Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Service Integration course

BPMN

Oszkár Semeráth Gábor Szárnyas

March 25, 2013

Contents

Chapter 1

BPMN

1.1 Introduction

BPMN (Business Process Model and Notation) is a widely used graphical representation for specifying business processes in a business process modell.

On the Service Integration course, we will use Bonita as our BPMN editor and workflow framework. Bonita is an Eclipse RCP application.

Figure 1.1: The logo of BonitaSoft

1.2 Sources

- http://www.bonitasoft.com/
- http://www.bpmn.org/

Chapter 2

BPMN laboratory – step-by-step instructions

In this laboratory, we will create the workflow of an application store. In the application store the users can browse and upload applications. On the Model Driven Software Development and Service Integration Courses In 2012, the teams had to design and implement the workflow of an application store.

2.1 Simple workflow

1. Start **Bonita Studio**. Bonita will prompt you to register. You can choose to skip it but it's highly recommended to register because registration provides access to well-made official tutorials and thorough documentation.

Figure 2.1: The opening screen of Bonita

2. Create a new process from **Process | New**.

3. A simple process will show with only a *start event* and a *human task*. Click on the process, choose the **Pool** page and click the **Edit...*** button. Rename the process to `Browse Application`.

4. Click the `Step1` task and look at it's properties on the **General** tab. On this tab, you can set the execution-specific properties of the process, e.g. it's **Name** and **Task type**. Rename the task to `Acknowledge`.

5. Add an *end event* to the workflow. Connect the `Acknowledge` to the end event.

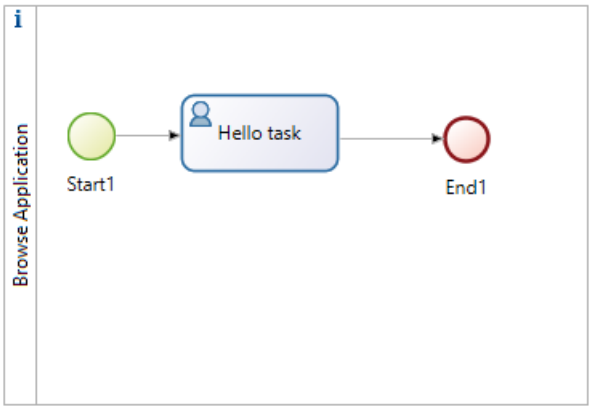

Figure 2.2: The `Browse Application` process

6. Let's create a graphical user interface for this task. Choose the **Application** tab. On the **Entry Pageflow** page click **Add...**. Click **Finish**.

7. A graphical editor will appear. Add a **message** to the top of the form. Edit the properties of the message element on the **Data** page. You can edit plain text or HTML code. Type `Hello world!`.

8. Click the **Run** button or choose your process in the **Run | Run** menu. The generated web page will show in a browser.

9. On the web interface, you can control the workflow by the buttons provided. In this example, if you click the **Submit1** button, the workflow finishes.

10. Click to the UserXP button and browse this interface. Try to start a new workflow from this.

11. Create the following tasks:

    - `Download the application names`: *abstract task*.
    - `Show the applications`: *human task*.
    - `Buy the application`: *abstract task*.

12. The `Show the applications` human task is not associated with an actor. Click the task and choose the **Actors** page. Next to the **Actor Selectors** box, click the **Choose...** button and select the **Initiator**. The **Initiator** is the role of the person who starts the task.

13. Let's add some workflow variables to the process. Click the process and choose the **Data** page. Create the following variables:

    - `Applications`: is the collection of names of the downloadable applications. The type of this variable is `Text` and the multiplicity is `Multiple`.
    - `SelectedApplication`: The user will select one of the available application. This `Text` variable with `Single` multiplicity contains the name of it.

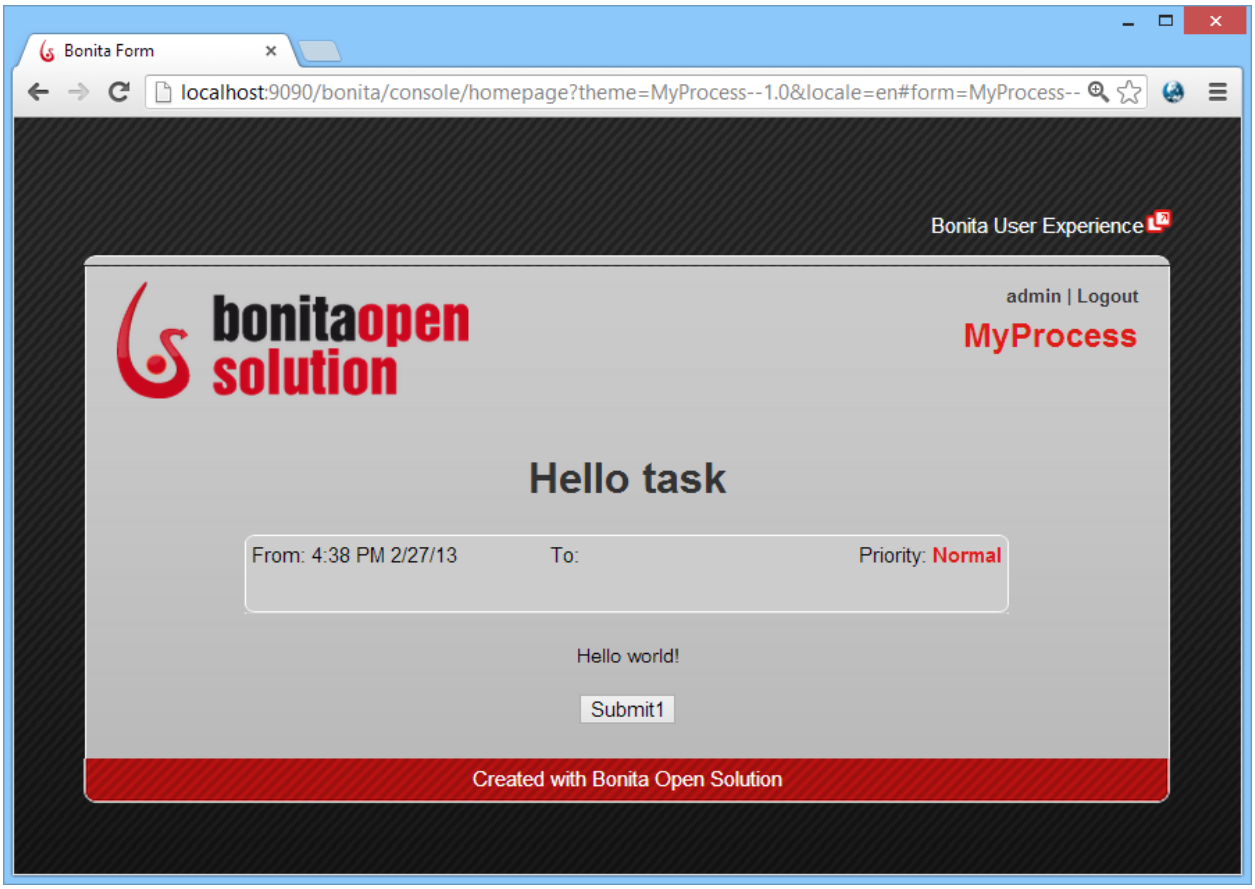

Figure 2.3: The `Hello task` in the browser

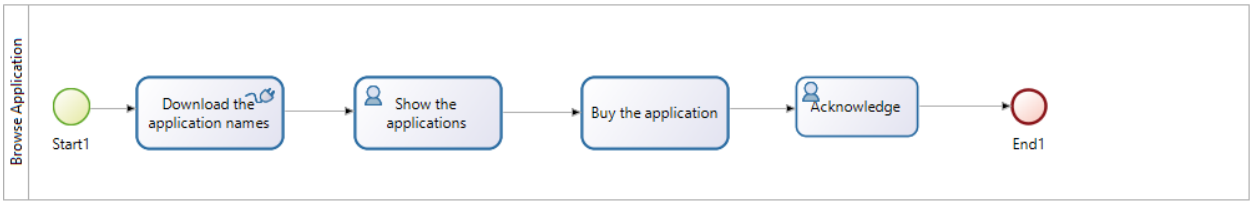

Figure 2.4: The `Browse Application` process

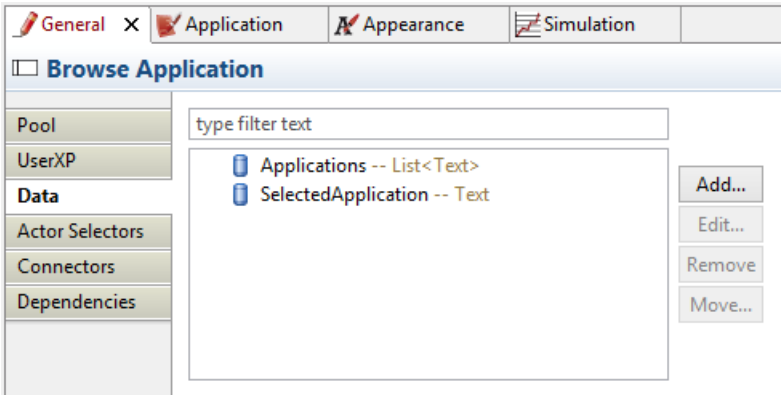

Figure 2.5: The variables of the `Browse Application` process

14. Let's create a script that substitutes the calling of other services. Select the **Download the application names** task and go to the **Connectors** page. Add a new script by selecting **Scripting | Groovy – Execute a Groovy script**.

15. Name the script instance to **Get the applications**, time it to the enter of the activity and hit next. At this window select the **Edit expression...** option from the combobox, and a conviniant Groovy editor will apear to write our script in it. This allows us to edit a Java-like expression or a method body where every flow variable is available.

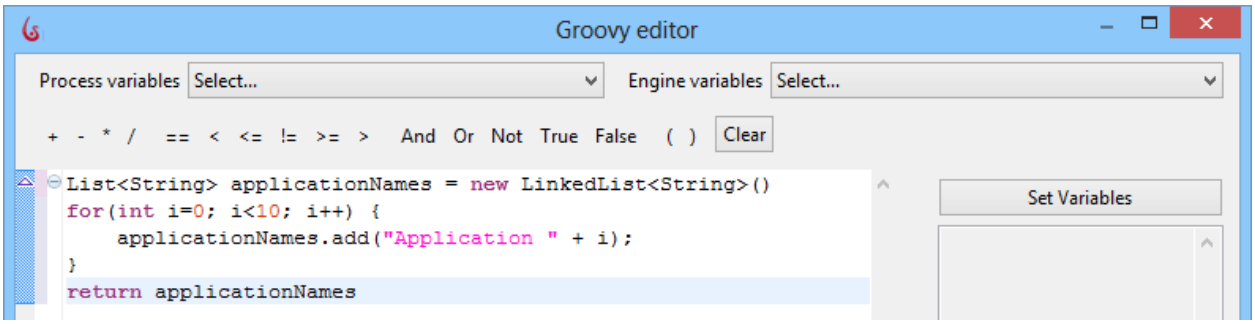

Figure 2.6: The Groovy editor

16. Create this script that returns a collection of application name:

```
List<String> applicationNames = new LinkedList<String>()
for (int i = 0; i < 10; i++) {
  applicationNames.add("Application " + i)
}
return applicationNames
```

    Click next and direct the result of this script to the `applications` variable and choose the **REPLACE** strategy.

17. Add a graphical view to the **Show the applications** task. There will be a section named **All widget based on...** that automatically derives the view from the checked flow variables. In this case we want to specify every element so unselect all.

    Drag a **Radio buttons** widget to the top of the view ad go to the data page of the property view. We would like to show the application names in this list, so got to the **Avaible values** and select the **${applications}...**

option from the right side. As you select it a window should appear where you pick the “The whole list of values” option.

We also want to put the name of the selected value to a variable, so edit that the `${field_Radio_buttons1}` saves to **selectedApplication**.

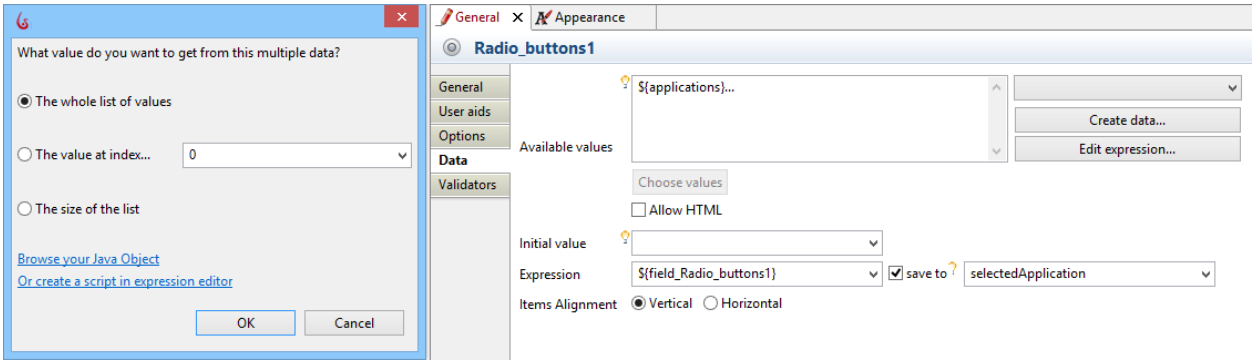

Figure 2.7: The final properties of the radiobuttons.

18. The message of the **Acknowledge** action should refer to the selected variable, so let’s set it. If you closed the editor page go to the properties of the action select the **Application** page and edit the **Hello task** pageflow.

    At the **Data** page `Hello world!` message with an expression by the **Edit expression...** option and write that:

    ```
    "Thank you for downloading the " + selectedApplication + " application."
    ```

19. Try to run the application. Don’t be afraid of the presettable variable at the begining.

20. Sone time an action may fail and the error should be handled. Add a script connector **Buy the application** action. Select the **Throw error event** at the **If connector fails...** options and name the error ot “failed”. The script should looks like this:

    ```
    if(selectedApplication == applications.get(0))
      throw new UnsupportedOperationException()
    ```

    This script will fail with an exception if the user downloads the first application. The output should be neglected.

21. Add a **Catch error** item to the **Buy the application** action from the palette. Create a human task for the initiator and edit the control flow:

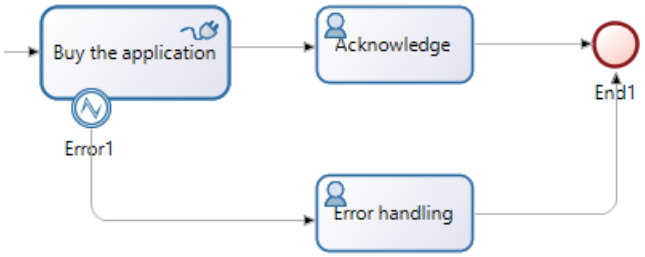

Figure 2.8: Error handling flow

Create a webpage for the task where there is a message that shows “Error in the web services!”.

22. Run the workflow and select the first application. It looks like that the workflow stops but eventually the next action arrives to the inbox.

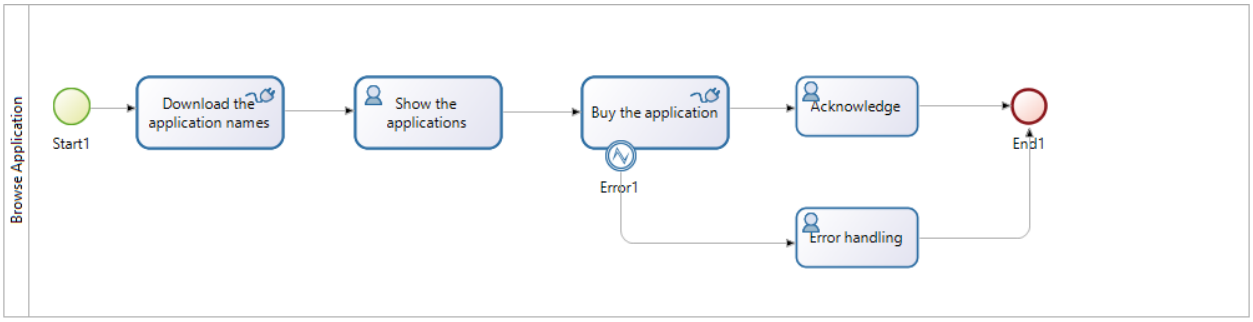

Figure 2.9: The final process with exception handling

2.2 Complex workflow

We will implement a workflow for the actions of the user.

1. Create a new *pool* and name it to `User workflow`.
2. Create a *start event*, an *end event* and create the following tasks:

   - `Authenticate`: *service task*
   - `User action`: *human task*
   - `Login failed`: *human task*
   - `Browse applications`: *call activity*
   - `Upload application`: *abstract task*
   - `Logout`: *human task*

   For the *human tasks*, set the actor to **Initiator**.
3. Create a **XOR gateway**.
4. Time to create some variables:

   - `User ID`: `Integer`
   - `Username`: `Text`
   - `Password`: `Text`
5. Also create a new variable named `Action`. To create an enumeration, set the **Data type** combobox to **List of options...**. Set the **Name** to `UserActionType` and add the following options:

   - `Browse`
   - `Upload`
   - `Logout`

   Click **OK** and **Finish**.
6. To create the login screen, click on the `User workflow` *process*. On the **Application tab**'s **Entry Pageflow** page add a new form named `Login`.

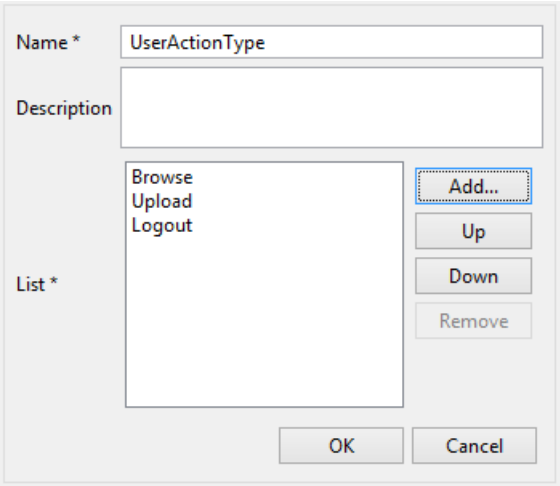

Figure 2.10: The `UserActionType`

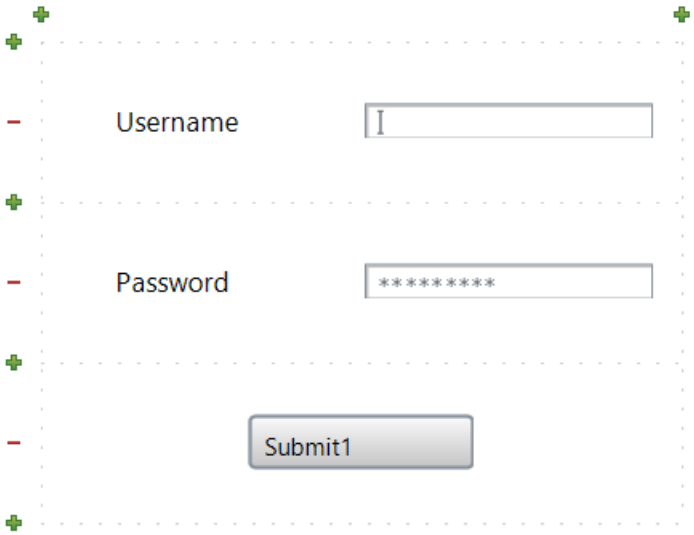

Figure 2.11: The login screen

7. In the **Add widgets based on...** groupbox only select the `username` and `password` widgets.

8. Set the password field's **Field type** to **Password**.

9. Connect the *start event* to the `Authenticate` task. This is a service task which simulates the authentication of the user. Add a new **Groovy** connector named `Simulation of Authentication`.

```
if (username == password) {
  return username.hashCode();
} else {
  return null;
}
```

   The `result` from connector's output goes to the `user_ID` variable. Click **Finish**.

10. Depending on the authentication's result, the user can proceed or fail the login. Create transitions from the `Authenticate` task to the `User action` task named `success` and the `Login failed` task name `fail`.

11. On the `success` transition choose **Edit expression...** in the **Condition** combobox and type `user_ID != null`. If this condition is not satisfied, the login fails. To implement this, tick the **Default flow** checkbox for the `fail` transition.

12. Add a form to the `User action` task. Only select the `action` widget, which will be mapped to radio buttons.

13. Now we have to create the conditions to the transitions from the XOR gateway. To do this, click on the transition and from the **Condition** combobox choose the **Edit expression...**.

    - For the transition to the `Browse applications` task, set the expression to `action == UserActionType.Browse`. Pay attention to import the enumeration – the Groovy editor is basically and Eclipse editor, so you use the content assist (`Ctrl+Space`) to do so.
    - For the transition to the `Upload application` task, set the expression to `action == UserActionType.Upload`.
    - For the transition to the `Logout` task, tick the **Default flow** checkbox.

14. The user can browse and upload applications multiple times. To implement this in the process, we have to create a loop. In order to do so, create an *abstract task* named `End of repeatable user action` and create the necessary transitions (see the figure).

15. For the `Browse applications` task set the **Subprocess Name** to `Browse_Application`.

16. Create a form for the `Logout` task. Add a message saying `The following user has logged out: ${username}`.

17. Create a form for the `Login failed` task. Add a message saying `Login failed`.

18. From the `Login failed` and the `Logout` tasks draw a transition to the *end event*.

2.3 Tips

- If you close some windows by mistakes, you can make them reappear by choosing **View | Reset view**.
- If you name a transition and then delete the name, Bonita will mark it as faulty with the following message: **Empty name detected for a SequenceFlow**. The solution is to name the transition.
- Sometimes, the error markings don't disappear until you manually validate the workflow by clicking **Process | Validate**.

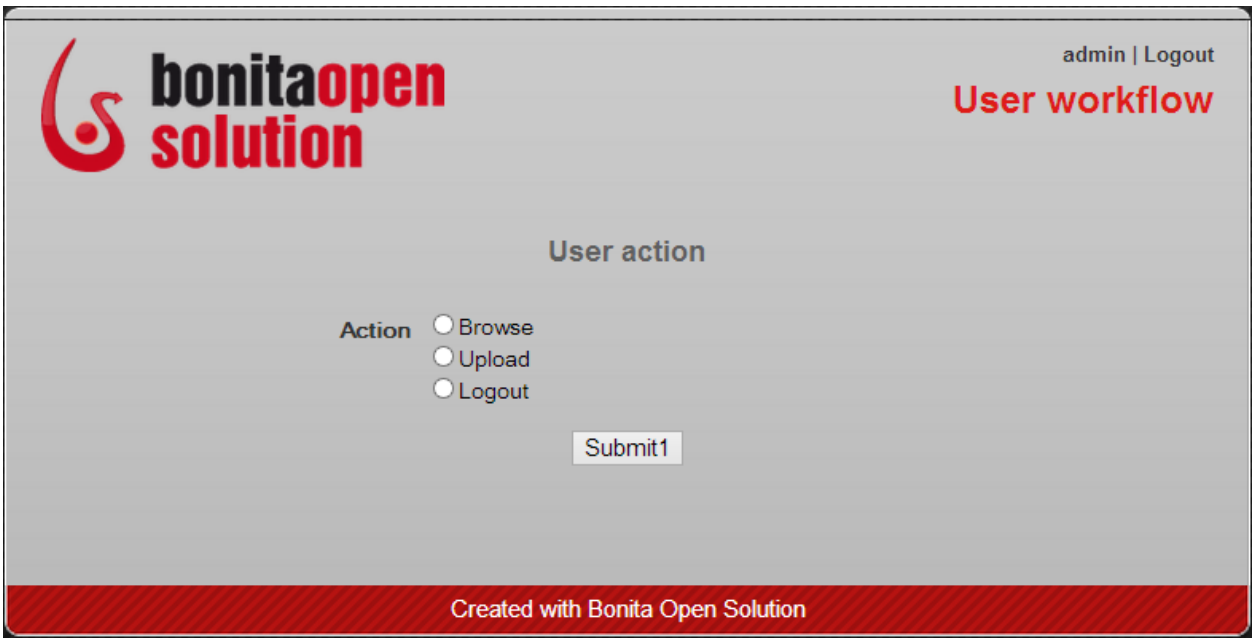

Figure 2.12: The `user action` form

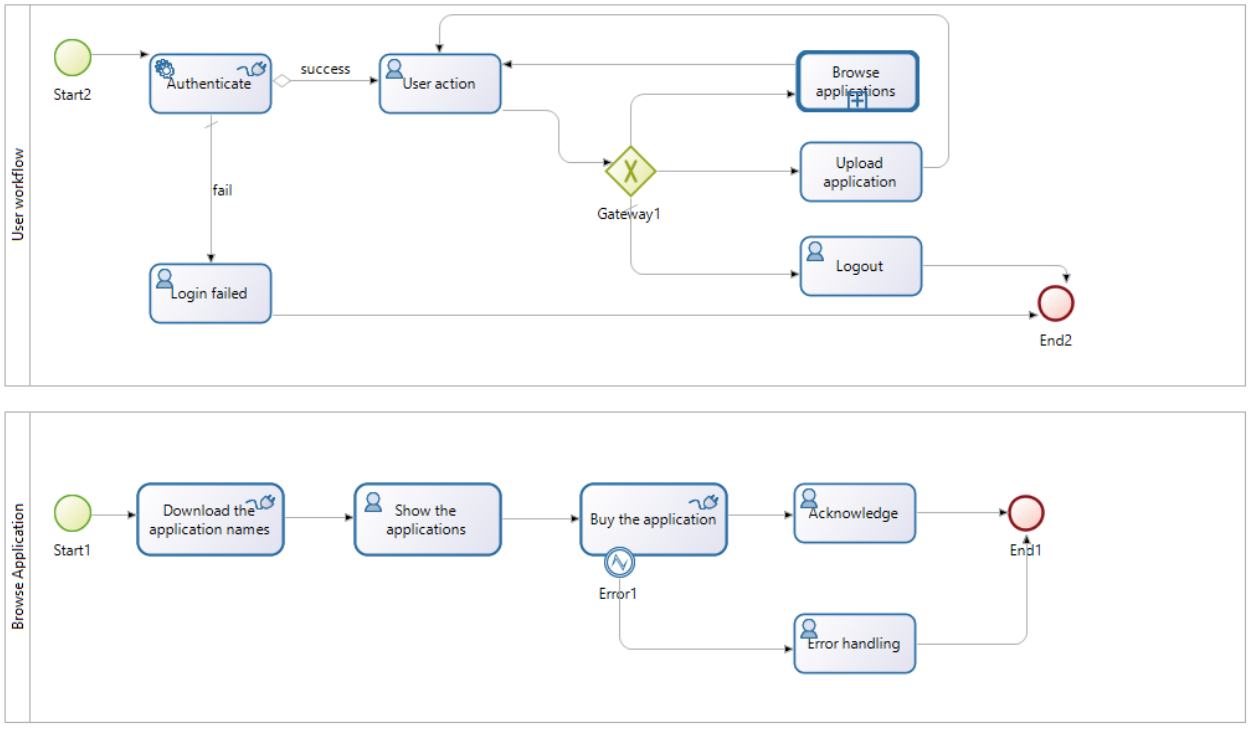

Figure 2.13: The final process