



# FUEGO



## CANCELLING SCREENFLOWS

February 8, 2005

### 1 Introduction

The following document shows how to design a screenflow in such a way that the user can cancel it at any point, reverting any change performed during the screenflow.

## 2 Motivation

Screenflows are a series of interactive activities which handle the user interaction with Fuego. Each interactive activity can be a Fuego Object presentation, display statement, input statement or JSP page. Eventhough each of these elements provide a *cancel* option (or button), that option does not cancel the complete screenflow, just that interactive activity. Cancelling the last presentation in a screenflow will *not* automatically revert the changes submitted in previous presentations belonging to the same screenflow.

For example, if a screenflow which edits a Fuego Object, has two presentations, and the user submits the first one, any changes performed in that presentation will be committed eventhough the user may then cancel the second presentation.

A way to cancel a whole screenflow needs to be introduced at design time.

## 3 Applicability

Use this design pattern when editing a Fuego Object using one or more presentations.

## 4 Structure

In our example, we will be dealing with a simple process which calls a screenflow passing it a Fuego Object to be edited.

### 4.1 Main Process

We have created a process which will call a screenflow passing along and receiving back a Fuego Object of type *Supplier* to be edited.

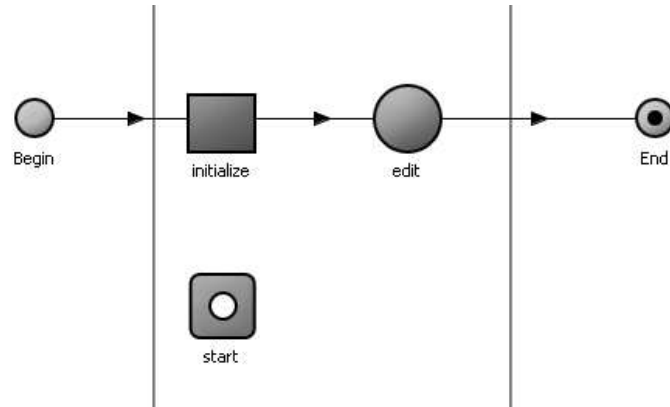


Figure 1: Main process layout

To simplify our example, our main process has just two activities. The first one is an automatic activity where we initialize an instance variable of type *Supplier*. The second activity is an interactive activity which calls the screenflow passing the *supplier* instance variable.

## 4.2 Screenflow

Our screenflow starts with an automatic activity which initializes an instance variable of the same type as that of the object being edited. However, this variable is assigned a *clone* of the object received as an argument and not the original object. It also has two interactive activities which will edit the clone created at the beginning of the screenflow. Each one will use a different presentation of the *Supplier* Fuego Object to edit different attributes of it. At the very end, an automatic activity will then decide if the modified clone object should return to the main process or if the user has cancelled the screenflow and then the original, unmodified, object should be returned back.

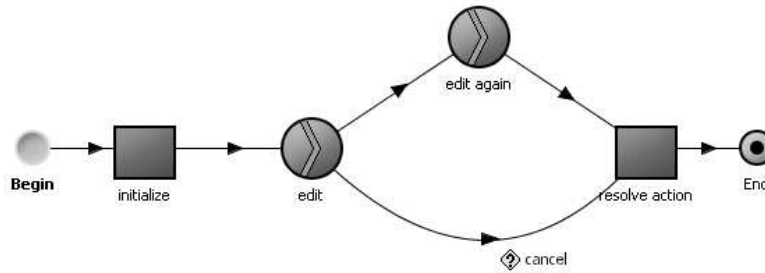


Figure 2: Screenflow layout

## 5 Implementation

To implement the cancelling operation we first need to design the screenflow that will be performing the Fuego Object editing.

### 5.1 Variables

We will need to create a screenflow and specify the following instance variables:

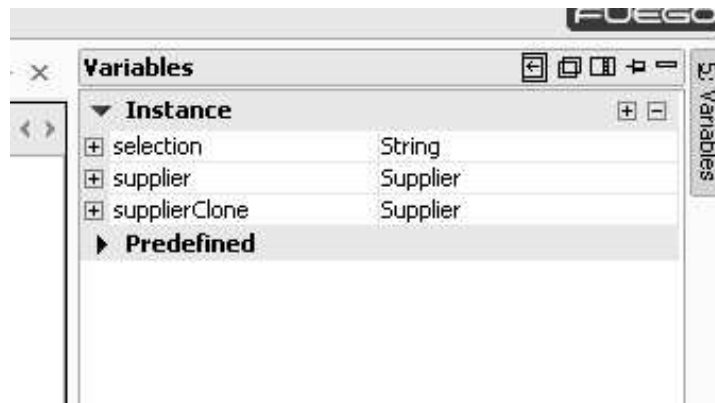


Figure 3: Screenflow Instance Variables

- The first variable, *selection*, will let us know which button the user pressed on each presentation.

- The variable *supplier* will hold the Supplier object that the main process wishes to edit.
- Finally, the variable *supplierClone* will hold a clone of the Supplier object which will be the actual object that the screenflow will be editing.

## 5.2 Arguments

Our screenflow will also need the following input arguments:

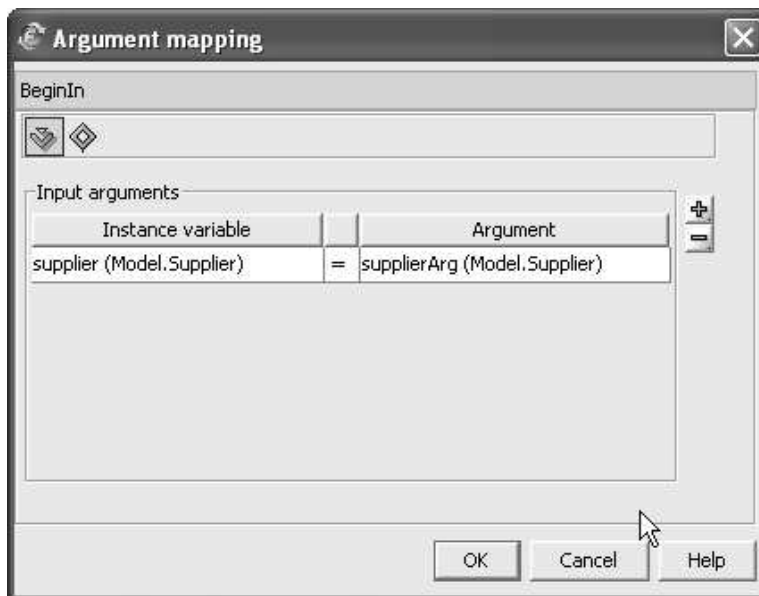


Figure 4: Screenflow Input Arguments

The input argument *supplierArg* will reference the Fuego Object that the main process wishes to edit. We will assign that argument to the screenflow's instance variable *supplier*.

On the other end of the screenflow we will need the following output arguments:

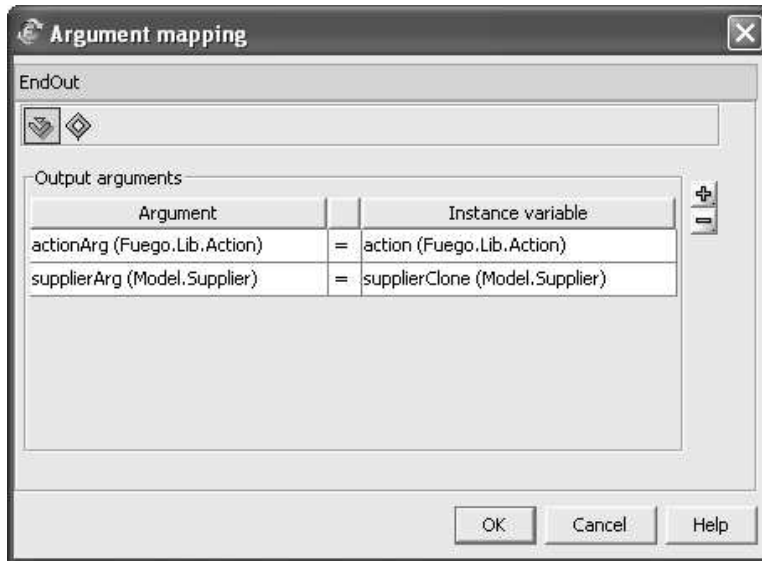


Figure 5: Screenflow Output Arguments

We will have two output arguments, the first one, named *actionArg*, will reference an object of type *Fuego.Lib.Action* which will indicate the main process which action it should be taking. The second argument, *supplierArg*, will carry the result of the screenflow editing. If the user has cancelled the screenflow then it will carry the original supplier object (kept, unmodified, in the *supplier* instance variable) or the modified clone. We will assign the right object to be returned in the last automatic activity of the screenflow (see below).

### 5.3 Activities

The screenflow starts with an automatic activity named *initialize* in our example. this activity just initializes the *supplierClone* instance variable with a clone of the original object received from the main process.

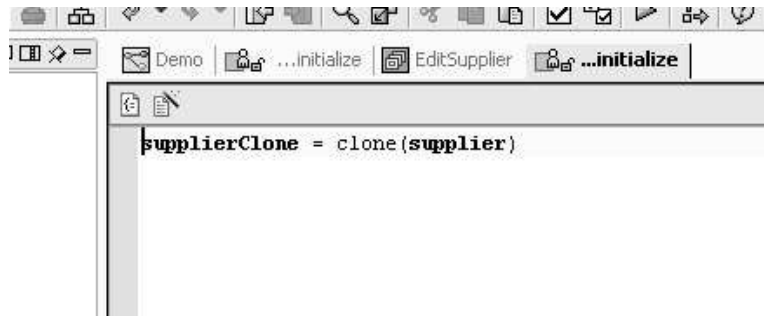


Figure 6: Initialize Automatic Activity

Next an interactive activity called *edit*, lets the user edit the cloned object stored in the *supplierClone* instance variable by display it through the *Edit* presentation.

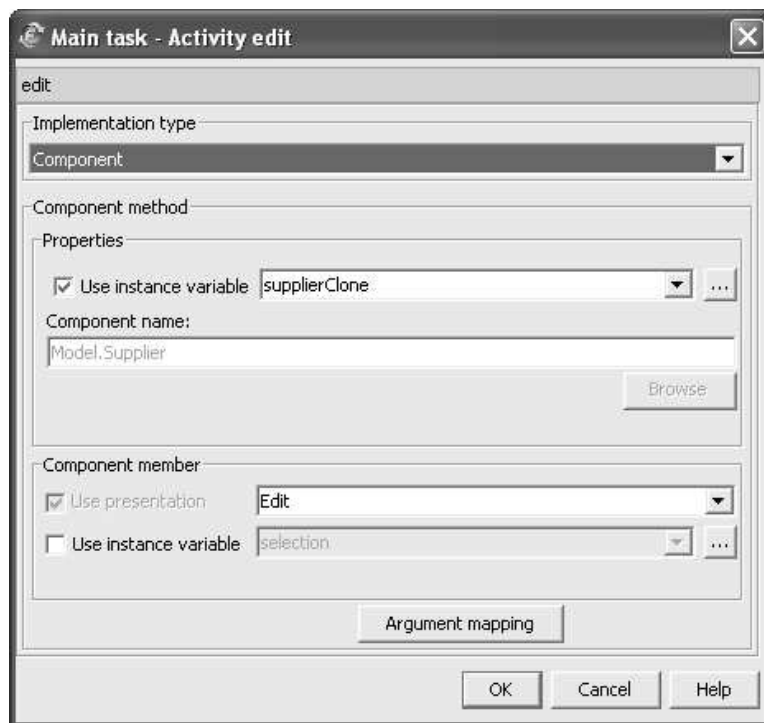


Figure 7: First Interactive Activity

The presentation has no input argument defined, but it does have one output argument. This output argument is the *selectedButton* argument which indicates what button the user pressed. We will store this value in the *selection* instance variable. In case that the user pressed the **cancel** button then the value returned in the *selectedButton* argument will be `null`, otherwise it will be the name of the button the user pressed.

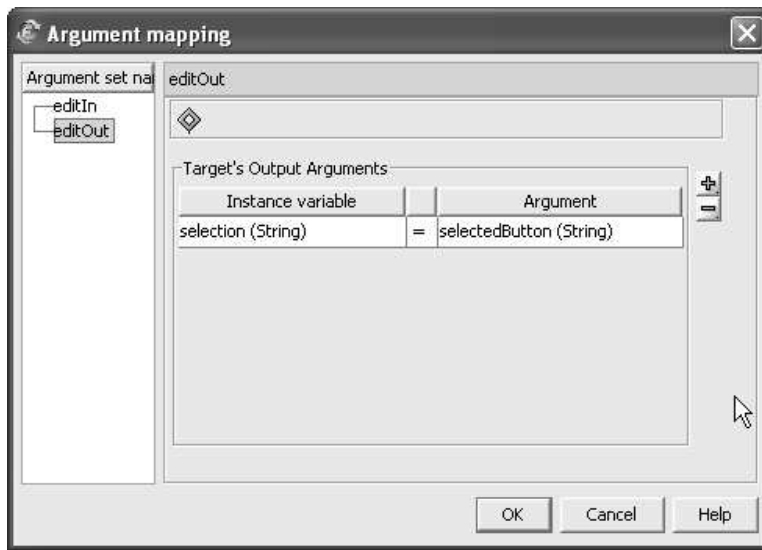


Figure 8: First Interactive Activity

This activity has two outgoing transitions. If the user pressed cancel the the flow is directed to the last automatic activity in the screenflow (see below). If not a second presentation is displayed using the second interactive activity present in the screenflow. This second interactive activity has the same properties as the first one, except that the presentation used is different.

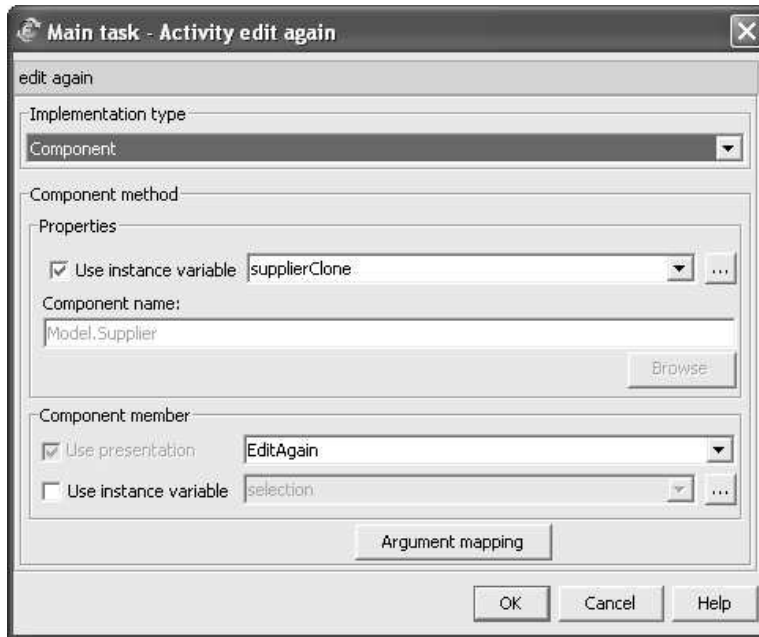


Figure 9: Second Interactive Activity

The argument mapping is the same as the one used on the first interactive activity.

Finally, an automatic activity determines what information should be passed onto the main process.

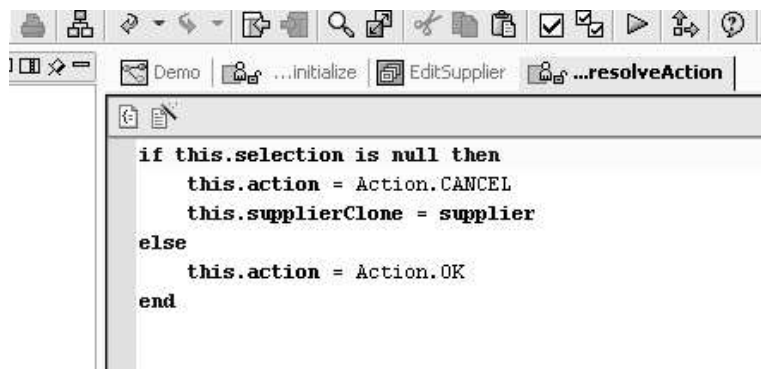


Figure 10: Resolve Action Automatic Activity

Note that the code checks what is the value of the instance variable *selection*. This variable indicates which was the button pressed by the user. In the case that the value is null (the user pressed **cancel**), then the action to be returned is `Action.CANCEL` which indicates the parent process that it should cancel the interactive activity execution. Also, the cloned object is replaced by the original, supplier object; in effect reverting any changes performed.

Otherwise, the action returned is `Action.OK` indicating the main process that it should continue the normal flow of the instance.

In this manner, should the user cancel the screenflow at any point, the original unmodified object can be returned as the result of the screenflow. Should the user finish the screenflow without canceling then the clone will be returned as the result of the screenflow containing all the modifications made by the user. Note that in both cases the variable returned is *cloned-Supplier*. In one case it contains the modified cloned object, and on the other the original object.

#### 5.4 Main Process

The main process will call the screenflow we described above through an interactive activity.

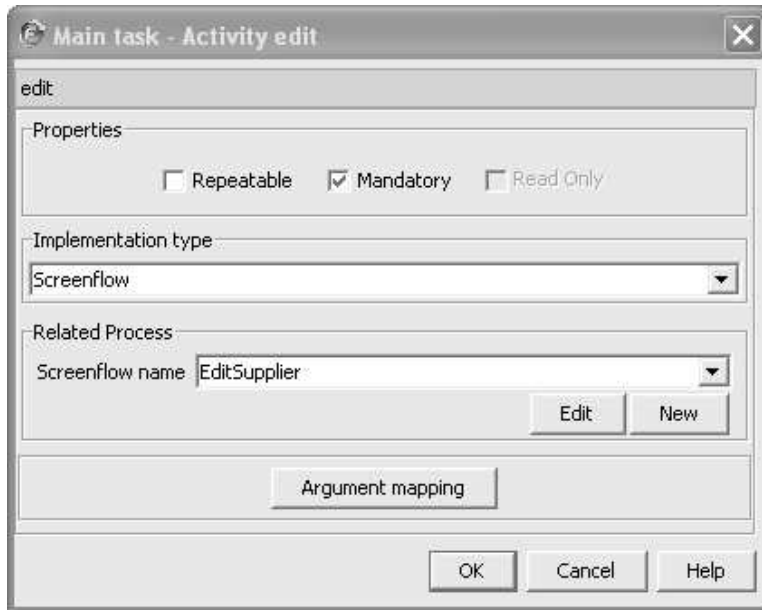


Figure 11: Main Process Interactive Activity

This activity provides the supplier object to be edited as the value for the single input argument required by the screenflow.

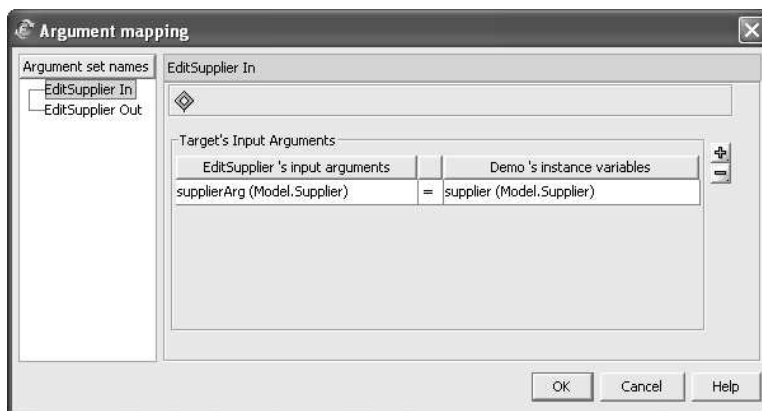


Figure 12: Main Process Interactive Activity Input Arguments

It also receives back the two arguments provided as output for the screenflow.

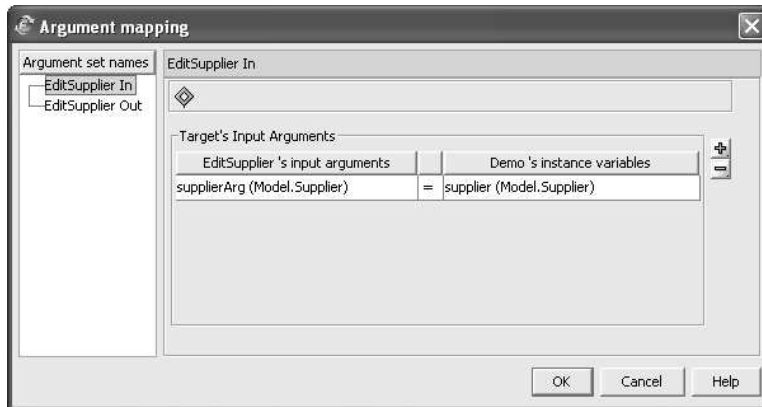


Figure 13: Main Process Interactive Activity Input Arguments

Note that it is effectively replacing the original supplier object (stored in the *supplier* instance variable) with the value returned by the screenflow in the *supplierArg* output argument.

One last thing that needs to be pointed out is that Fuego will automatically cancel or not the execution of the activity depending on the value contained in the action predefined variable at the end of the activity's execution. There is no need for you to do anything else than to receive the *actionArg* value into the *action* predefined variable.

## 6 One Presentation Screenflows

In case that your screenflow only has one presentation, then the cloning of the object does not need to be included in the design. You can design your screenflow in such way that it edits the same object that the main object passed along. Just setting the action returned to the main process to `Action.CANCEL` at the end of the screenflow is enough to revert any changes performed to the object being edited.