integranova

Software Solutions

DEFINING PRECONDITIONS AND CONSTRAINS

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 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 03.

In the Tutorial 03, we learned how to manage dynamic relationships in our model and how to restrict our model using preconditions, integrity constraints and horizontal visibility formulas. In this new tutorial, we will learn how to increase the complexity of these formulas using *association operators*.

By the end of this tutorial, you will developed a system in which you will be able to create only one level of subcategories, the customer only can add one line of each article in a purchase order and pay a purchase only when the payment type is defined.

2 Opening the initial model

In this tutorial, we will start from the model we created in the previous tutorial in the series: 'Tutorial_03_Final.oom'.

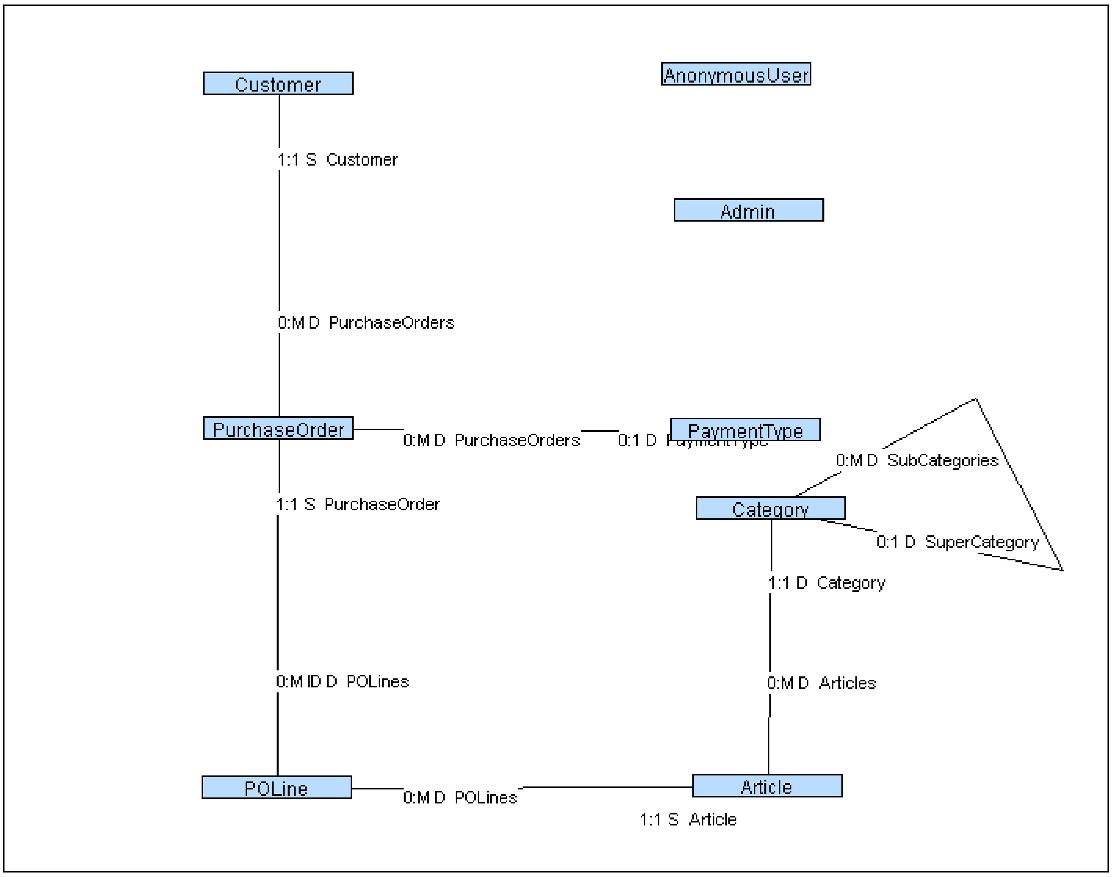


Figure 1 Tutorial_03_Final.oom

3 Defining complex preconditions

We would add certain restrictions for our model. To do that, we use *Preconditions*. As we saw in the previous series, *Preconditions* are conditions that must be satisfied before executing a service.

Our first precondition in this tutorial must cover the following requirement: The customer can only pay a purchase order if it has assigned a payment type.

The way in which we can check if a payment type has been already related with the purchase order is using the *EXIST* operator. The precondition formula must be:

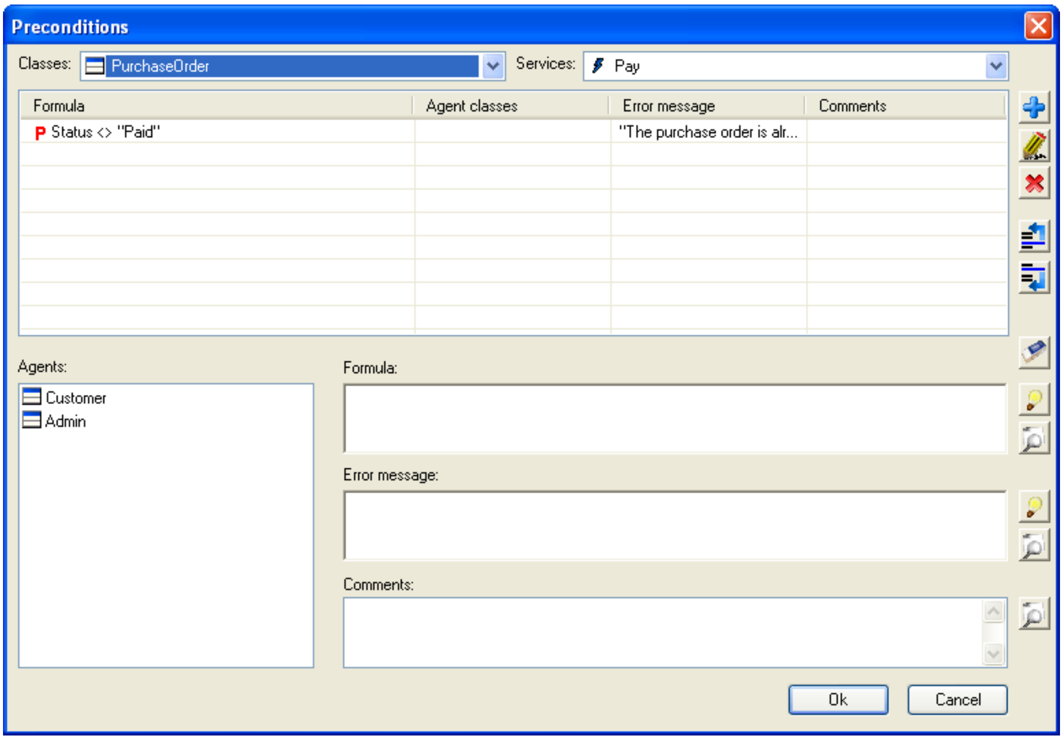
```
EXIST (PaymentType) = TRUE
```

The *EXIST* is an association operator that returns TRUE when there is at least one instance reached trough a role path, in this case, our role path is 'PaymentType (name of the role through 'PurchaseOrder' access to 'PaymentType') and FALSE when no instances can be reached.

Note: Notice the *EXIST* operator is useful when the minimum cardinality of the related role is 0. With 1 as minimum cardinality, *EXIST* operator returns always **TRUE**. Integranova do not allow creating data inconsistent with the model.

Let's create our first precondition with association operators in the model:

1- Select Pay event and press Precondition button.



Classes: PurchaseOrder
Services: Pay

Formula	Agent classes	Error message	Comments
P Status <> "Paid"		"The purchase order is alr...	

Agents:
Customer
Admin

Formula:

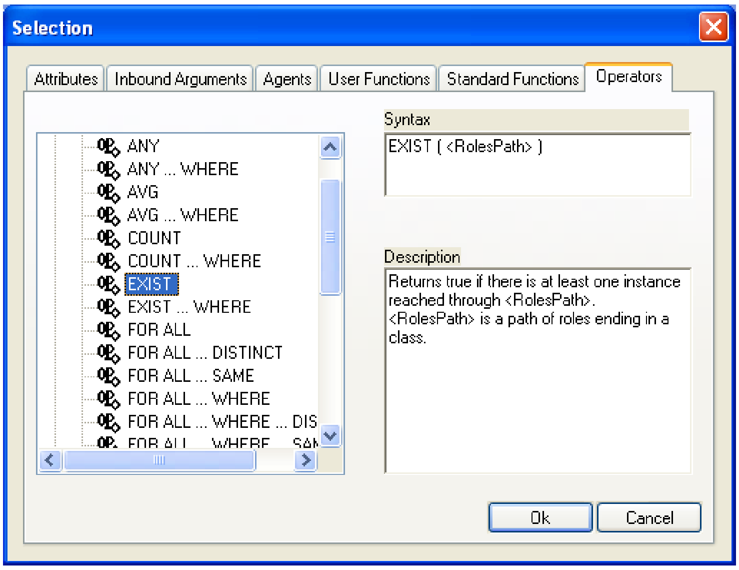
Error message:

Comments:

Figure 2 Preconditions

2- Press the *Help* button to access at *Help Navigation Window*

3- Go to *Operators* tab → *Collection Operators*, and choose the *EXIST* Operator



Syntax: EXIST (<RolesPath>)

Description: Returns true if there is at least one instance reached through <RolesPath>. <RolesPath> is a path of roles ending in a class.

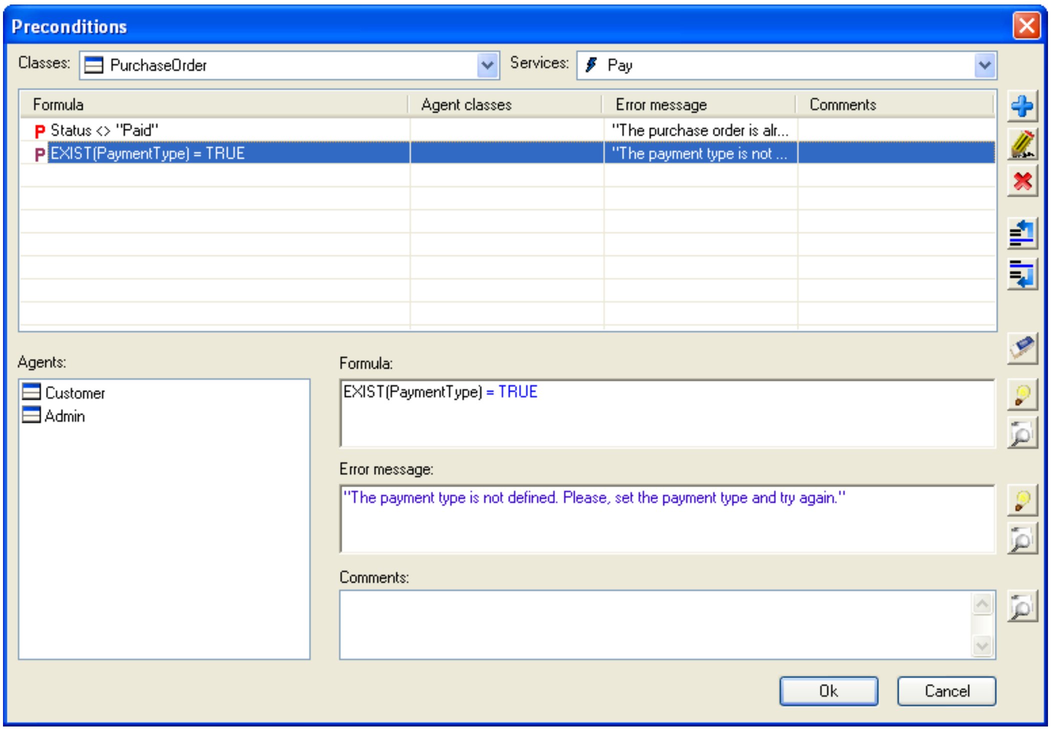
Figure 3 *EXIST* operator

4- Click on OK button

5- Integranova Modeler has added in the formula:

```
EXIST ( <RolesPath> )
```

6- Fill the data of Formula and Error Message as it is shown bellow:



Classes: PurchaseOrder

Services: Pay

Formula	Agent classes	Error message	Comments
P Status <> "Paid"		"The purchase order is alr...	
P EXIST(PaymentType) = TRUE		"The payment type is not ...	

Agents:
Customer
Admin

Formula: EXIST(PaymentType) = TRUE

Error message: "The payment type is not defined. Please, set the payment type and try again."

Comments:

Figure 4 Preconditions of 'Pay' event

7- Press Add ✚ button

8- Click on OK button

4 Defining complex constrains

As said in the Tutorial_03, the values of the attributes of an object can be restricted by a series of conditions that need to be satisfied at *any time* (= all of the states of the object). These conditions are called *Constraints*.

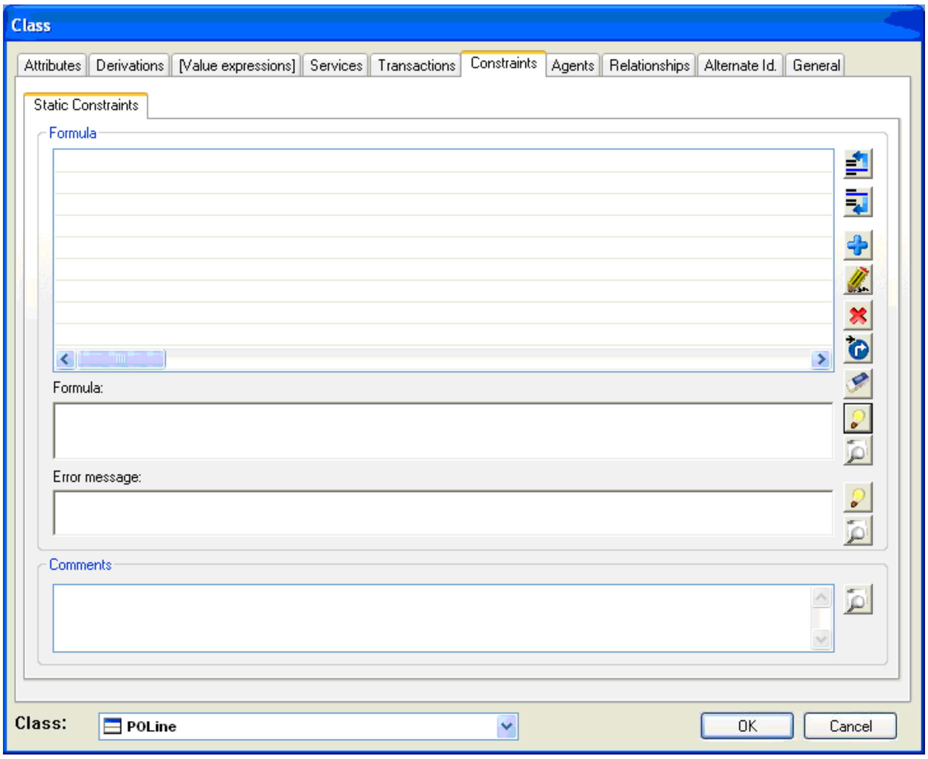
In our model we need to control that "the customer only can add one line of each article in a purchase order" that means, when a customer wants to create a purchase order line, it must be checked that all the articles of the lines of the purchase order are different. The formula, placed in 'POLine' class, will be:

```
FOR ALL PurchaseOrder.POLines DISTINCT
(PurchaseOrder.POLines.Article.ArtCode) = TRUE
```

To do this, we need the *FOR ALL … DISTINCT* association operator. This association operator returns TRUE if all instances reached through a role path have different values for a given attribute. In our tutorial, the role path is "PurchaseOrder.POLines" (being in a line, this roles path represents all the lines related to my own purchase order, including myself), and the attribute whose value must be different is "ArtCode" (being acceded from the article related to the lines of the purchase order).

We are going to define and create now our constraint with association operators in our model:

1- Go to the 'POLine' class and click on *Constraints* Tab

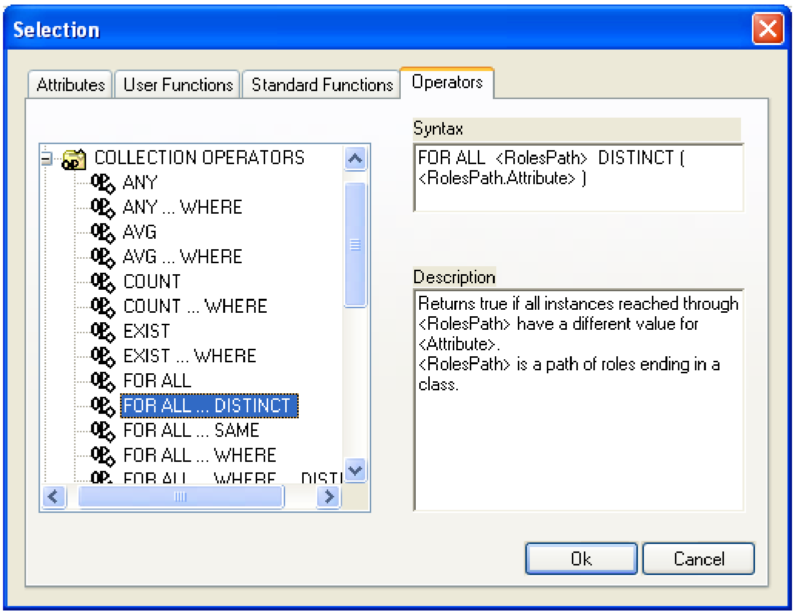


Class: POLine

Figure 5 Constraints Tab

2- In the Formula text box, click on *Help* button to access at *Help Navigation Window*

3- Click on *Operators* tab → *Collection Operators*, and choose *FOR ALL … DISTINCT*



Syntax

FOR ALL <RolesPath> DISTINCT (<RolesPath.Attribute>)

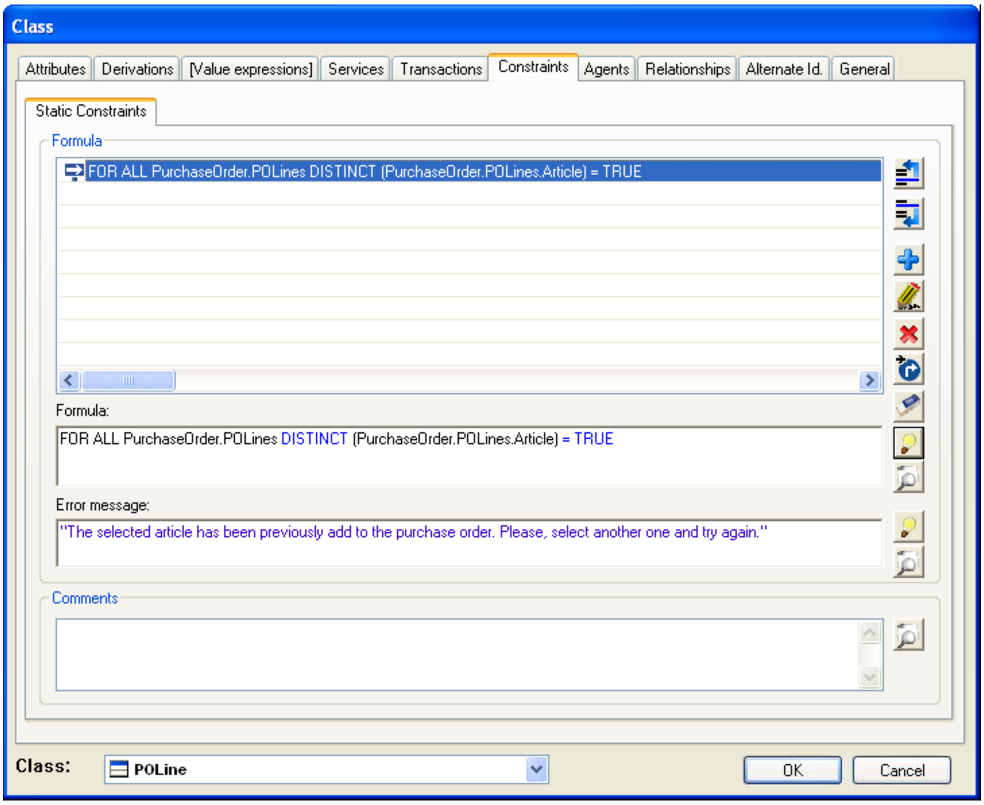
Description

Returns true if all instances reached through <RolesPath> have a different value for <Attribute>.
<RolesPath> is a path of roles ending in a class.

Figure 6 Selection of a collection operator

4- Click on *OK* button

5- Then fill the formula as following:



Formula

FOR ALL PurchaseOrder.POLines DISTINCT (PurchaseOrder.POLines.Article) = TRUE

Formula:

FOR ALL PurchaseOrder.POLines DISTINCT (PurchaseOrder.POLines.Article) = TRUE

Error message:

"The selected article has been previously add to the purchase order. Please, select another one and try again."

Class: POLine

Figure 7 Integrity Constraint in 'POLine' class

6- Press *Add* ✚ button.

7- Click on *OK* button.

Now, following with the tutorial description, we want to limit the levels of subcategories. To explain this requirement let's see an example: if we have a *Films* category, subcategories of this one could be: *Action*, *Fantasy*, *Drama* etc. but these subcategories of *Films*, could not have other nested categories.

We can express that, saying: "If a category has subcategories, then it is a main category, and it cannot have related a super category, and if has a super category, then it is a subcategory and it cannot have subcategories itself". In other words: "A category cannot have *subcategories* and a *super category* at the same time".

To represent this fact, we need to add a new constraint in 'Category' class.

The formula of this constraint will be:

```
EXIST (SubCategories) = FALSE OR EXIST (SuperCategory) = FALSE
```

To do this, we use twice the operator *EXIST* and the logical operator *OR* that returns TRUE if at least one of the conditions is TRUE.

Now we are going to add the constraint to our model.

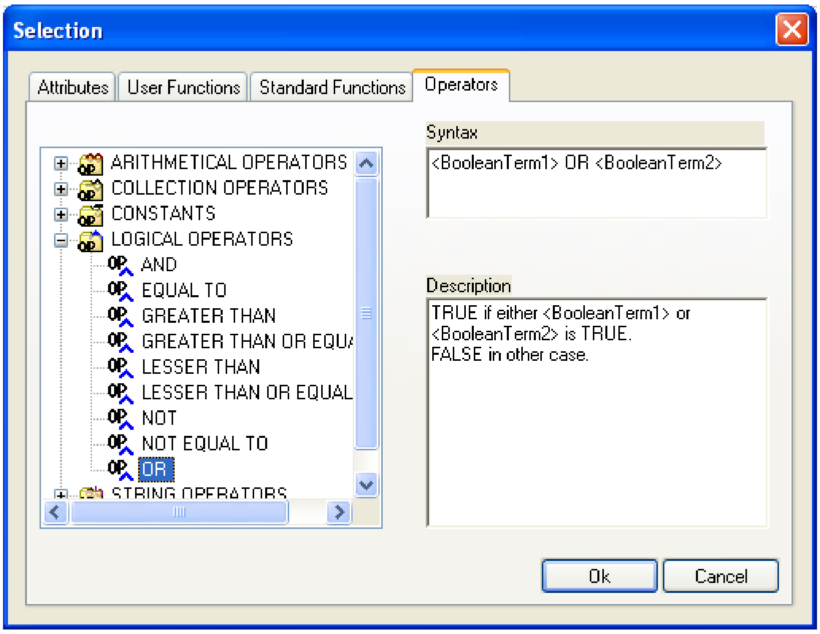
1- As in the previous steps, choose *Category* class and click on *Constraints* tab.

2- Click on *Help* button to access at *Help Navigation Window.*

3- Go to *Operators* tab → *Collection Operators*, and choose the *EXIST* operator.

4- Press *OK* button.

5- Now we are going to add the logical operator *OR*, repeat the previous steps and go to *Operators* tab → *Logical Operators* choose the *OR* operator, Press *Ok* button.



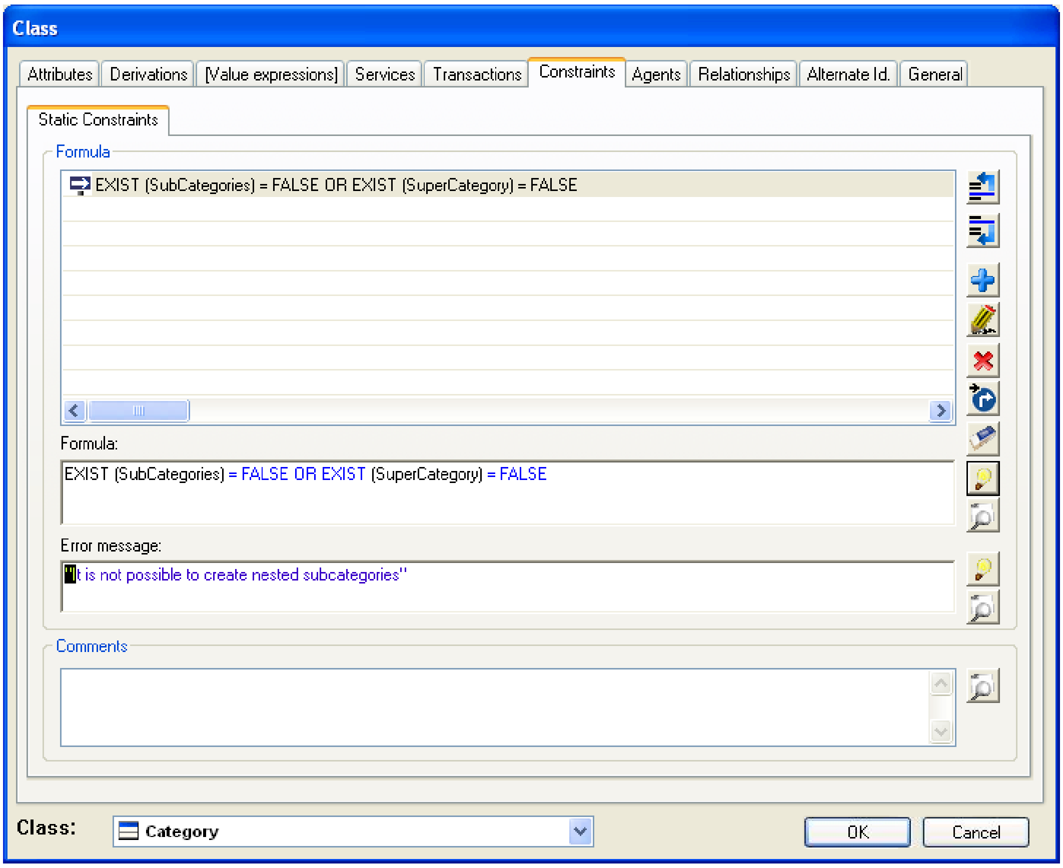
Syntax
<BooleanTerm1> OR <BooleanTerm2>

Description
TRUE if either <BooleanTerm1> or <BooleanTerm2> is TRUE.
FALSE in other case.

Figure 8 *OR* operator

6- Now repeat these steps to add a second *EXIST* clause.

7- Then fill the formula and the Error Message as in the next figure.



EXIST (SubCategories) = FALSE OR EXIST (SuperCategory) = FALSE

Formula: EXIST (SubCategories) = FALSE OR EXIST (SuperCategory) = FALSE

Error message: "It is not possible to create nested subcategories"

Class: Category

Figure 9 Integrity Constraint in 'Category' class

8- Press *Add* ✚ button.

9- Click on *OK* button.

Finally, you will have defined all new restrictions in your model

5 Default user interface

Integranova Modeler offers the possibility of creating a customized user interface using the features associated to the *Presentation Model*. All these features will be presented from Tutorial 9 in the series tutorials to the last one (Tutorial 15).

Meanwhile, **Integranova Modeler** provides a default user interface where all the functionality defined in the model can be accessed. It will be possible to access to the application using any of the agents defined in the model.

The main menu will have as many menu entries as classes defined in the model (the name will be the alias of the class). Each menu entry will have the following subentries: one scenario to show **one instance** of the class, another scenario to **list all the instances of the class** and as many scenarios as **services** we have defined in the class.

When an agent is connected to the application, he only will have access to the services that he can execute and the roles and attributes over which he has visibility. The menu entries which the agent has no permission (no execution, no visibility, no navigation) will be hidden.