click the *Add Class*  icon from the Drawing toolbar then click on the (empty) work area. The *New Class* dialog prompts.

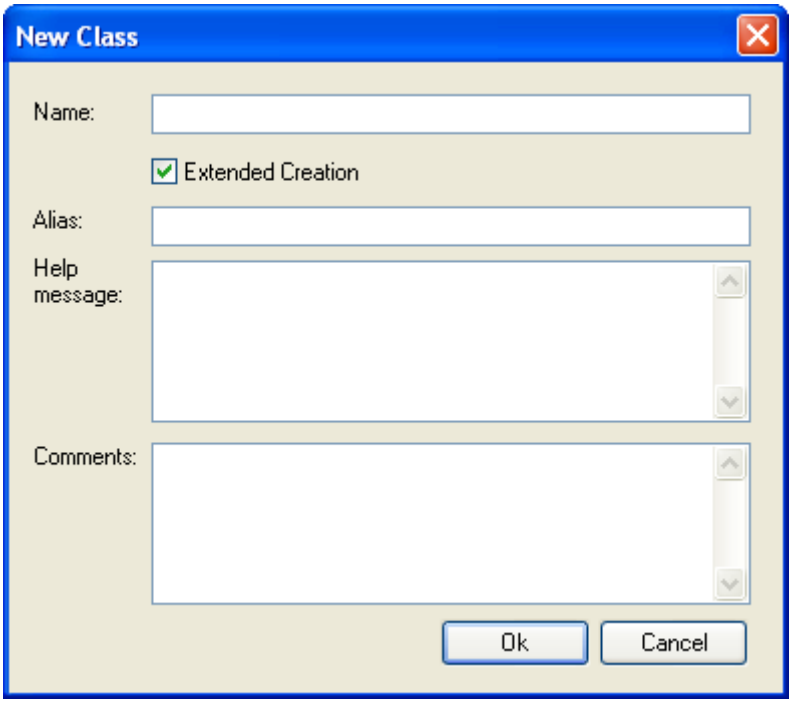

Figure 4 Creating a New Class

Fill in the fields in the dialog with the info depicted in Figure 5 below.

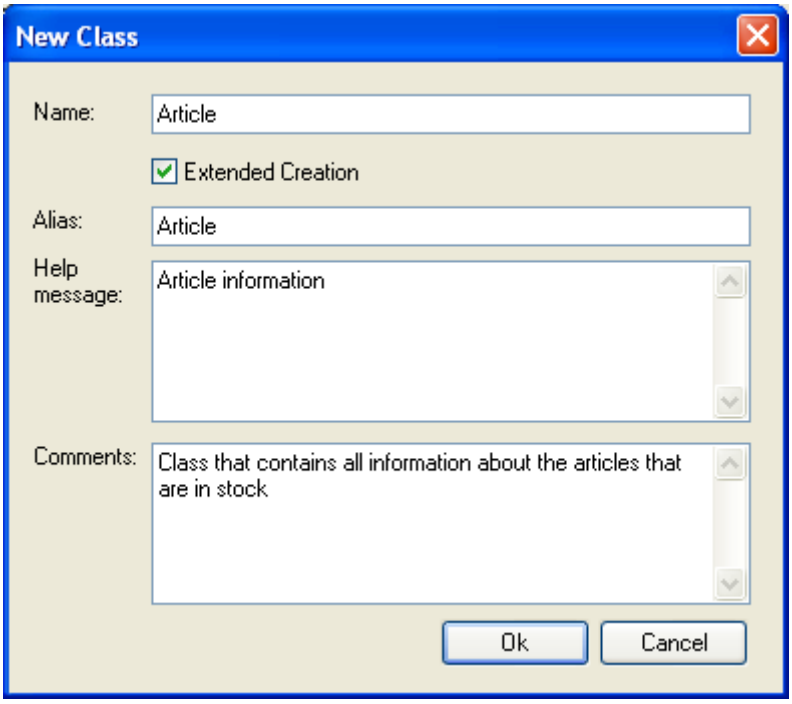

Figure 5 New Class dialog with information

- ✓ The *Name* field is the name of the class, and it must be unique in the model.
- ✓ The *Alias* field has the text by which this class will be referred to in the final user application.
- ✓ Use the *Help message* field to enter information that may be shown to users of the final application.
- ✓ The *Comments* allows you to document the purpose of the class. You will find "Comments" fields in many places in the model. This way you can document as you model and hence keep documentation up-to-date.

Press the *OK* button to close the dialog and see the 'Article' class in the work area.

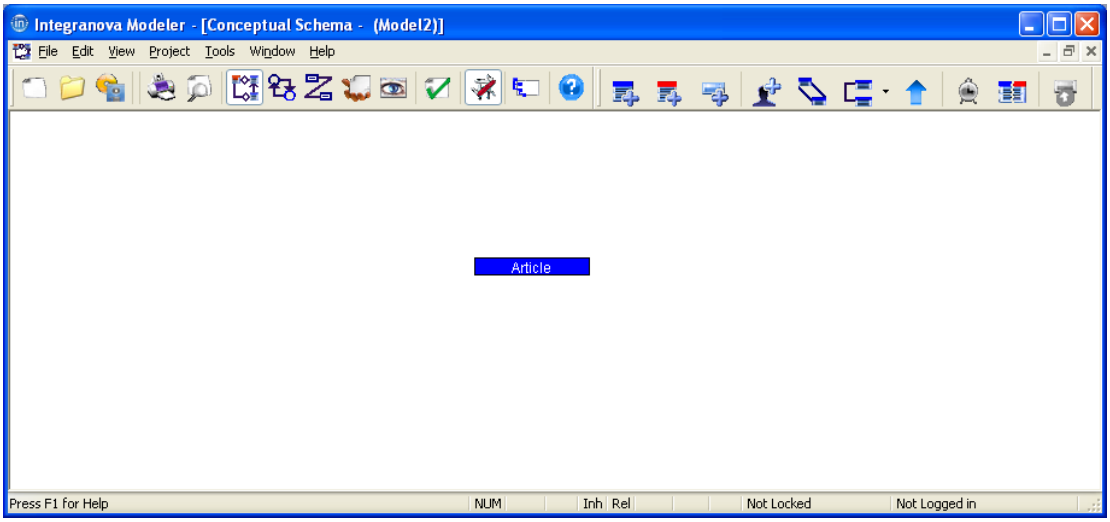

Figure 6 Work area with 'Article' class

4 Creating attributes for 'Article' class

Now that we have our 'Article' class in place, we will specify the relevant properties of articles: some kind of identifier, a name, the date when the article was added to the catalogue, the article price and so on.

What we need to add to our 'Article' class is some attributes, which are part of the structural properties of a class. Attributes have a *Name*, a *Type* (Constant, Variable, Derived) and *Data Type* (Autonumeric, Blob, Bool, Date, time, DateTime, Int, Nat, Real, String, Text), may or may not allow null values (*Null allowed*), and may or may not be requested upon creation (*Request upon creation*). Optionally, attributes may have a *Default value*, an *Alias*, a *Help message* and some *Comments*.

To start adding attributes, double click the 'Article' class on the work area and the *Class Edition Dialogue* opens showing several tabs, the first of which being the *Attributes* tab.

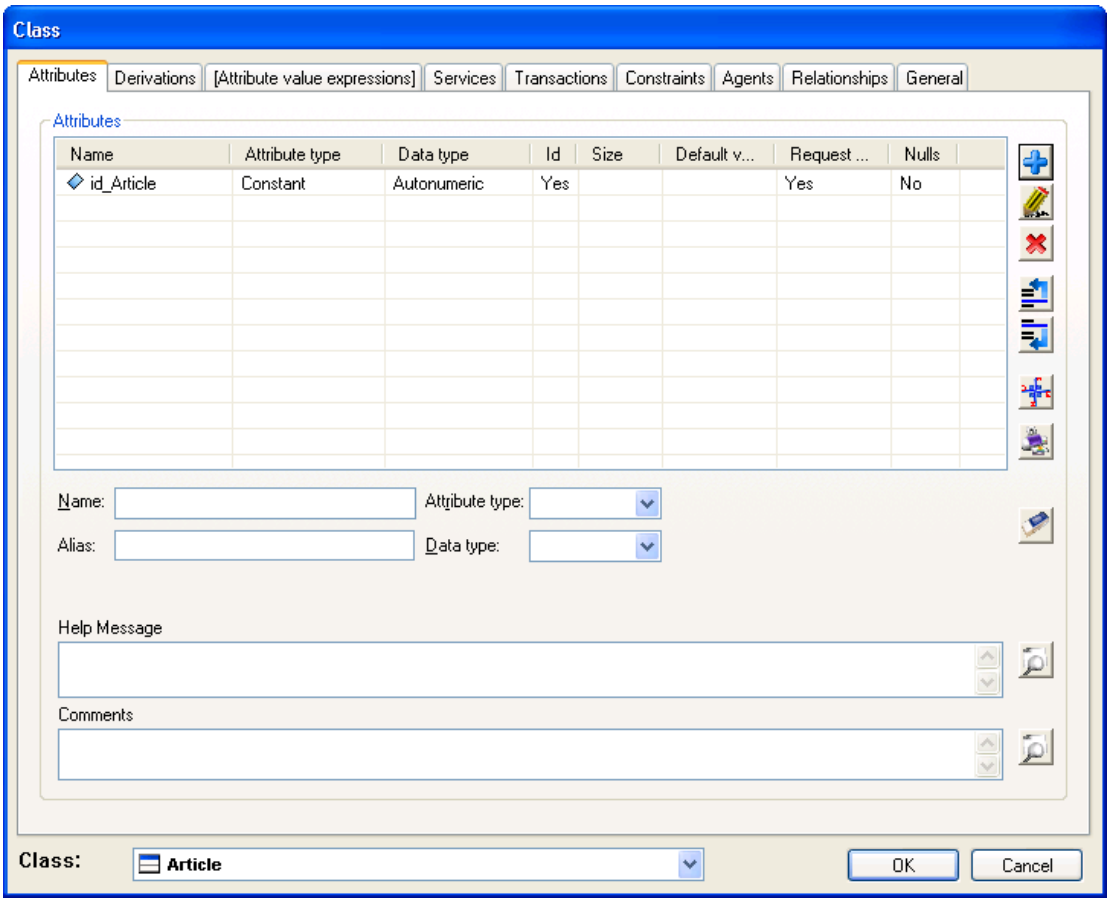

Figure 7 Class Edition dialogue

Notice that there is already an attribute in our class: 'id_Article'. Remember that we created the 'Article' class with the *Extended Creation* checkbox checked. One of the things this implies is that the tool will automatically add an attribute to the class and use it as its identifier. The attribute name is made by prepending 'id_' to the class name. As one would expect, this attribute is of Type *Constant* and is an *Id*.

Although the Data Type is set by default to *Autonumeric*, notice that the attribute is marked as *Request upon creation* (to allow a user of the system provide a value for this attribute upon creation of a new article). Since this is not the case (e.g. we want the system to automatically assign a numeric, unique value to identify the article) we will need to edit the attribute so as to change this.

In order to edit the 'id_Attribute', just double-click on it and notice that the fields on lower part of the tab are filled with the details about our attribute. Uncheck the *Request upon creation* checkbox and save your changes by clicking on the *Edit* button 

We need to name the articles in our catalogue. So we need a new attribute in the 'Article class' that keeps this data, which we will call 'ArtName'. We may want to change the name of our articles in the future (due to commercial reasons, and so on.) that's why we need it to be of Type *Variable*. It also makes sense to have *String* as its Data Type and we'll save a maximum length of 30 characters for it, which should be enough to store the name of each article. This attribute will not allow null values (it will be mandatory that every attribute has a name) so we uncheck the *Null allowed* checkbox. In addition to this, the article name value will be asked to the application user at creation time so we are going to leave the *Request upon creation* checkbox checked. And, as we need to change its value eventually, we make sure we check the *Add to edit* checkbox. Finally, we are going to display this attribute with the *Alias* 'Name' instead of using 'ArtName' (which is the attribute).

Fill in the fields on the lower part of the tab with the information summarized in the table below:

Table 1 Attribute Name of Article

Name	Alias	Type	Data type	Default value	Null allowed	Request upon creation	Add to edit event	Help message	Comments
ArtName	Name	Variable	String – 30	–	X	V	V	Article name	Used to name the article

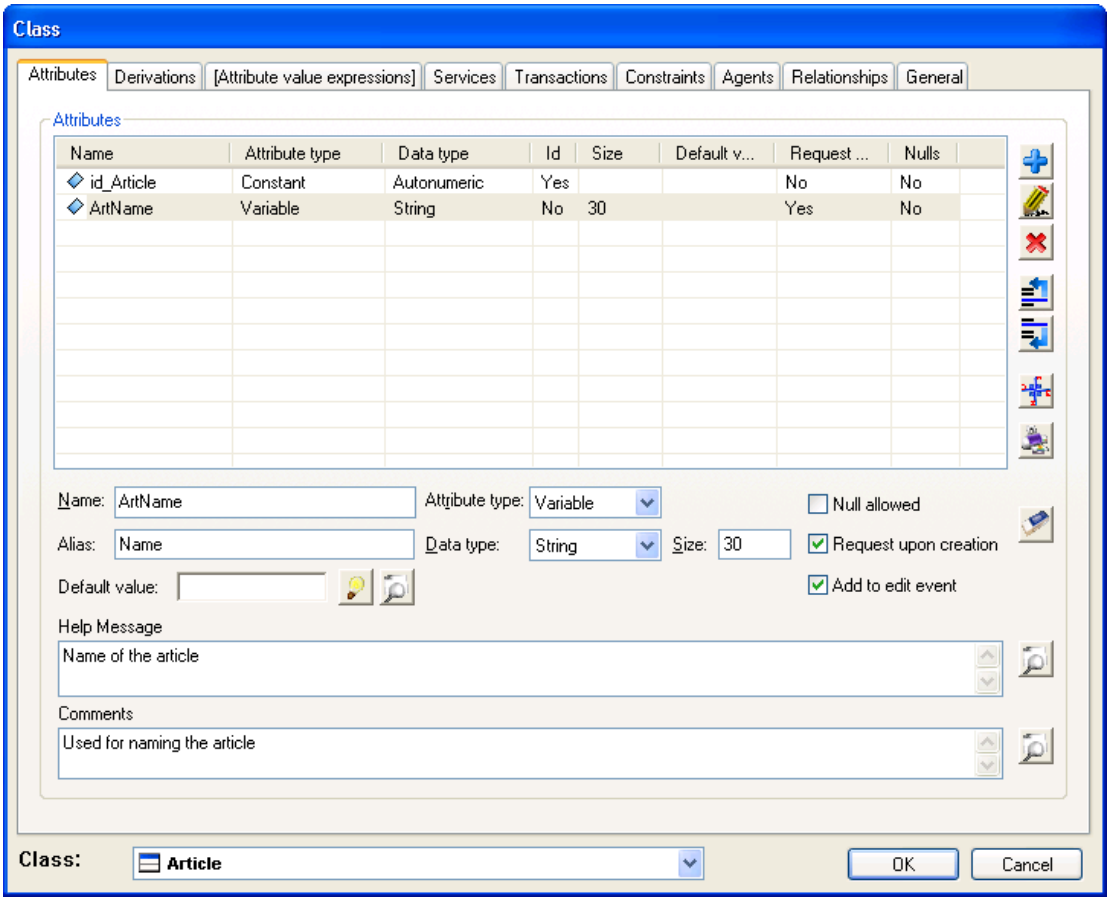

Figure 8 Adding attributes to class Article

Once you are done, click the *Add*  button to have the attribute added to the class.

We will go through this process to add the rest of attributes.

We want to keep the date when an article is created (added to our system). So we need a new 'CreationDate' attribute the Data Type of which will be *Date*. It will be of *Constant* type since its value will never change after an article is created. We assume that articles are added to the catalogue on the same date we create them, so instead of bothering users of our system entering the current date every time they create an article, we will let the system do it for us by (1) unchecking the *Request upon creation* checkbox (so the value for the attribute is not requested), and (2) by defining a *Default Value* for the attribute. The *Default Value* is a formula (we will get into more detail later) of the same data type as the attribute. In our case, we will be using a standard function[1] called *systemDate()* which, no surprise, returns the current system date. We will also uncheck the *Null allowed* checkbox and have 'Date' as alias for the attribute instead of just its name ('CreationDate').

The table below summarizes the properties of the 'CreationDate' attribute:

Table 2 Attribute 'CreationDate' of Article

Name	Alias	Type	Data type	Default value	Null allowed	Request upon creation	Add to edit event	Help message	Comments
Creation Date	Date	Constant	Date	SystemDate()	X	X	X	Creation date	

We want to keep track of the articles in our catalogue that are no longer available but we do not want to just delete them. So we need to somehow differentiate between articles which are "active" and those which are not. Therefore, we add a new *Boolean* attribute to our 'Article' class named 'isActiv'. Articles may or may not be active, that's why we need it to be an attribute of *Variable* Type (so we can change its value). We assume that any article becomes active upon creation in the catalogue, so we set its Default Value to TRUE and uncheck the *Request upon creation* checkbox. And, since we need to change its value eventually, we make sure we check the *Add to edit event* checkbox. Finally we decide we want to display this attribute with the Alias 'Active?' instead of its 'isActive' name.

This is the new attribute we get to add to our 'Article' class:

Table 3 Attribute 'IsActive' of Article

Name	Alias	Type	Data type	Default value	Null allowed	Request upon creation	Add to edit event	Help message	Comments
isActive	Active?	Variable	Bool	TRUE	X	X	V	Is this article available?	Used to discard articles without actually deleting them

[1] Standard Functions are functions already defined in **Integranova Modeler** that provide some common and recurring functionality such as mathematical operations. Developers can use them without implementing any code: the generated code will have them implemented and ready to use.

Finally, let's create the 'ArtPrice' attribute to store the price. We need to save the price of each article and prices may obviously change, so we need to create a new attribute of type *Variable*. As its aim is to store the price of an article, it makes sense to have its data type set to *Real*. This new attribute will not allow null values (every article will have a price in our catalogue) so we will make sure to leave unchecked the *Null allowed* checkbox. We need to set the article price at creation time (we do not want the system pick it for us!) so we will check the *Request upon creation* checkbox. In addition to this, as we decided the price may change, the *Add to edit event* checkbox will be also selected to allow changing this value in the future. Finally we are going to display this attribute with the Alias 'Price'.

This is the new attribute we get to add to our 'Article' class:

Table 4 Attribute 'Prices' of Article

Name	Alias	Type	Data type	Default value	Null allowed	Request upon creation	Add to edit event	Help message	Comments
Prices	Price	Variable	Real	-	X	V	V	Price of the article	

By now, you should have the following set of attributes in your 'Article' class:

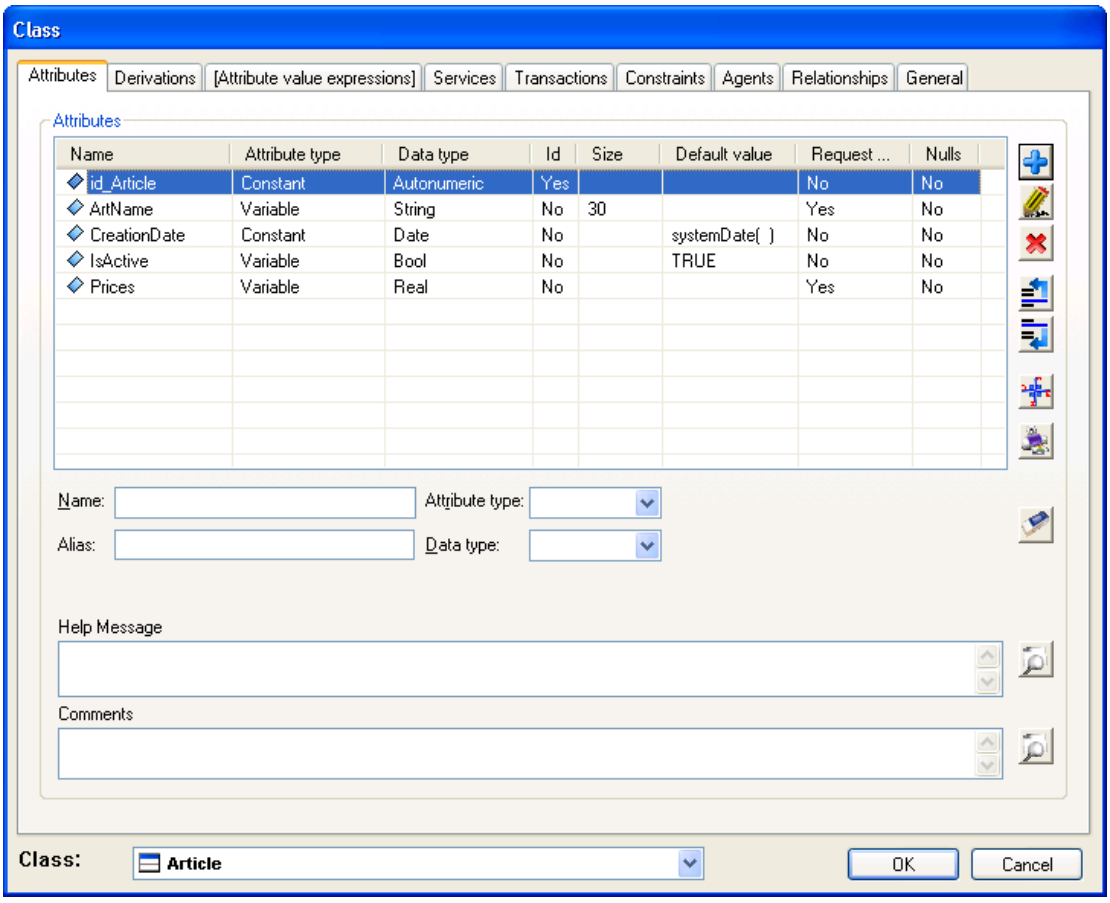

Figure 9 Attributes of class Article

Once all attributes are defined, click *OK* button to save the modifications made to the 'Article' class and return to the work area with the class diagram.

5 Defining basic services and functionality

We have an 'Article' class and its relevant properties: identification mechanism, name, creation date, a flag to tell whether an article is active or not, and the price. We will also need some basic functionality to manipulate our catalogue, namely:

- ✓ Functionality to create articles
- ✓ Functionality to edit article's information
- ✓ Functionality to delete articles

We will define this functionality by means of services.

The services of a class are the basic components associated with the specification of the behavior of a class. The concept of service can be defined as a processing unit, which may be atomic (an event) or molecular (a local transaction or local operation).

All services have a Name and, optionally, an Alias, a Help Message and some Comments. In addition to this, service may have a number of arguments.

We can have three kinds of services in our classes: Events, Transactions and Operations. Events are the simplest kind and we will discuss a bit about them here. Further details about events and transactions/operations will be presented later in the series.

If you double-click the 'Article' class and go to the *Services* tab, you will see there are three services (specifically events) already in place.

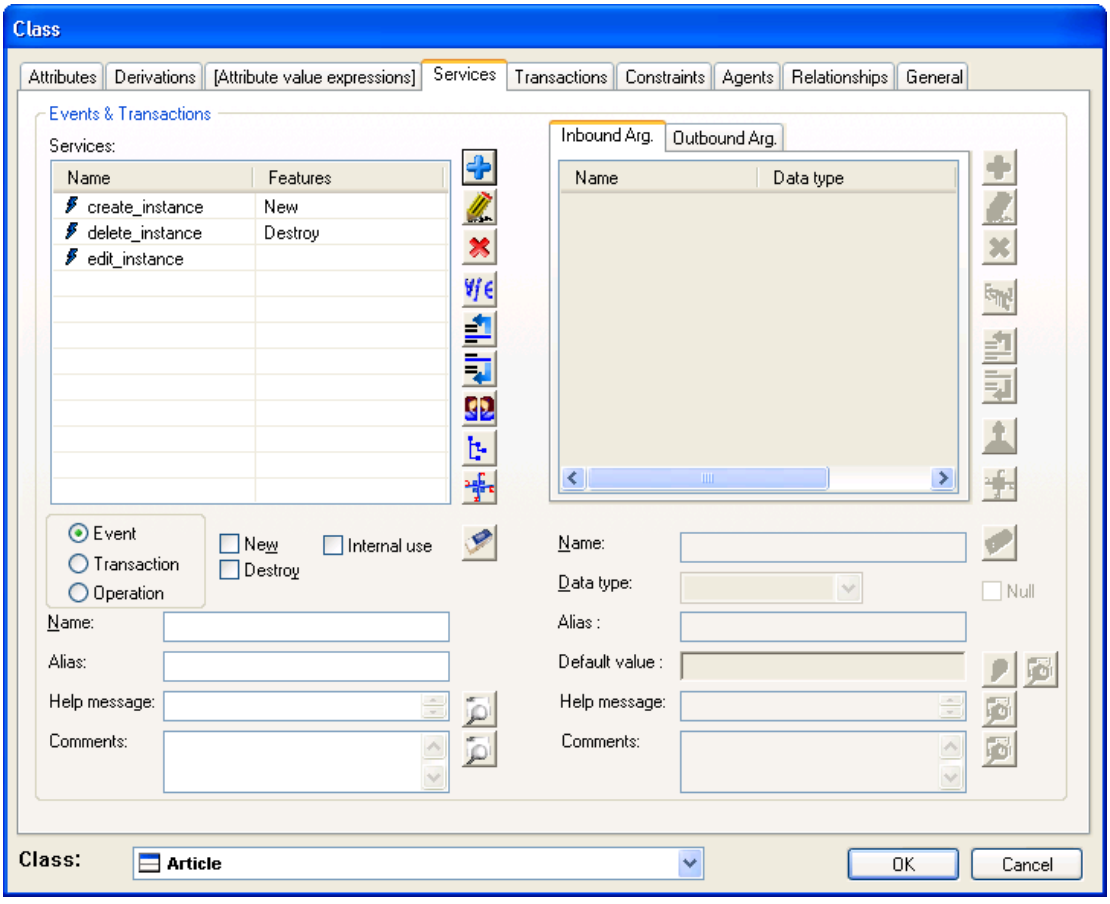

Figure 10 Services of 'Article' class

These three events were automatically created by the tool when we created our 'Article' class with the *Extended Creation* checkbox checked. The names of each event will give you a hint of what they are for: 'create_instance' is used to create articles, delete_instance is used to delete articles from our catalogue and 'edit_instance' is used to modify the editable properties of an article (that is, the attributes of *Variable* type).

For the 'create_instance' and 'delete_instance' events, the functionality is automatically managed by the **Integranova Modeler**.

Double click the 'create_instance' event and look at the list of arguments to the right of the screen: there is an argument for every attributes whose value will be requested upon creation of a new article (those attributes we defined with the *Request upon creation* checked). In this case, attributes 'ArtName' and 'ArtPrice'.

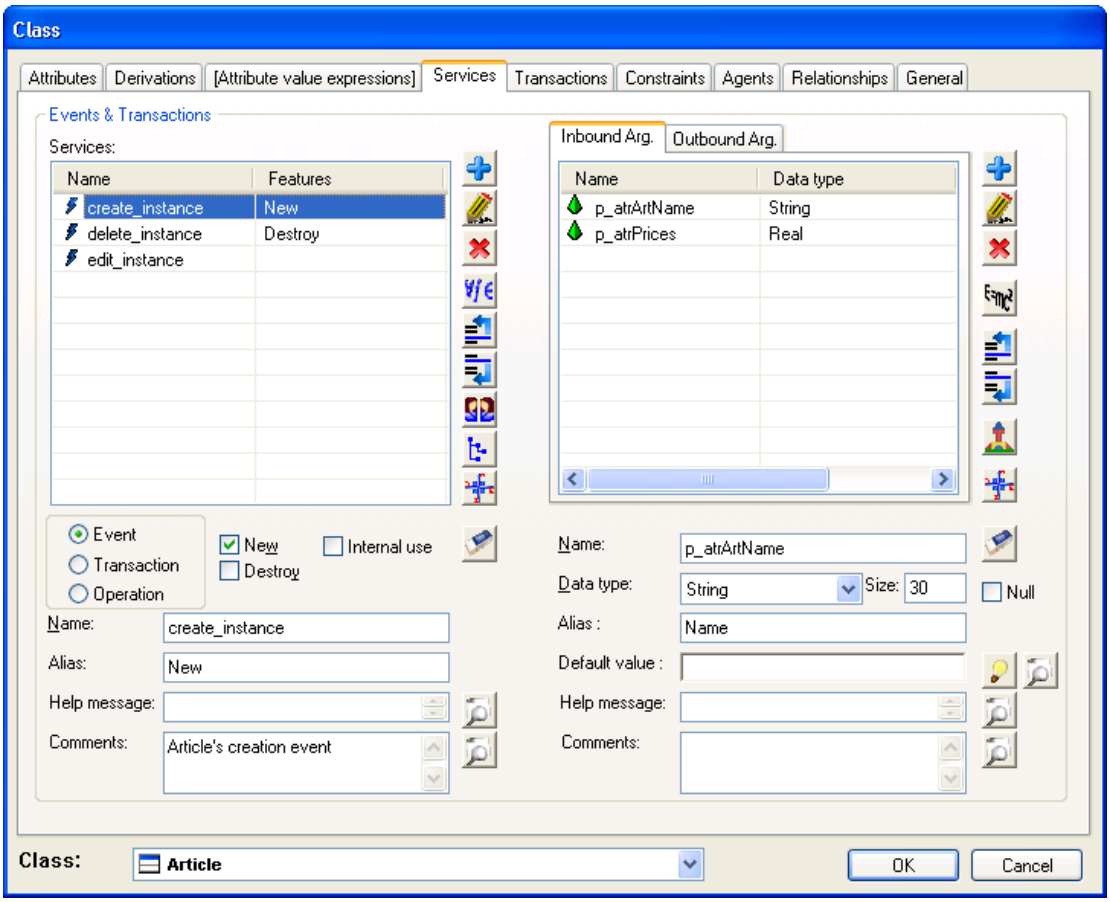

Figure 11 *Create_instance* event

What the 'create_instance' event does is to create a new article and assign the following values to its attributes:

- 'id_Article': a numeric value (an integer) which is unique among all the articles in the catalogue.
- 'ArtName': whatever name is provided in the 'p_atrArtName' argument.
- 'CreationDate': the current system date (value provided by standard function *systemDate*)
- 'IsActive': a Boolean value of TRUE.
- 'ArtPrice': whatever prices are provided in the 'p_atrArtPrice' argument.

Double click the 'delete_instance' event and you should see it only has an inbound argument named 'p_thisArticle', the type of which is 'Article'. That argument represents the actual article that is to be deleted.

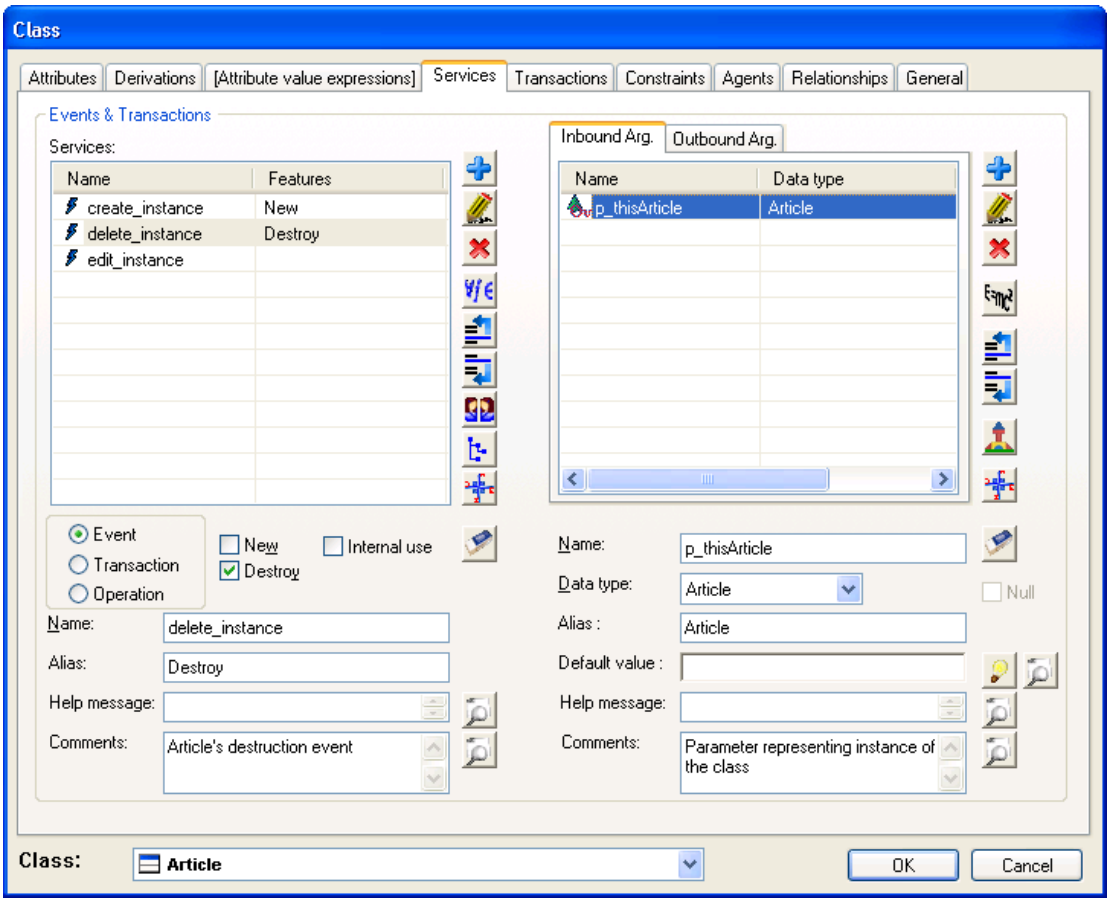

Figure 12 Delete_instance event

What the 'delete_instance' event does, in this case, is just to delete the article from the catalogue.

Finally if you double click the 'edit_instance' event, you will see a 'p_thisArticle' inbound argument representing the article that is to be edited and also some inbound arguments representing those *Variable* attributes we created with the *Add to edit event* checkbox checked. This was the case of attributes 'ArtName', 'IsActive' and 'ArtPrice'.

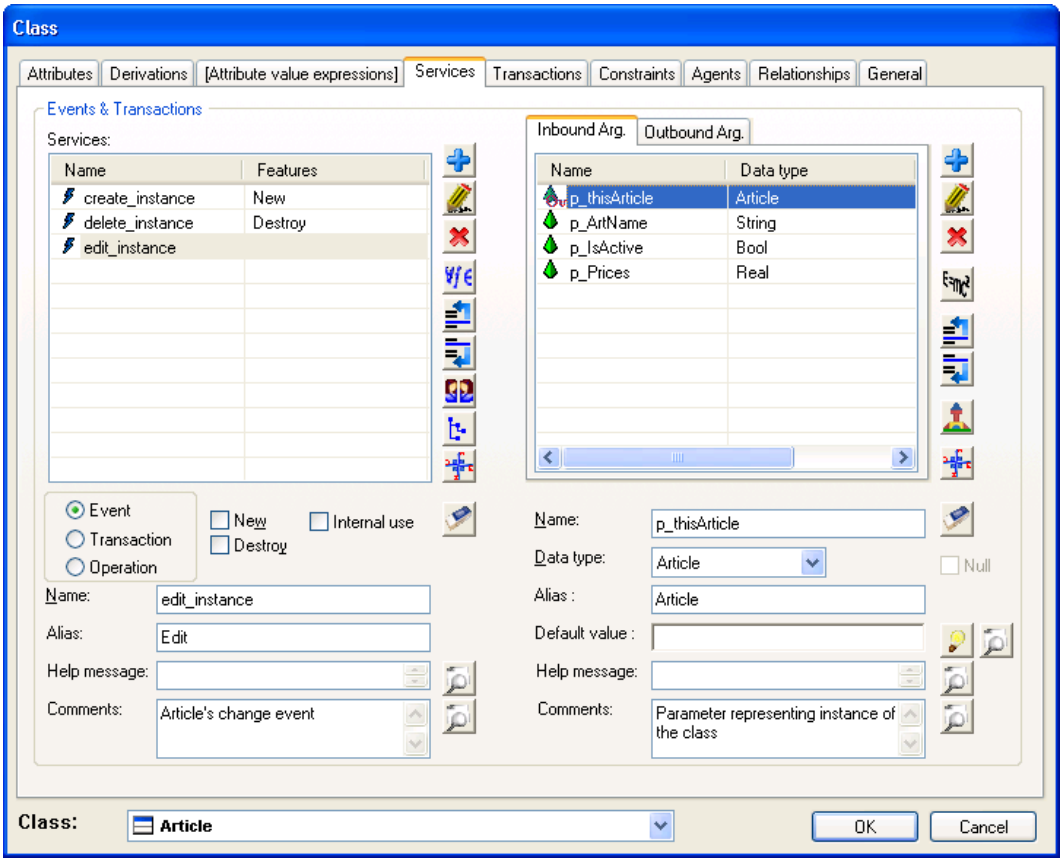

Figure 13 'Edit_instance' event

The functionality of 'create_instance' and 'delete_instance' is somehow implicit, but in the case of 'edit_instance', the functionality is explicitly defined by means of *Valuations*.

To take a look at them, click the *Go to…* button  having selected 'edit_instance' and select *Valuation* in the dialog that prompts.

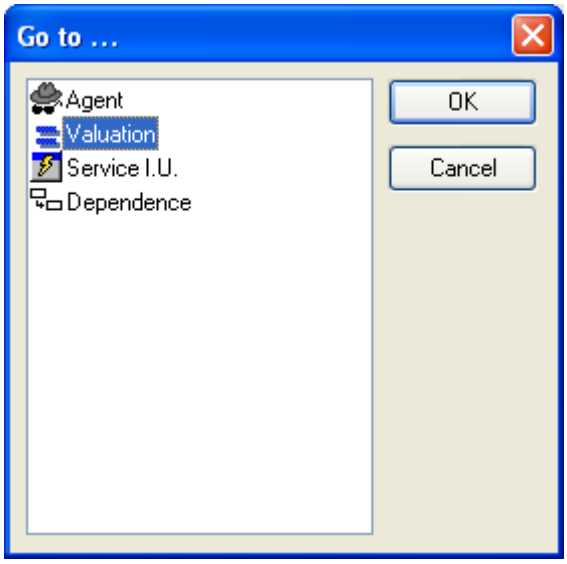

Figure 14 *Go to …* dialog

Click *OK* button so the *Functional Model* dialog shows with the valuations for the 'edit_instance' event.

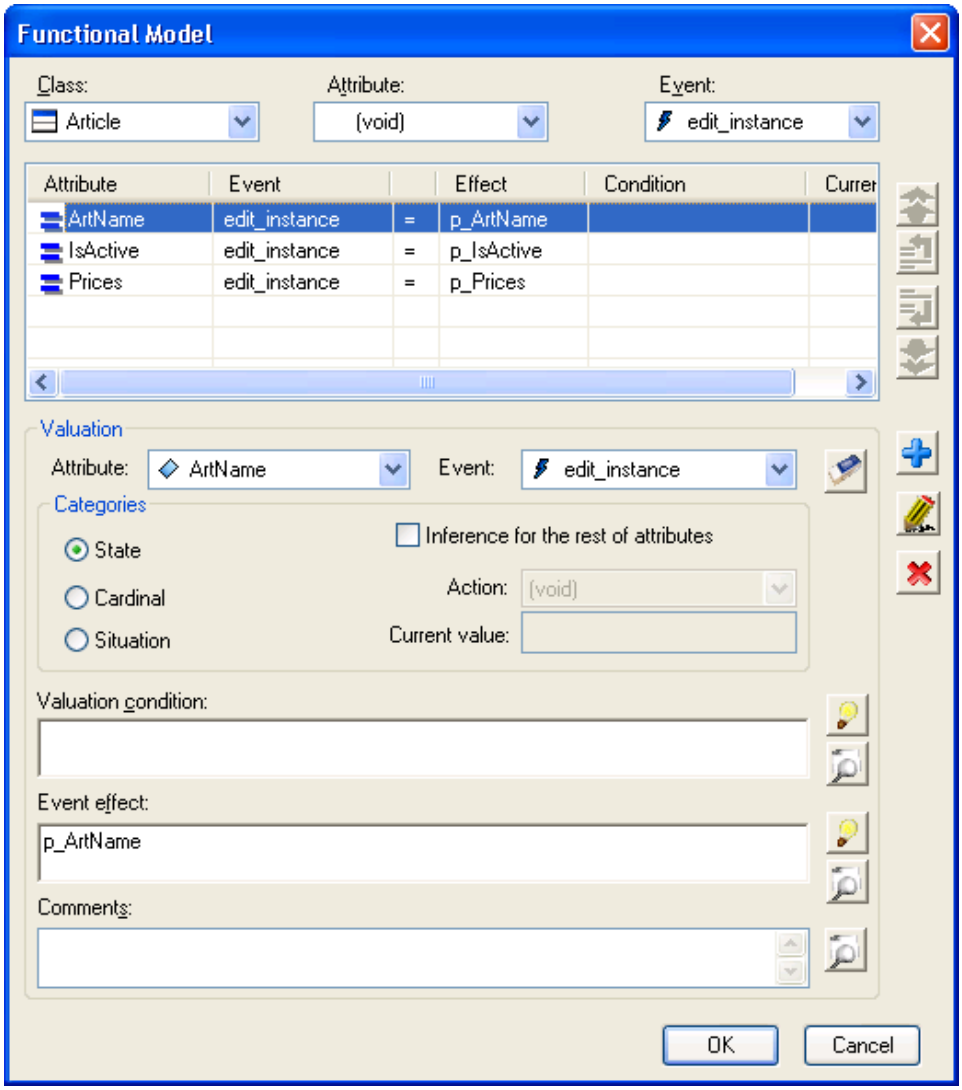

Figure 15 Valuation formulas for the 'edit_instance' event

A valuation is the effect of an event of one class on a determined variable attribute of the same class. It determines the value taken by a variable attribute when an event occurs. As you can see in the case of the 'edit_instance' event of our 'Article' class, upon its execution, the 'ArtName' attribute will change to the value of the 'p_ArtName' argument, 'IsActive' will change to the value of 'p_IsActive', and 'ArtPrice' will change to the value of 'p_ArtPrice'.

6 Defining access to the catalogue and permissions

As this is a very simple system, we will not care about who can access the system for the moment. Therefore will grant access to the catalogue to everyone and everyone will be able to browse the catalogue and create, delete and modify articles.

Let's introduce the notion of *Agent*. An agent is just another class in our model (so it can have attributes and services) for which we define permissions on the kind of this it can do to other classes. An *Agent Interface* (or *Interface*, for short) relates two classes – one playing the role of agent, the other playing the role of server – and defines what the agent class can do to the server class.

What we will do next is to create a new 'AnonymousUser' class that will represent anyone accessing the catalogue and grant it permission to do everything.

So follow the previous steps to create a class, and create a new 'AnonymousUser' class with these properties:

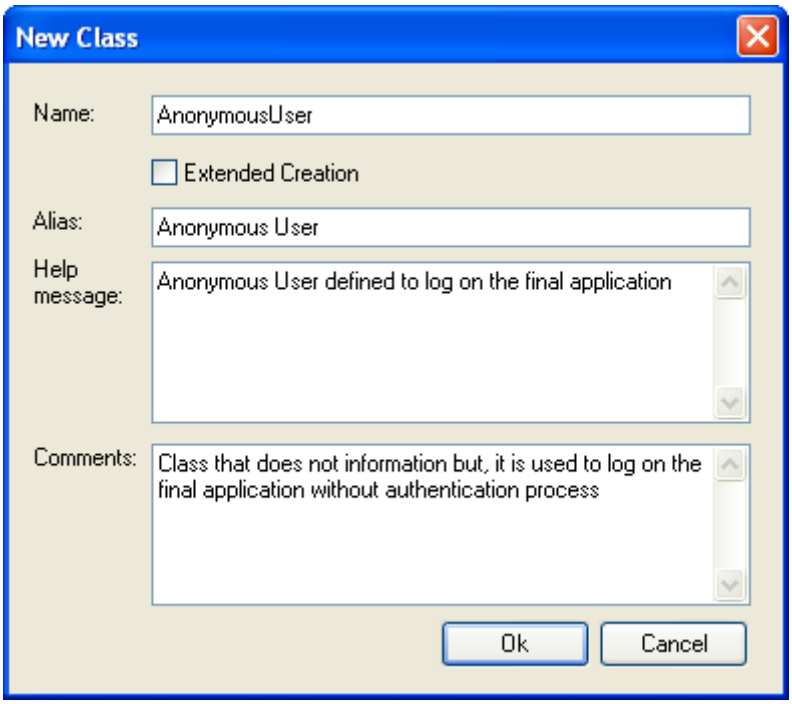

Figure 16 'AnonymousUser' information

Note: Notice that for this class the *Extended Creation* check box has not been checked. This is because we do not need our 'AnonymousUser' agent class to have attributes or services.

Once you are done, click the *OK* button and the 'AnonymousUser' class will be placed in the work area.

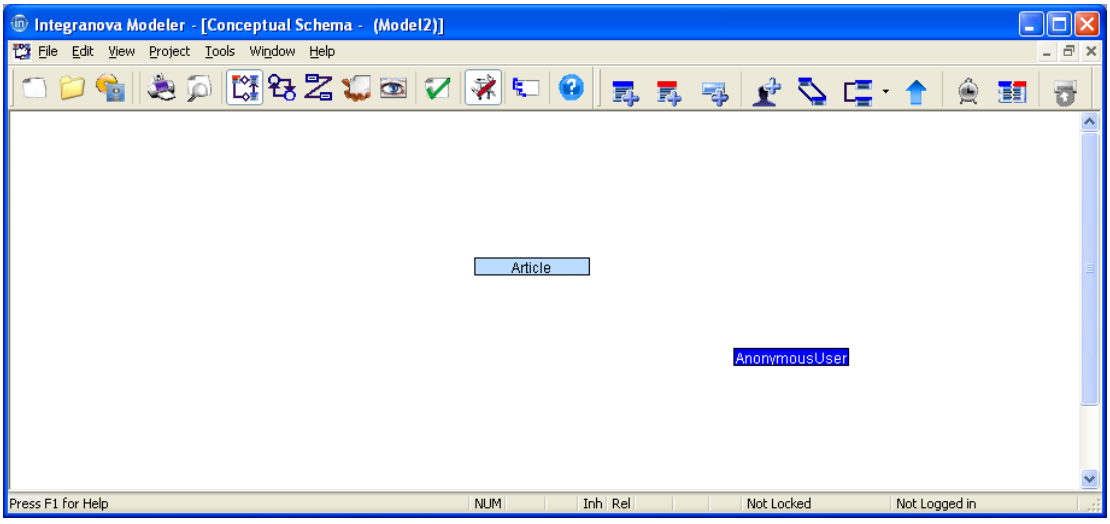

Figure 17 Work area with 'Article' and 'AnonymousUser' classes

Double click on the 'AnonymousUser' class, go to the *General* tab and check the *Anonymous Agent* checkbox as shown in the image below.

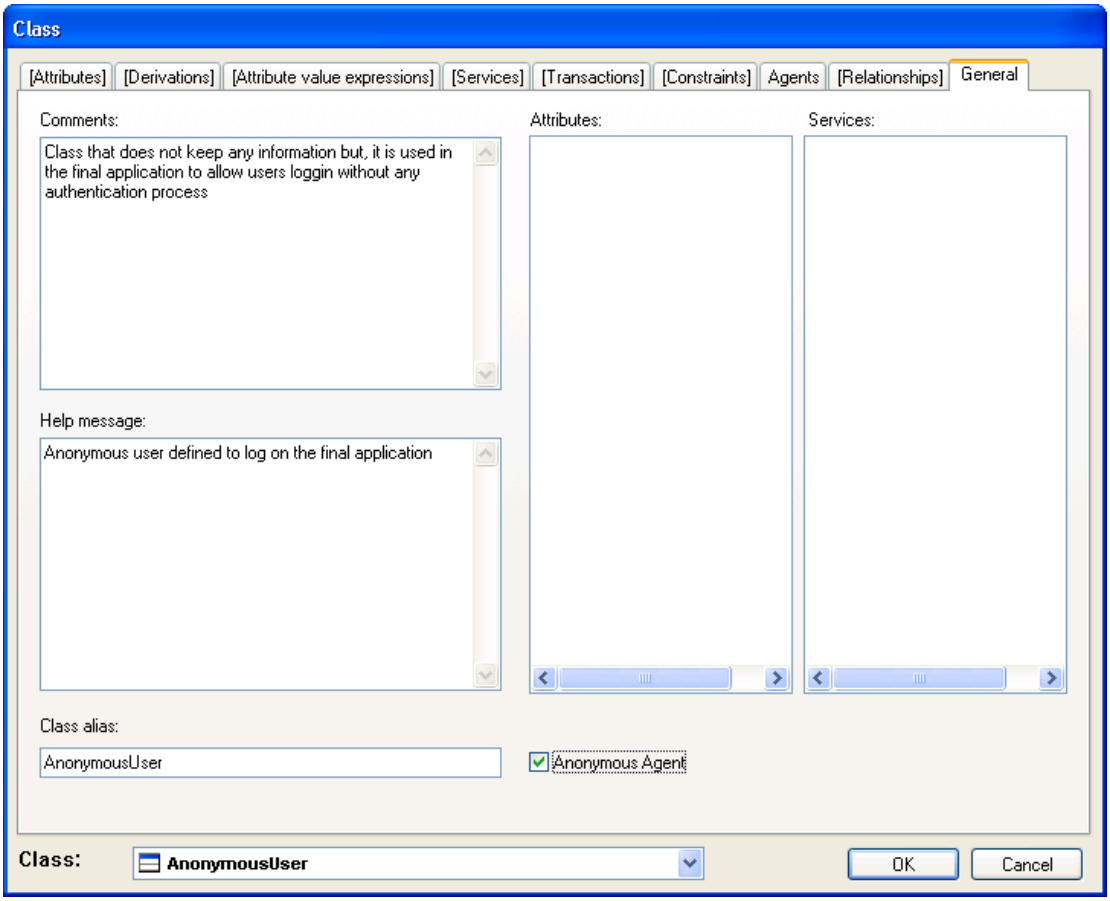

Figure 18 Defining a class as anonymous agent

Click the *OK* button. The 'AnonymousUser' class is an anonymous agent class. We just need now to assign permissions to it.

Note: After selecting the *Anonymous Agent* check, all tab names in the dialog, except for *Agents* and *General*, are shown between brackets. That means these tabs are not accessible. The reason for this is that an anonymous agent class has no data or functionality. It is created just to define a set of agent permissions over other classes of the system.

The next step will be to define the permissions of the anonymous agent on the rest of classes of the system. Since we want everyone (which is what our 'AnonymousUser' class represents) be able to do everything, we have an easy and convenient way to specify this in the tool: define the 'AnonymousUser' class as *Agent of the whole system*.

Right-click the 'AnonymousUser' class and select *Agent of the whole system* option from the context menu:

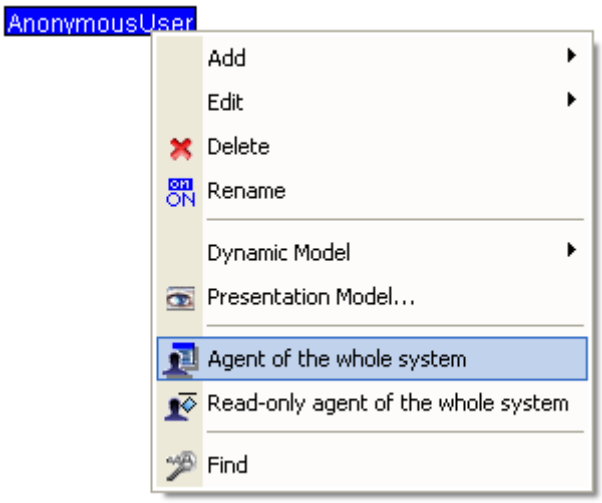

Figure 19 Agent of the whole system

Note: *Agent of the whole system* is a utility which can set up a class as the agent of all classes in the system. This property is not automatically updated, so the selected class will be the agent for all elements in the system at the time this option is selected. In other words, this class will not be the agent for any subsequent elements added after executing the utility. To update and include such elements for the class to be agent of the whole system again, the utility must be executed again, specifically adding the recent elements.

Press *YES* in the confirmation dialog:

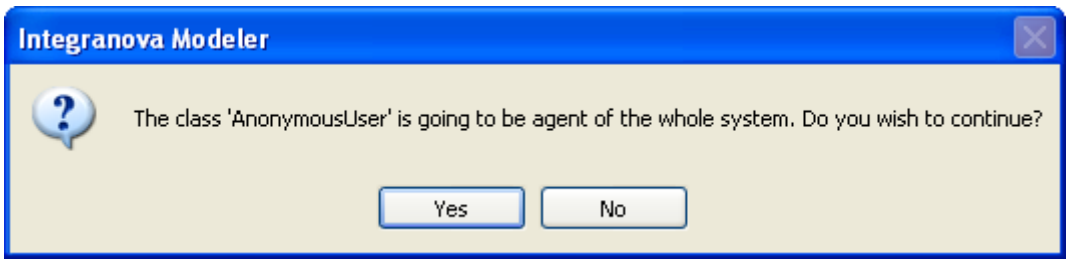

Figure 20 Confirmation dialog

7 Default user interface

So far we have defined the static (structure) and dynamic (behavior) part of our system: our 'Article' class with its attributes and our 'AnonymousUser' class with its permissions to do everything. You might be wondering what happens to the interaction aspects (i.e: the user interface) of the system we are developing.

The analyst can specify a user interface through the concept of *View* and subsequently through the *Presentation Model*.

Both concepts exceed the contents of this tutorial document, so we will just present a very concise introduction to these concepts.

VIEW

A *View* defines, first of all, the collection of agents that can connect to the system and their visibility (what they can see and do). Since a view is a set of agent interfaces, the visibility of a view must be restricted at run-time to that of the connected agent, i.e. to that determined by the interfaces of the view for the agent class, but not the rest of the interfaces of other agent classes.

PRESENTATION MODEL

The *Presentation Model* defines the abstract user interface that the system offers to the connected user. Since the user is connected as an instance of an agent class, this user interface is restricted to the effective visibility of this user as an instance of the agent class.

Nevertheless, if no user interface is defined, **Integranova Modeler** will automatically create one by default. In this default user interface, there will be all the necessary interaction mechanisms to visualize data and execute services to manipulate said data.

In this tutorial we will rely on **Integranova Modeler** to have that default user interface defined for us. We will get into the details of defining our user interface later in the series.

8 Validating the model

Once we are done with the model, we need to validate it before we can have it automatically transformed to the source code of our application. We validate the model by clicking the *Validate Model* button  in the toolbar (or, alternatively, by pressing Ctrl+F5).

Figure 21 Validate Model option

The *Model validation* dialog is shown reporting information about the results of the validation. There should no be any errors or warnings.

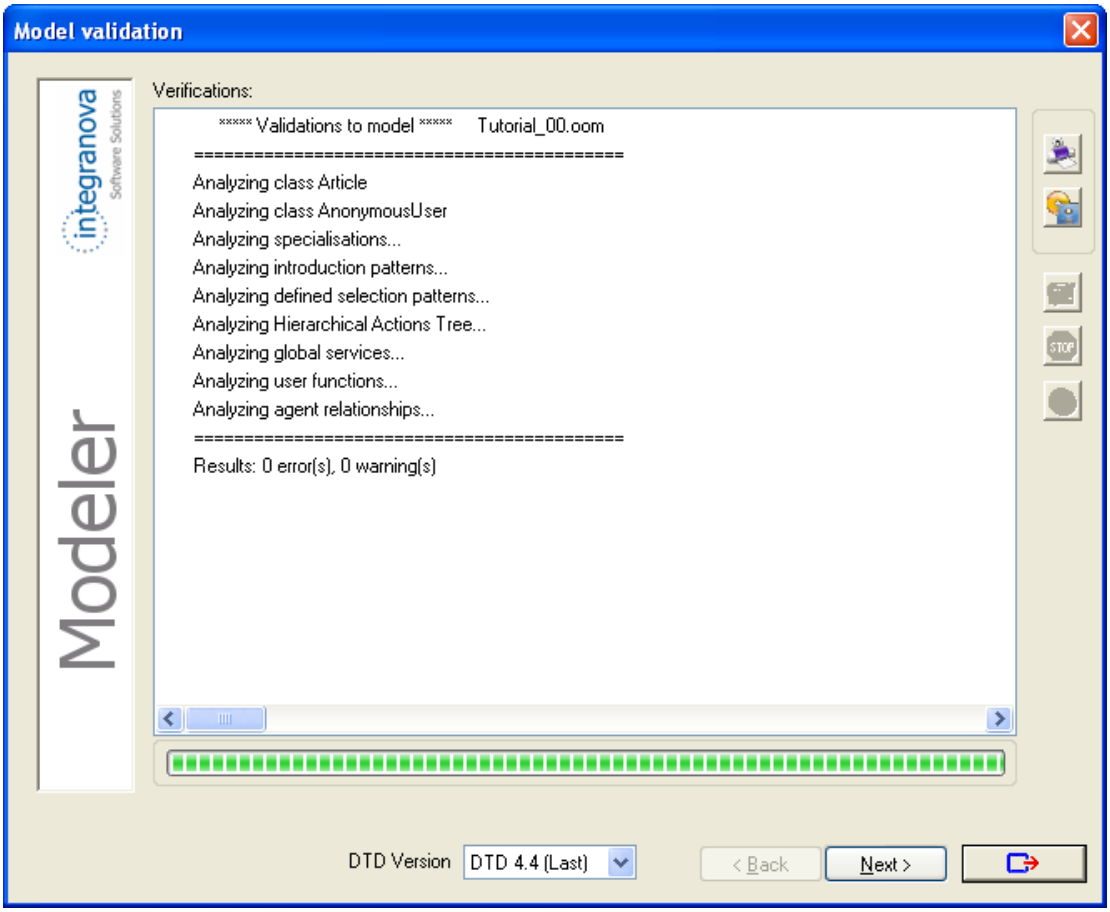

Figure 22 Model Validation dialog with validation information

Click the *Next* button and you will be prompted for a path and file name to save your model to an XML file. Select the path and file name and click *Save* button.

The *Model validation* dialog is shown again but reporting the results of the generation of the XML file. Click *Next* to transition to the **Integranova STAR Client** and specify your preferences for transforming your model to the source code of your application.

9 Generating the application

Once in the **Integranova STAR Client** application, you only need to follow the same steps we learnt in the first of the series (Tutorial 0), as to send your Transformation request through the **Integranova STAR Client** application to the Transformation Engines.