No-Code COSE Developer Users Manual

About the Document

This document provides information about how to use the tool, through development examples. There are different examples, beginning with a Travel Reservations system and an Automated Teller Machine (ATM) machine.

The tool is intended for development of any software application by re-using or developing the following three categories of items:

1- A Graphical User Interface design: Drawing the screen elements by drag-drop actions,
2- Process Model Development: Specifying the order of operations for the application through graphical drag-drop actions, and
3- Acquisition of a set of operations (components, services etc.) and connecting them to the process model.

The development efficiency will be better if a mature domain environment is already provided: Such an environment supplies all predictable operations as parts of readily available components, and optionally a set of template process models (or partial models).

General Approach

Suggested order of development starts with the specification of the Graphical User Interface (GUI). What an application does is presented in the opening screen, as a set of functions to be started with these GUI elements. Such a screen may correspond to a mobile application or a desk-top screen. The items in the GUI are directly related with the tasks (system operations/functıons) they start.

A system operation in return, is represented in a process model. For each operation in the GUI menus, one specific process model should be started. Therefore, the GUI elements will be connected to the process models. Of course these process models should be developed also. As a result, at run time, clicking on one menu item will start the execution of one process model.

A Process model is created with its defined beginning point and a set of activities. Those activities are ordered after the beginning point: they could simply be in a sequence allowing the next one pointed by the arrow from the previous one to start when previous one finishes. In some cases, there may be an alternative 'next activity', rather than a straight single activity. In that case, decision nodes are placed after the previous activity and before the set of alternative 'next activities'. To place the activity boxes, decision boxes, and the connection arrows in a process model, they are selected from the tool bar and dragged. Finally, it is necessary to indicate who (which one among the existing software program units) will conduct the duties of those activities. So far, the activities are only represented by their names in the activity boxes. Now we need to get in the box and tell it who should execute for it. This is done by connecting the activity box to the component functions ("methods").

To connect an activity to an operation, a developers selects a method among an existing wide variety. However, the operations are organized as internal members of components. After selecting a component, its methods are available and one method should be picked. This method stands for the operation to connect with the selected activity through easy ways the tool provides. The components are presented similar to the way the products are presented in an e-commerce site. They should be catalogued. Your tool may present them in a diagram where you start from the top and select the lower-level categories by navigating downward that take you to a group of components. Once a component is located, an operation in it can easily be selected. As a summary, a hierarchical organization of components is offered to serve as the catalogue.

The following sections will show how to conduct those steps for software application development.

Example 1. Travel Reservation System

In this example we are developing a system where a user can make reservations and buy a ticket for airplane travels. Also the system is maintained by the Admin who defines and modifies lists of trips, employees, customers etc.

Step 1. Design the User Interface

Figure 1 depicts the view of the GUI that should be developed. This figure displays all the menu items as if they would show up at the same time. However, when the application executes, actually only the two words at the top menu bar will be visible. The user should select one main menu. There are two main menus in this picture that are the User and the Admin. When the mouse comes over these menu items, the bottom parts (pull-down menu) also show up. Selected **pull-down menu** appears stronger and especially a selected **menu item** in it is highlighted. The sub-menu items will be displayed only if the mouse comes over these main menu items (User or Admin). You are expected to create these menu structures through the easy to use drag-and-drop environment.

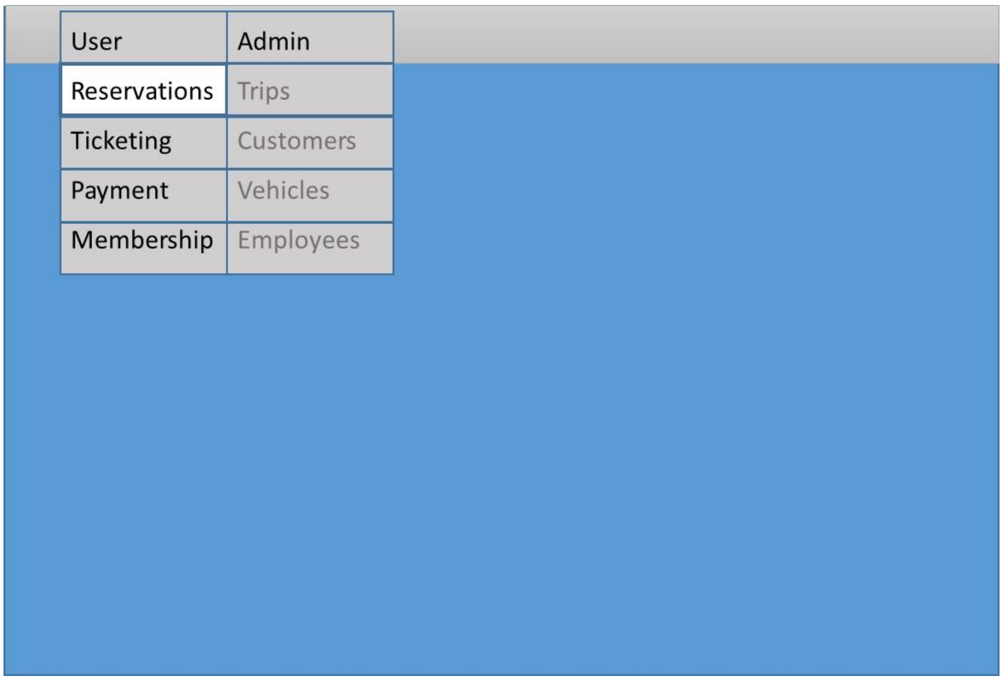


Figure 1. General Graphical User Interface for the Travel Reservation System

Step 2. Draw the Process Models for the menu items and connect

In general, one separate process model should be developed for each menu item. We will start with the “reservation” function. Figure 2 displays the finished process model for the reservation function. How to develop this process model will be explained later.

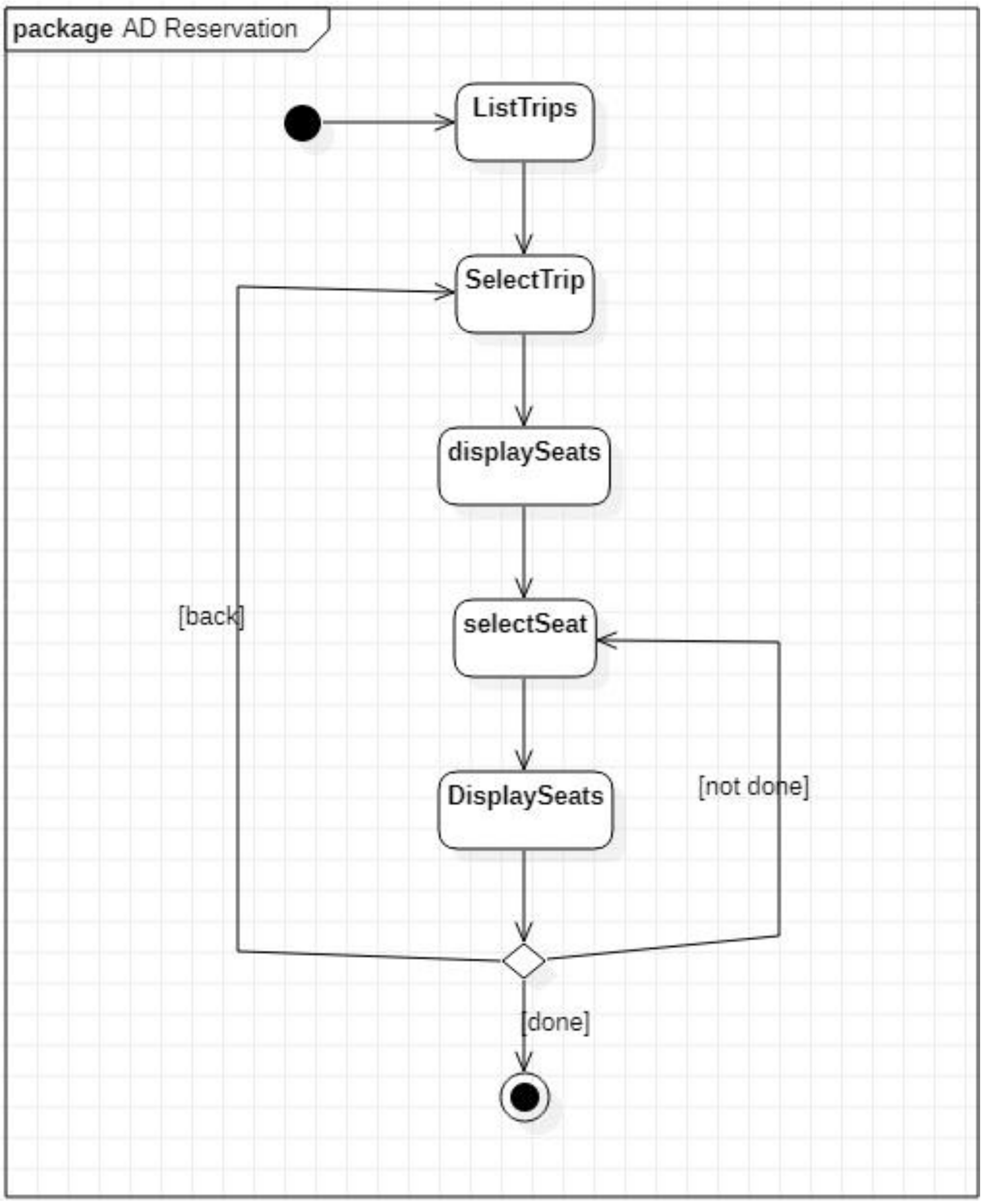


Figure 2. Process Model for the Reservations function

The process model suggests that reservation operation starts with ‘list trips’ task. Then the user should select a trip from the list, which is handeld by the ‘Select Trip’ task. After the trip is selected the user wahts to display the seat layout before selecting a seat. Displaying the seats and selecting a seat are handled by the corresponding operations. Selecting a seat would end up with another display to show the newly selected seat in a highlighted color. At this point the user may want to start all over by selecting another trip, that is why the connection back to the SelectTrip activity is provided at the left. Alternatively. The user may continue changing seat selections that would be enabled by the arrow on the right that goes back to the SelectSeat activity. When done with seat selection, the process will end which is modeled with the arrow at the bottom (done) to the finish circle.

One process modeling example for one menu item has been completed in the paragraph above. Note: the process model is provided as a UML Activity Diagram. Later examples may use the BPMN diagram as an alternative. These examples start with more simplistic functionalities and diagram types.

Another process model can be shown here for the admin process that is for managing the list of trips. Figure 3 shows the related process model.

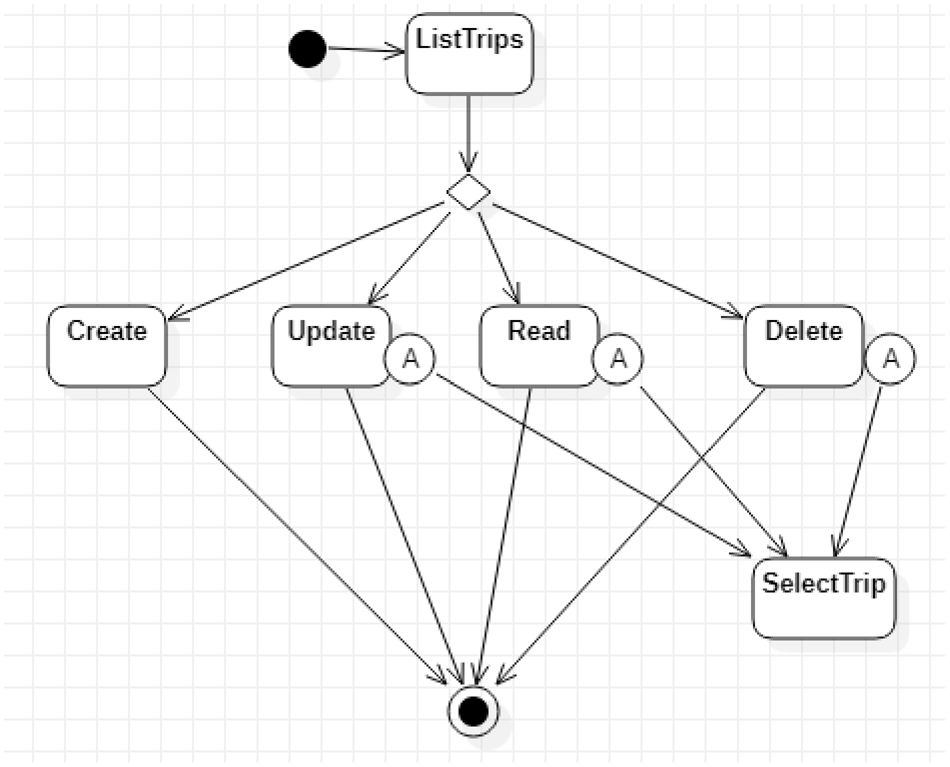


Figure 3. Process Model for the Manage Trips function

In this process, we expect the admin to first see the list of the trips before any action. Then there are options to Create a new trip, Update an existing one, Read (see the details of a trip), or Delete a trip. Except for the Create option, the other three functions also require the selection of the trip. Therefore those three functions also call the select trip activity.

Step 3. Operations

Figure 4 displays a "component model" that we will utilize in this example as a a catalog for operations. Here, a sub category is drawn inside a main category. For Example, Airbus350 is drawn inside Airplanes. Our goal is to connect the activity boxes in the process models to the operations in the component model. In other words, we are telling who will do the job.

Note: this diagram is a UML Component Diagram. In later models, there could be more 'component technology specific' alternatives used, such as COSEML diagrams.

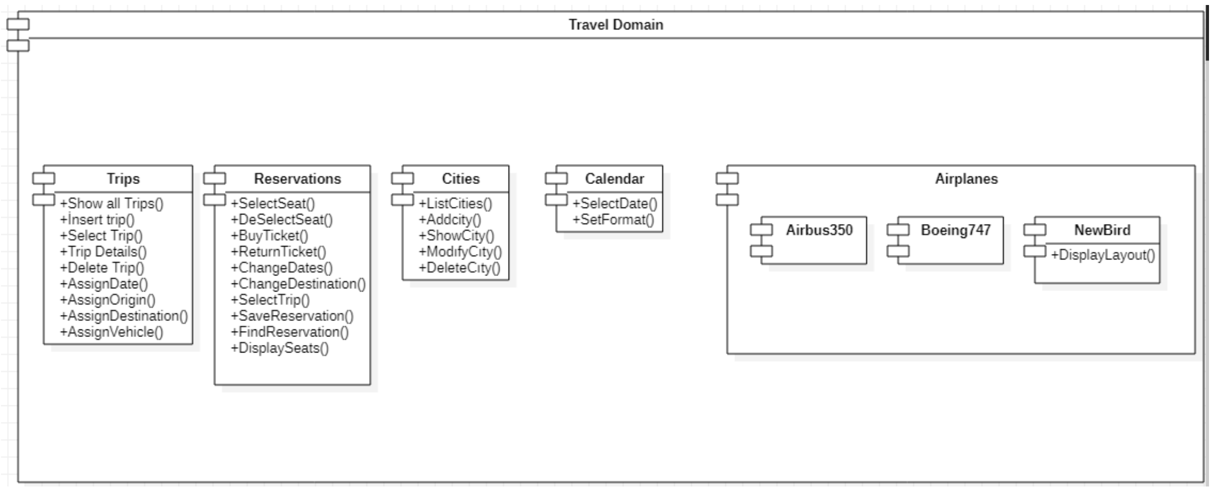
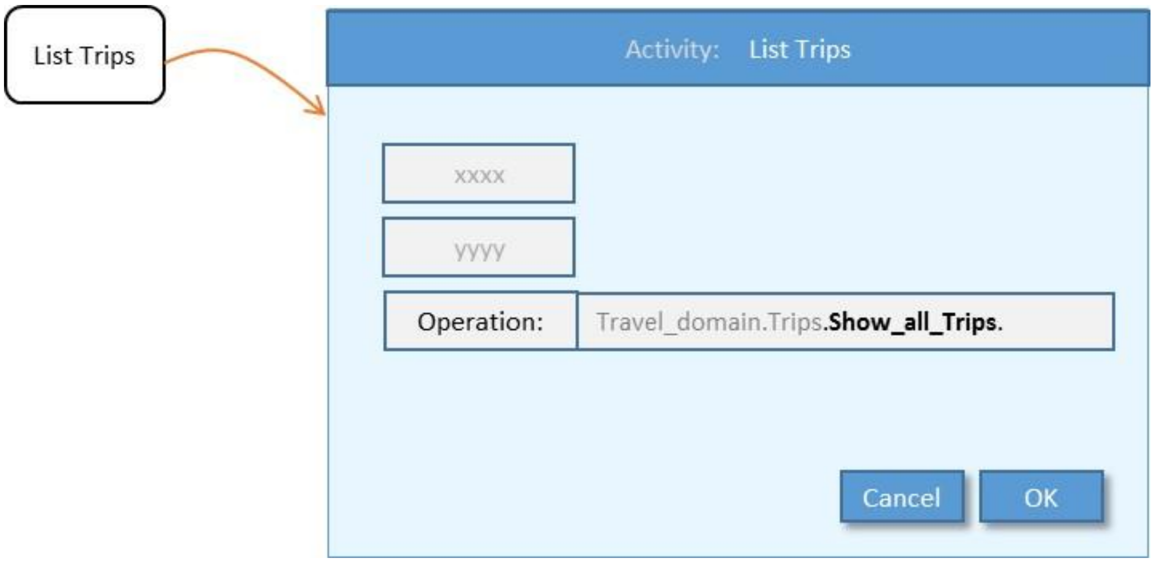


Figure 4. Component model for locating the operations (format: UMLs Component Diagram)

Let us start with the 'list trips' activity in Figure 2 (Process model for reservations). A right click on the list trips box and then selecting properties or double clicking on the box should take us to the screen to do that. Figure 5 shows the dialog box corresponding to the mentioned screen.



Operation: Travel_domain.Trips.**Show_all_Trips**.

Figure 5. operation connection in an activity box

It is assumed here, that the operation box is selected and to its right, the long box is clicked on to provide an operation name either by typing or better: Go to the component model and select by clicking, the show_all_trips operation in the component model (inside the Trips component). The name of the operation preceeded with its component name will be typed here automatically to identify the item you clicked that is connected.

So far, a minimal explanation for developing an application has been demonstrated. It is helpful at this point to mention some additional information. One step we skipped was the connection between the menu items in the Graphical User Interface and the processes. That is very similar to how we connected an activity to an operation in the previous paragraph. After selecting the menu item (in the design environment – not when application is running !) its properties will show up in a dialog box. Then, click on the process/operation field and then click on the name of the process model in your tool where process models are listed. Alternatively, you can write its name, or if its window is open, click on the "start" circle (black filled) in the corresponding process model.

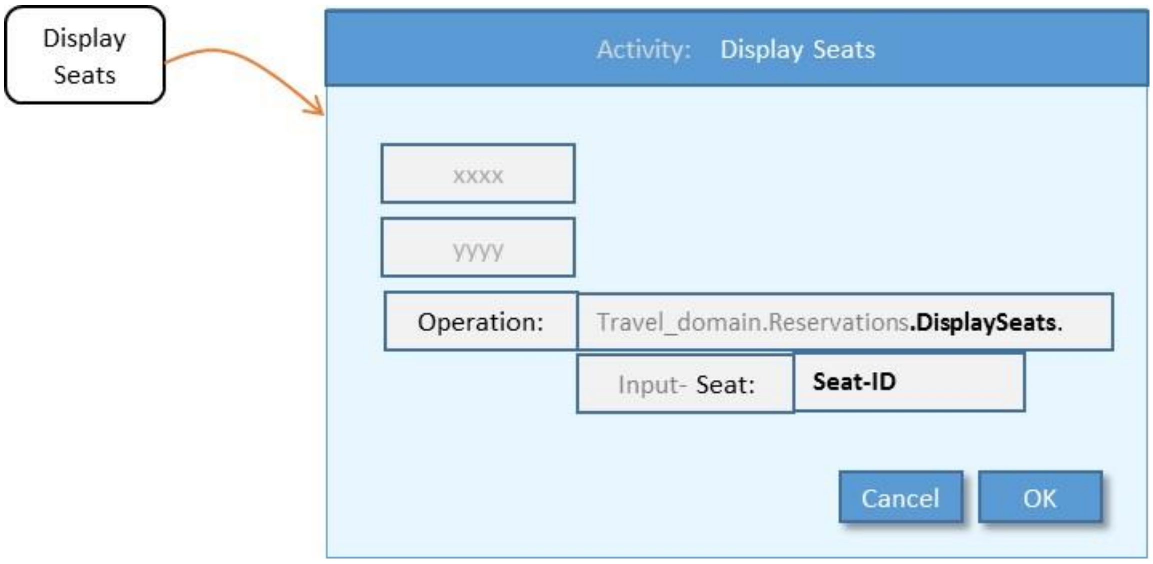
Details

For the application you have created to run successfuly, some homework needs to be done. For example, in your first execution of the application, when you list the trips, an empty list will show-up. That is because, as expected, the ‘admin’ has not added trips to that list yet.

During development, selecting an operation for an activity is not all that is required sometimes: Some operations need further information to be able to compute. For example, drawing the layout may require the reservation information. It could draw the layout but without painting your selected seat in a different color. Therefore, it requires your seat-selection information. If an operation needs such a ‘parameter’ you have to provide a value for it. You could write here, for example ‘25’ as your seat number. Or better, this number was stored after your seat selection operation in a variable and you can write the name of that variable: whatever was selected, that should be passed to the drawing function.

Actually, in general a process model besides ordering the activities, should be able to ‘save’ the result of an operation and use that saved value later as parameter to be sent to another operation. There are ‘variables’ in programming languages for the purpose of saving a value and using it later. Variables can also be declared, values can be written into them and later these variables can be used for sending to an operation, if the operation needs such a value. An example: we can declare a variable called “seat number”. Most probably, the “select seat” operation will return the number or ID for that seat. In the activity box that is linked to the “select seat” operation, there should be abilities to read this returned value. Better, there could be a slot in the activity box that asks you to write the name of the variable to store the returned value from the operation.

Likewise, when you are connecting your activity box to an operation using the text boxes in the corresponing screen (dialog box), there should be other slots (text boxes) for providing the input parameter for this operation. There were no such text boxes in Figure 5 – the only example included so far for connecting to an operation. The reason for their absences is simply because the “list trips” function does not require input parameters, neither does it return any output parameters. Figure 6 provides a more detailed screen that is for the “display seats” activity for its connection to an operation.



Operation: Travel_domain.Reservations.**DisplaySeats**.

Input- Seat: **Seat-ID**

Figure 6. operation connection with input parameter

In Figure 6, it is assumed that the connected function (DisplaySeats in the Reservations component) requires an input parameter that is coded as “Seat”. That is why your tool writes this “input- Seat” prompt and expects you to provide the variable name (Seat-ID). As the information you provide in the box to its right (Seat-ID), we assume this is the name of a variable. We further assume that this variable was filled with value as a result of a previous activity: when the ‘select seat’ function was completed, it provided the seat ID as a return parameter and we specified to the tool to save what is returned, into this variable (Seat-ID). Consequently, the seat information comes from the “select seat” and goes to “display seats” functions (so that during the layout draw, this seat can be highlighted). So, in Figure 6, you supply the variable name (Seat-ID). As well, you provide the same variable name to the select seat operation as shown in Figure 7. Actually, the presentation of those two activities corresponding to Figures 6 and 7 should may be reversed (7 first, then 6) to represent a more intuitive development chronology. In reality the order does not matter.

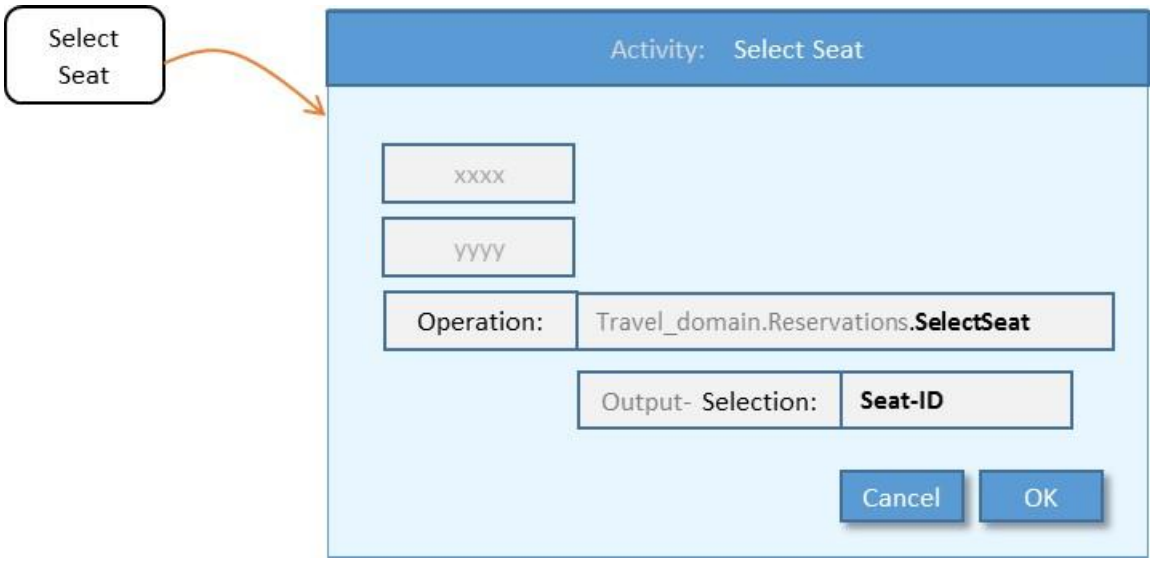

Select Seat
Activity: Select Seat
xxxx
yyyy
Operation: Travel_domain.Reservations.**SelectSeat**
Output- Selection: **Seat-ID**
Cancel
OK

Figure 7. operation connection with output parameter

Likewise in Figure 7, you provide the variable name (Seat-ID) as a place to store what the operation will give us. (Hence, you have also declared this variable here: if you completed the actions in Figure 6 before declaration is done there – at its first mentioned time). The tool knows that this operation will return a parameter whose name is ‘Selection’.

Another capability to know is the process models “calling” another process model. One big process may start and in the middle may require another to be executed before it continues. This way, part of a process or a whole process l can be “re-used” by others, without having to be repeatedly defined as a part in a bigger model. For example, user authentication can be initiated by different process models before those models continue with their specific activities.

Example 2. Automated Teller Machine

The GUI for this project is for the special small screens to be found in the ATMs. Our example is a small one, not having many functions. This application will accept deposits, withdrawals and transfers.

Step 1: User Interface

The GUI is displayed in Figure 2.1

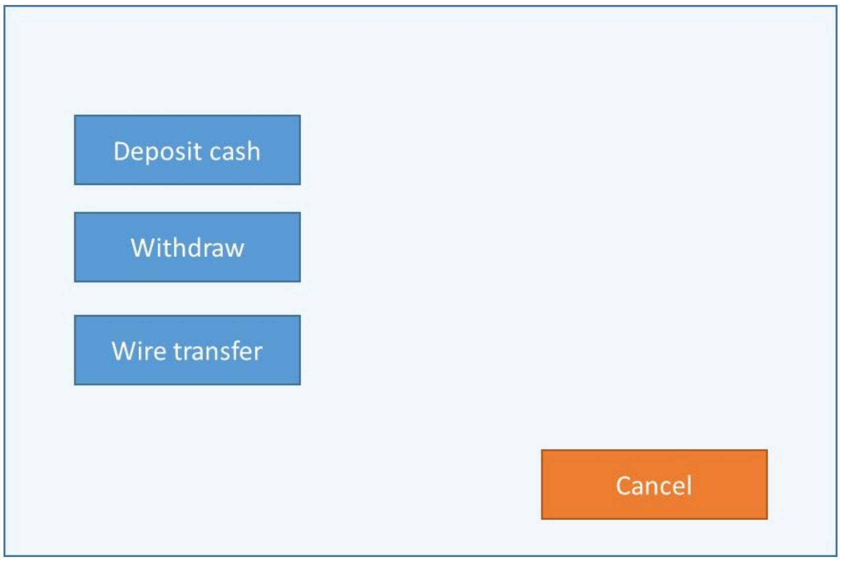


Figure 2.1 GUI for the ATM example

Step 2: Process Models

In this example, we assume an overall process model for the complete application. This process model is depicted in Figure 2.2 and embeds all the menu items.

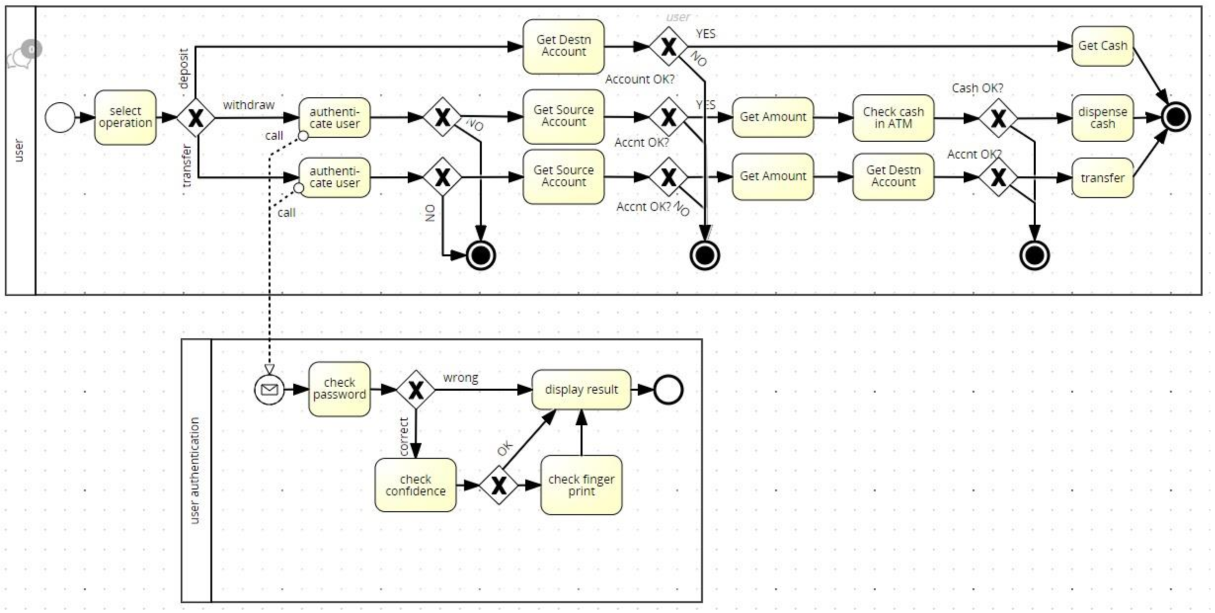


Figure 2.2 Process Model for the Deposit Cash function

Such a global process model assumes the control of the program execution from the beginning (Before the main menu). It activates the main menu and the menu selections are shown as the first operation ('select operation') in the process model.

Note: Figure 2.2 provides the process model in BPMN format,

Alternatively, if the main menu is meant to be the first item to execute (take the control at the beginning) then the functions it offers could be modelled as different processes. In that case, the first process is for the Deposit Money function whose model is provided in Figure 2.3.

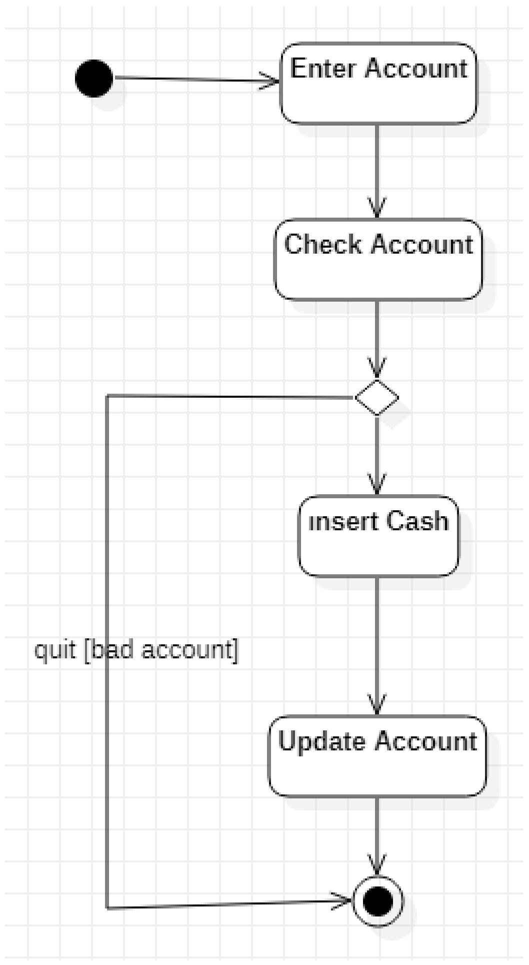


Figure 2.3 Process Model for the Deposit Money function

Step 3. Operations

Here, the component model is presented in COSEML format. The functions are avaliable at the lowest-level boxes that are the interfaces for components or web-services.

Let us now try to make connections reequired to complete the 'deposit' system-function, starting from the beginning of the application. We will pick the overall process model shown in Figure 2.2.

A little note: At first, we will assume some simplicity: all the activity boxes can be implemented by one function in the component model. In fact, we will see that this process model does not contain enough activities to accomplish the system functions with the existing components. We will keep that secret until it is time for changing the process model to match the existing operations. Now let us proceed with the simpler assumption.

The first activity is 'select operation'. Getting into this activities box with a right click, we see the details and fill the operation box. Usually we fill this box with a component (or web-service) name followed with a methods name. This time, we are starting the main menu because the only duty of this operation is to start the main menu and get the result of the users selection.

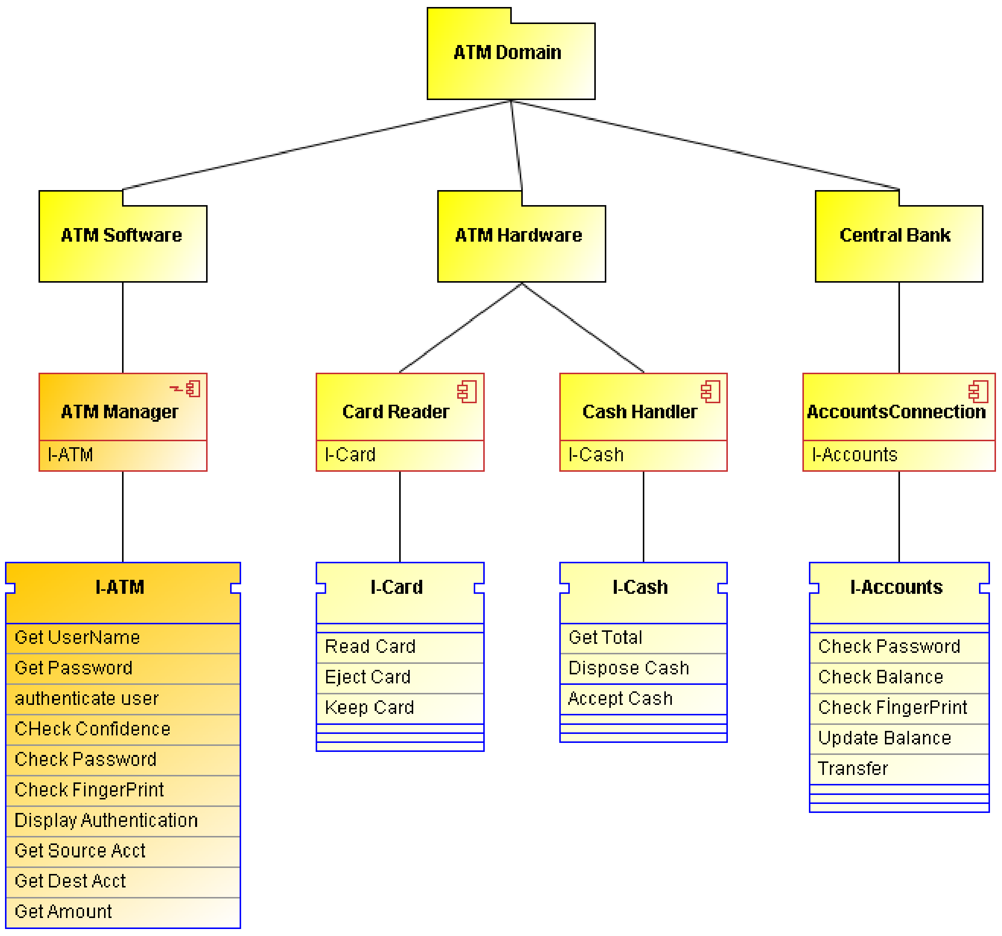
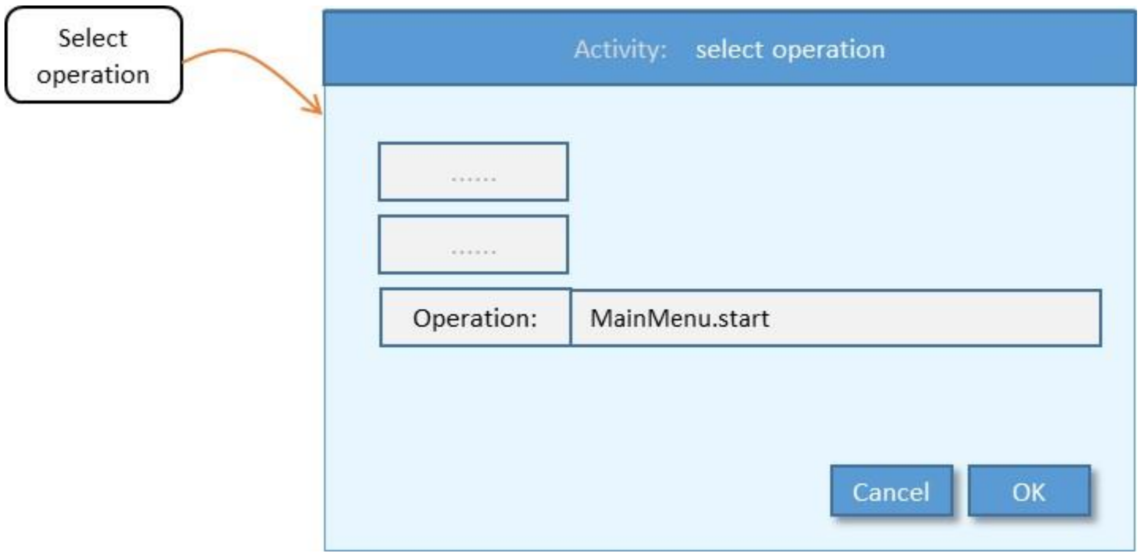


Figure 2.4 Component Model for the ATM Domain

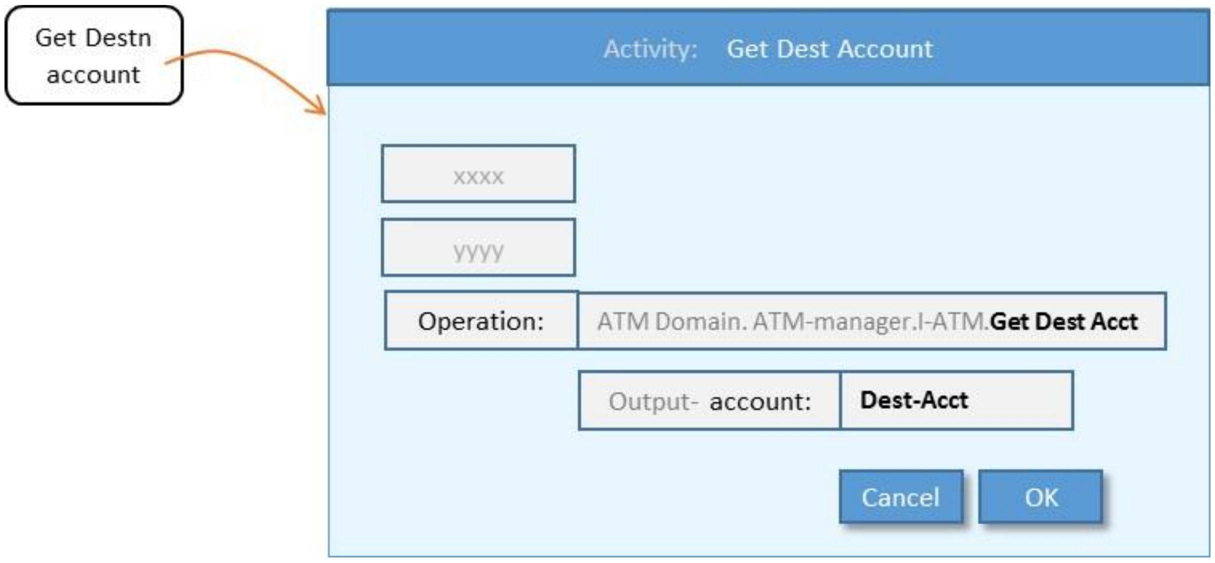


Operation: MainMenu.start

Figure 2.5 Connecting the first activity box with an executable function

Next, we will proceed with the activity in line, that is the ‘get destn account’. We will connect it to an operation. The operations are presented first with the component (or web serviece), then the corresponding interface, and finally the name of the method listed in that interface. Figure 2.6 displays the dialog box (screen) for this connection.

Before moving from the main menu to specific functions, an ssumption should be explained. All the menu items must be connected to some process activity in similar configuration windows. Here we assime that the ‘Deposit cash@ menu item is linked to the first activity corresponding to this system funciton, that is the “Get Destn account” in the process model (Figure 2.2).



Operation: ATM Domain. ATM-manager.I-ATM.**Get Dest Acct**

Output- account: **Dest-Acct**

Figure 26. Connecting the Get dest accnt with an operation

This screen connects the activity ‘Get Destn Account’ in the process model, to the operation ‘Get Dest Acct’ in the web service. Also, the variable ‘Dest-Acct’ is declared here as its name is written in the box to the right of the output parameter. The operation (Get Dest Acct) in the web service, returns a value in its return parameter named ‘account’. That value will be copied to the variable you declared (Dest-Acct). We are assuming that this operation will do whatever is necessary to provide us the account name or ID. For example, the account may be read from the screen during this operation, prompting the user to enter it. Or, it might be retrieved from the central database, may be as a set of available accounts that are associated with the card and user is prompted to select among them, etc. That detail is not implemented in this example.

After we got the account number, the process model suggests to get cash. This is done by delegating this activity box its duty to the accept cash function of the ATM hardware. Such delegation is our connection as shown in Figure 27.

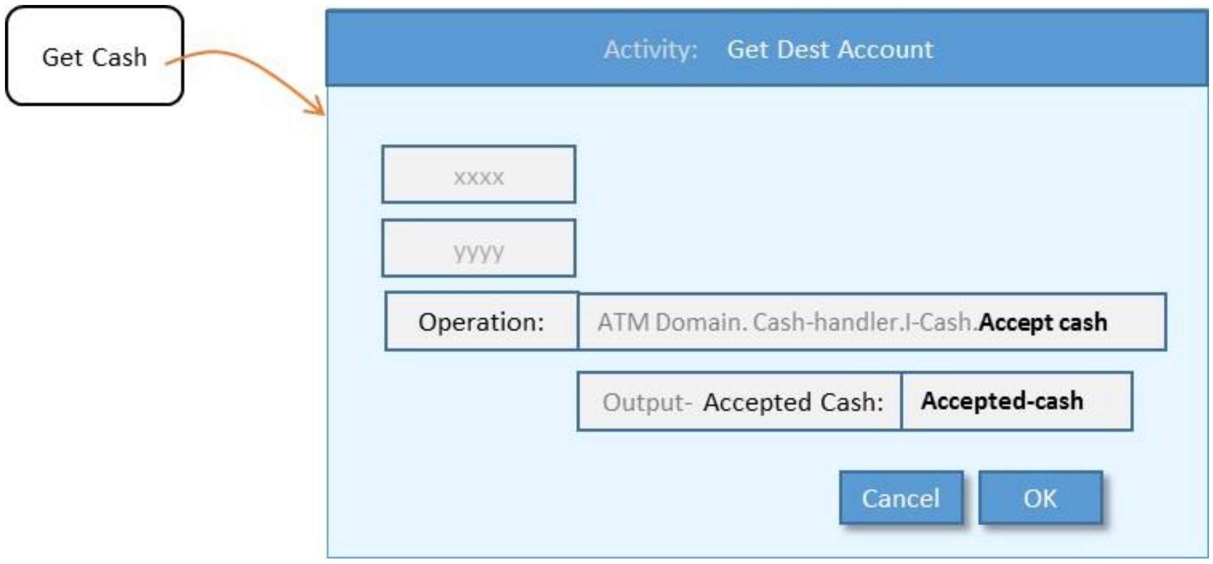

Get Cash
Activity: Get Dest Account
xxxx
yyyy
Operation: ATM Domain. Cash-handler.I-Cash.**Accept cash**
Output- Accepted Cash: **Accepted-cash**
Cancel
OK

Figure 26. Connecting the Get dest accnt with an operation

Next it would be suggested to call the update function in the central database, send it two parameters, the destination account and the accepted cash. We will not continue that far and stop here. That would also require the change in the process model (Figure 2.2).