USING CONNECTED AGENT

SAMPLE: SHOPPING CART

Last Update: Aug 2010

Table of Contents

1 Goal 2
2 Agent logged to the application 2
3 Customer: I create *my* purchase order 3
4 Admin: *I* verify purchase orders 5
5 Customer: If I am disabled *I cannot* create purchase orders 7
6 Default user interface 9

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

In the Tutorial 05 we learned how to create transactions and operations, and how to define their formulas. In this new tutorial we will learn how to modify the application behavior according to the agent who is logged in the application.

By the end of this tutorial, you will have developed a system able to directly use the connected agent in some process, for example avoiding the creation of new purchase orders by disabled customers and using the agent data in services without asking them to the logged customer or administrator.

In this tutorial we will start from the model we created in the previous tutorial in the series: Tutorial_05_Final.oom.

Agents' visibility: In this tutorial, some attributes and services will be added to the model. Please remember to assign the agents visibility over these elements.

2 Agent logged to the application

In this tutorial, the modeling element you will need is the "***AGENT_****ClassName*". It allows accessing to the connected agent data. Consequently *ClassName* has to be any agent class defined in the model. In this tutorial, we are going to use:

- ✓ ***AGENT_****Customer*
- ✓ ***AGENT_****Admin*

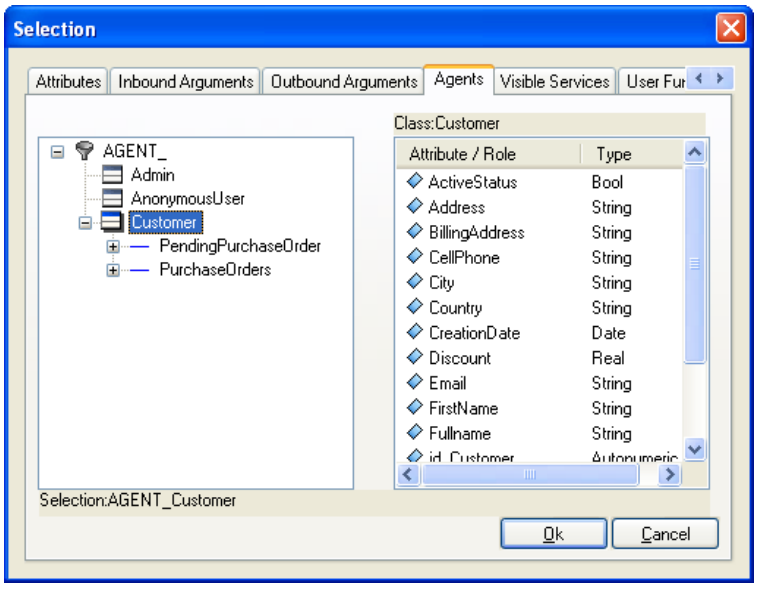

Figure 1 *AGENT_**ClassName* element**

3 Customer: I create *my* purchase order

By definition, when a purchase order is created, a customer must be assigned. Besides, this customer is who created the purchase order and so, the current user logged as customer.

This behavior should be transparent to the users, since we can directly obtain the current customer logged. So, it will not be necessary to ask for a customer as inbound argument.

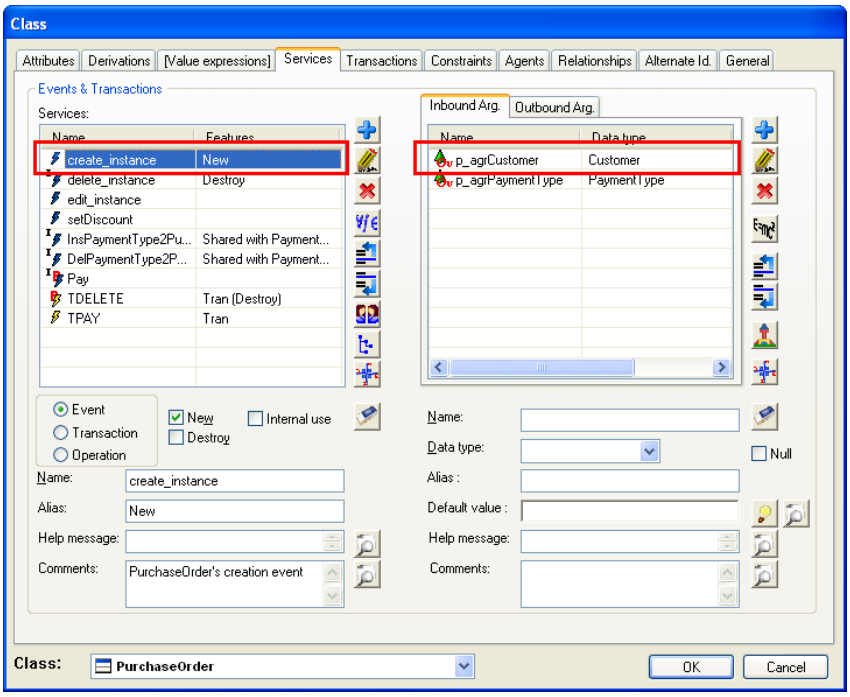

Figure 2 Customer in default purchase order creation service

Because the 'create_instance' has a Customer inbound argument and it cannot be removed, the solution is to create a new transaction to create purchase order named 'TCREATE'. This new transaction does not require arguments because the only needed value is the logged customer:

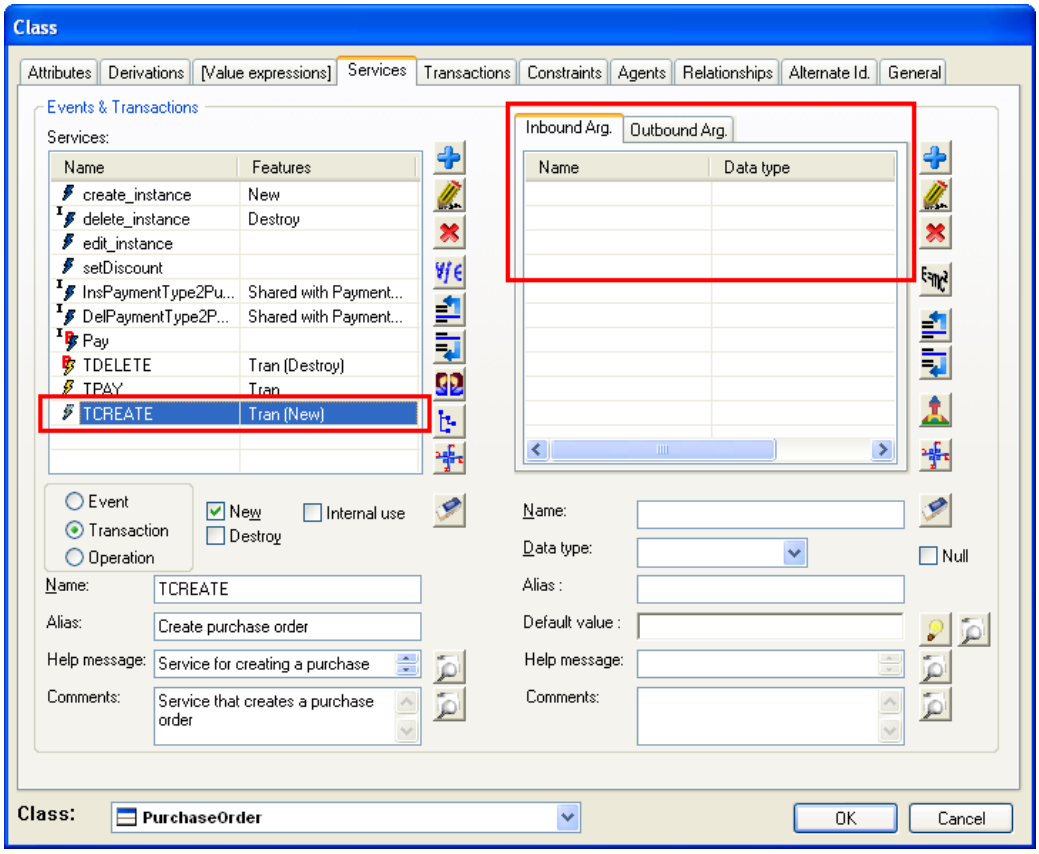

Figure 3 No arguments are needed in the new creation transaction

In the new transaction formula, we call to the original creation service using "*AGENT_Customer*" as argument.

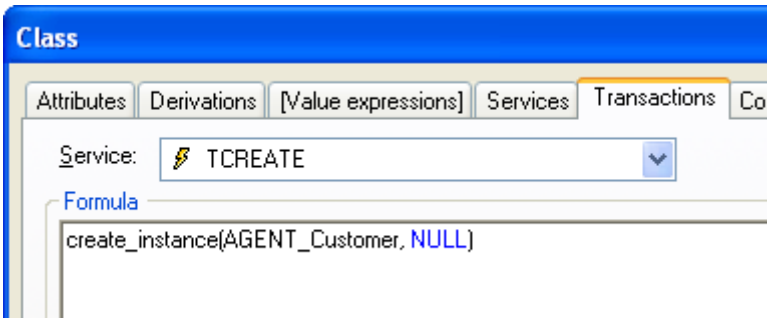

Figure 4 Original 'create_instance' encapsulated in the new creation transaction

Note: In the original service, there is an argument named 'p_agrPaymentType', representing the payment type that the customer will use to pay the purchase order. This information is only required during the payment process, so no value (*NULL*) is indicated during the creation.

After that, the original 'create_service' event should not be visible to any interface. Then, you can mark it as internal.

Figure 5 Original create service marked as internal

4 Admin: *I* verify purchase orders

On the other hand, every purchase order must be checked by an administrator before it could be served. Besides, you need to know which administrator verifies each purchase order. For this reason the administrator name have to be stored in every checked purchase order.

In the model, you need to add a new attribute '*VerifiedBy*' in the 'Purchase Order' class where the administrator name will be stored, and also a new event '*SetVerified*' in charge of setting the name to the attribute.

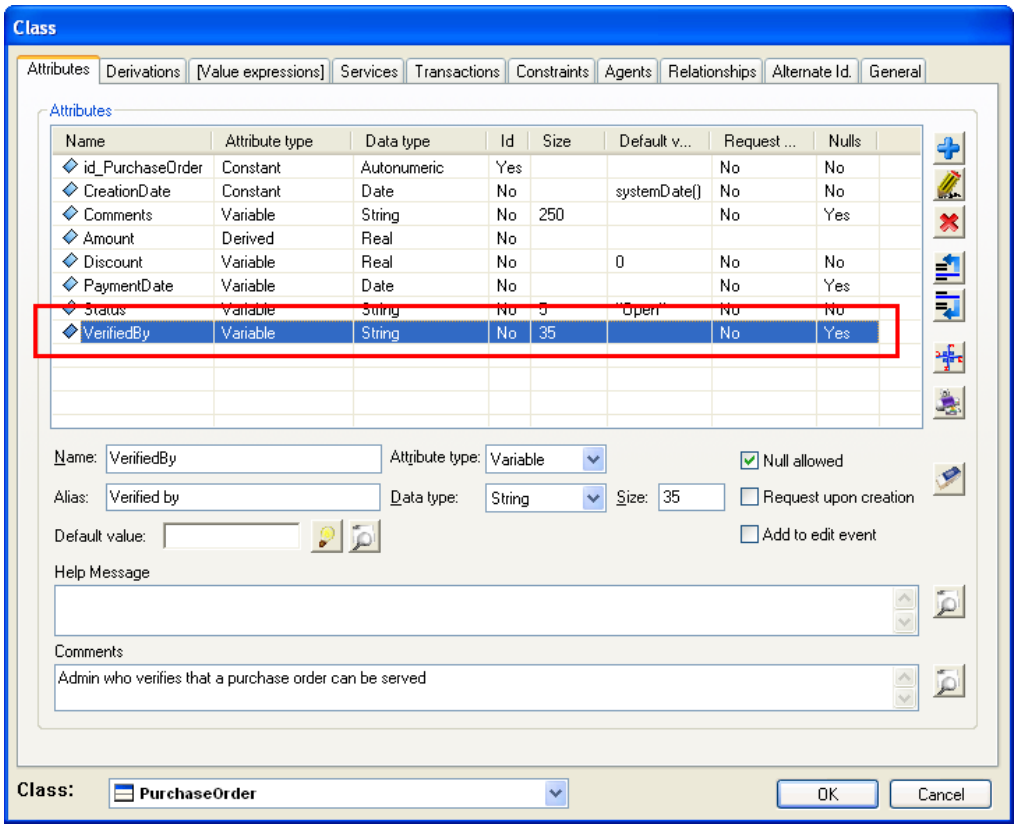

Figure 6 Attribute to store the administrator name

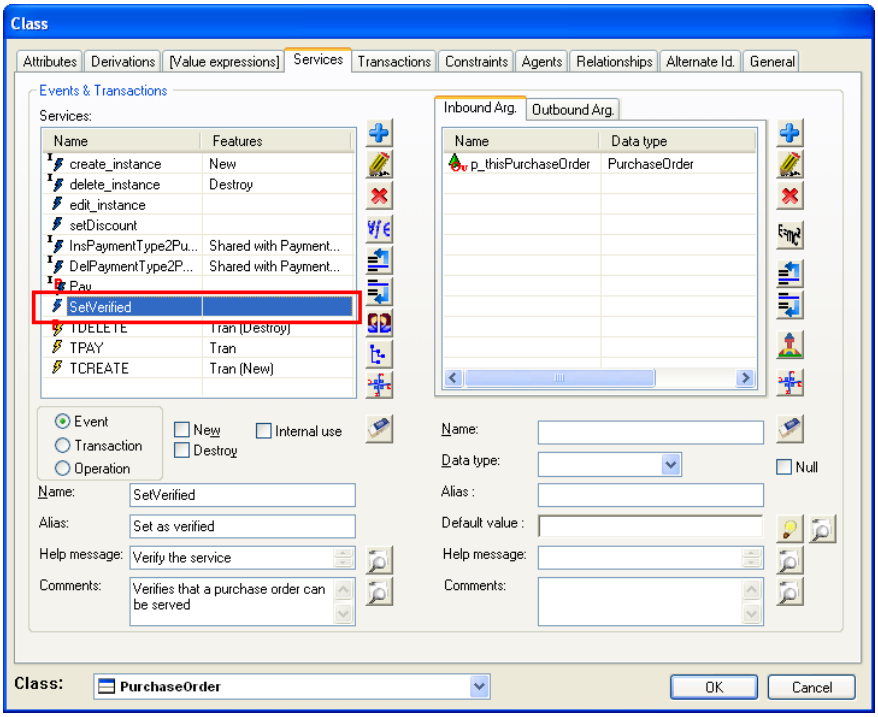

Figure 7 Event to store the administrator name

After creating the 'VerifiedBy' attribute and the 'SetVerified' event, you must add an evaluation in the event to set the administrator name in the attribute. In the effect of this evaluation you should indicate "***AGENT_Admin***.*AdminName*".

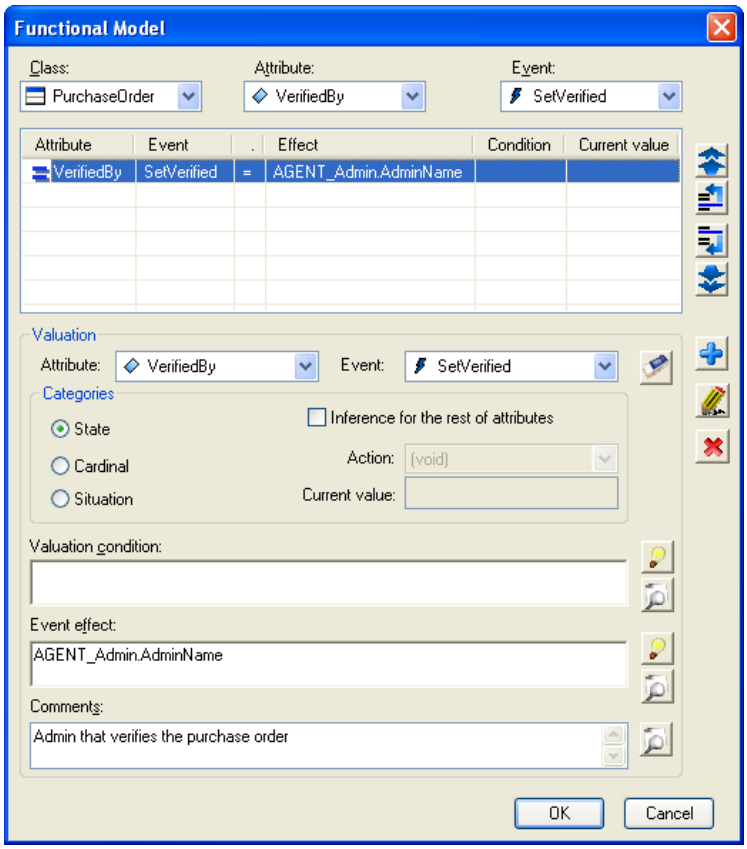

Figure 8 Evaluation in the '*SetVerified*' event to store the administrator name

5 Customer: If I am disabled *I cannot* create purchase orders

An administrator must be able to enable and disable customers, since a disabled customer could not create new purchase orders.

First of all, the 'Customer' class needs a new attribute 'ActiveStatus' to store the enabled/disabled status, and a new event 'ChangeActiveCustomer' to modify it.

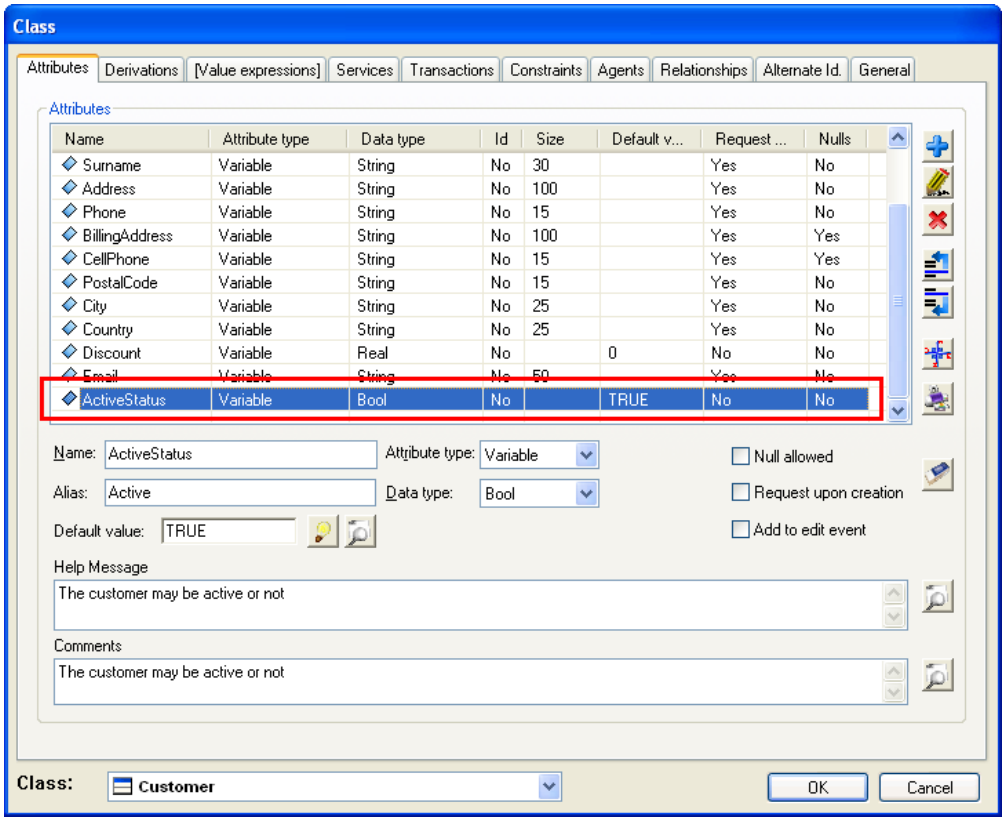

Figure 9 Customer attribute to store the enabled/disabled status

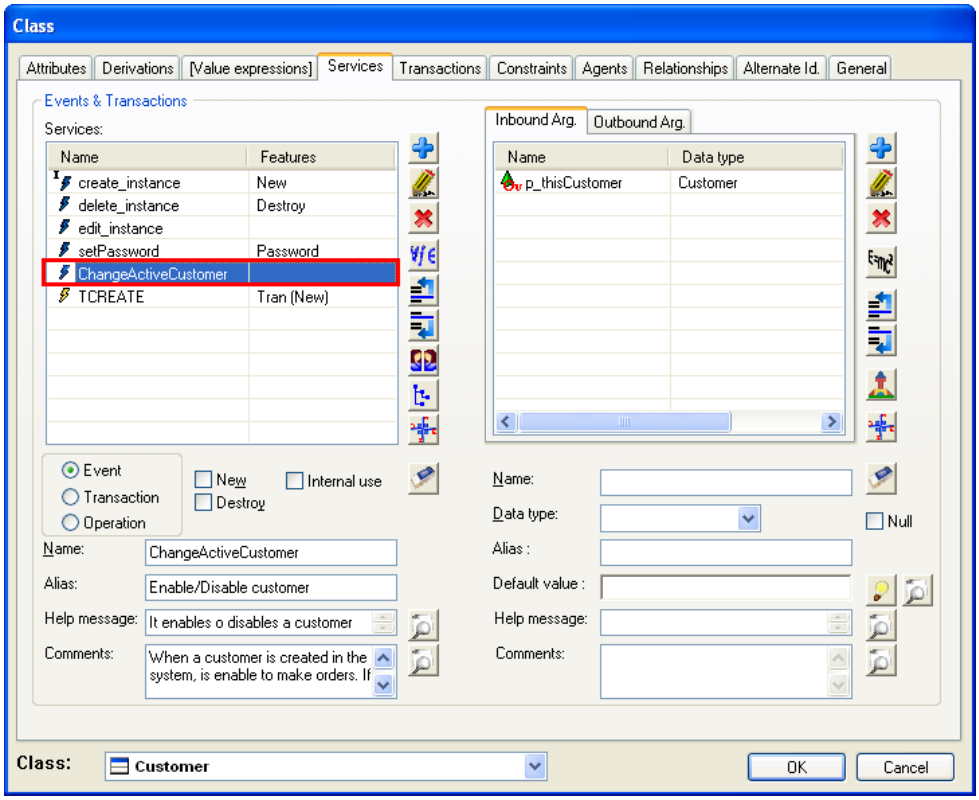

Figure 10 Event to modify the enabled/disabled status

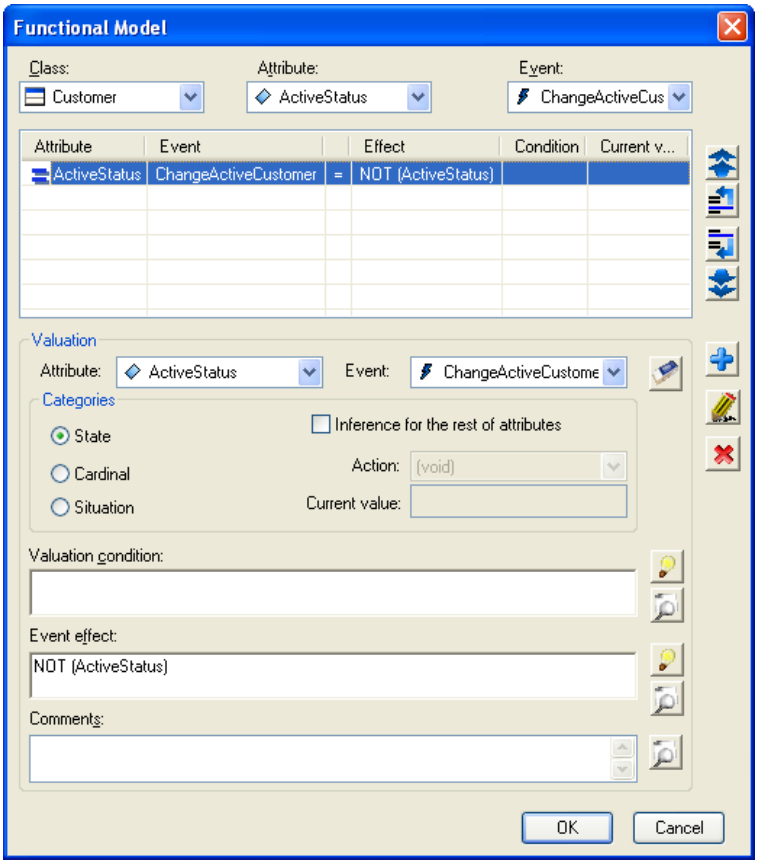

Figure 11 Evaluation to switch the Customer enabled/disabled status

Besides, you need to add a new precondition in the *TCREATE* transaction of 'PurchaseOrder' class. This precondition will check if the logged customer is enabled. In case the customer is not enabled, the application will show the following message: "*Not possible to start shopping. This customer account has been disabled. Contact the support department*".

The formula to retrieve the logged customer status is "***AGENT_Customer**.ActiveStatus*".

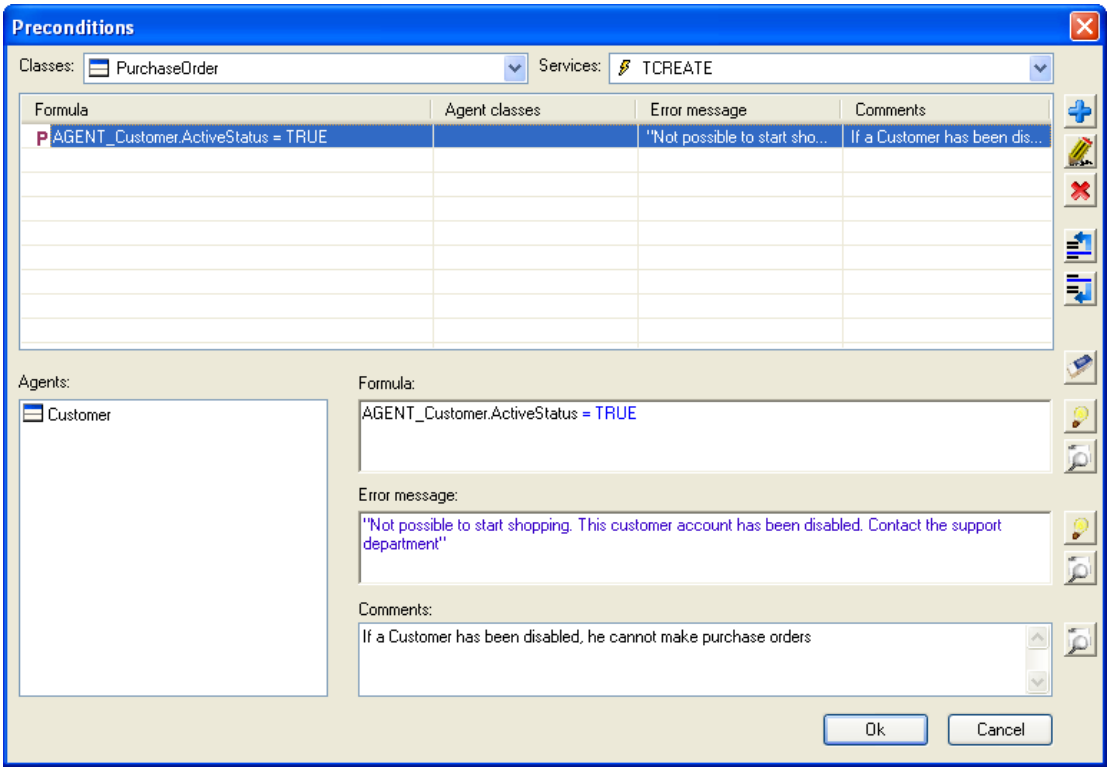

Figure 12 Connected agent in precondition

6 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 **non-internal 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.