Executable Process Models

Going the Last Mile with Bonita BPM

(Community Edition, Version 7.9)

Joerg Evermann
Memorial University of Newfoundland
July 2019

Please report errors and omissions, send corrections, or offer ideas to
jevermann@mun.ca

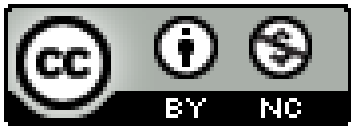


Table of Contents

Introduction

Bonita BPM is an open-source workflow management system. The community edition is free to download and contains both a modelling component (based on Eclipse), called Bonita Studio, and the actual workflow management system, containing the workflow engine, a built-in database for persisting case information and workflow information, a resource service that manages organizational groups, roles and members, and other services and interfaces for external services. The workflow engine is built on a Tomcat web server. The web interface to this component is called the Bonita Portal. Bonita BPM does not offer the full range of BPMN 2.0 symbols and also has some limitations in the way that symbols may be used.

This tutorial will guide you through the entire process of developing a BPMN based workflow application on Bonita. The tutorial is developed using the community edition, Version 7.9. It is assumed that you have already downloaded and installed the software. This tutorial is based on exercise 10.10 in the book "Fundamentals of Business Process Management" by Marlon Dumas, Marcello La Rosa, Jan Mendling and Hajo A. Rijers published by Springer (hereafter called "Fundamentals").

Bonita BPM

Bonita BPM uses a number of concepts and technologies that go beyond those introduced in "Fundamentals". This section will introduce the main ones in order to prepare the reader for the remainder of the tutorial.

Relational database

Bonita BPM uses a relational database to store case information. Throughout the tutorial, we will make use of relational data definitions, show how to access the relational database, and use SQL queries to interact with case data from process activities. While not essential, an understanding of relational databases and SQL enhances reader understanding.

Java & Groovy

The runtime component of Bonita BPM is written in Java and makes heavy use of Groovy, a "variant" of the Java language for scripting, value assignments, conditional expressions, etc. The persistent case data in the relational database is encapsulated in a Java-based persistence layer to run-time access. Run-time case data is accessed by process activities through Java objects and their methods. While not essential, an understanding of Java/Groovy enhances reader understanding.

Web-based User Interface

Bonita BPM uses web-based user interfaces to present user activities with case data, case instantiation forms, and case overview forms to the user. User interfaces are not extensively covered in "Fundamentals" as this is where WfMS differ most. In Bonita BPM, every user activity has an input contract that specifies the case data that is used by the activity. Every user activity has a user interface form associated with it. Bonita BPM provides a web-based interface designer that can create forms automatically from the contract. The form shows JavaScript variables. These are initialized from case data when the form is shown, and return to Bonita BPM when the form is submitted. However, these

data are not automatically written back to case data upon activity completion. While not essential, an understanding of JavaScript and CSS enhances reader understanding.

Operations

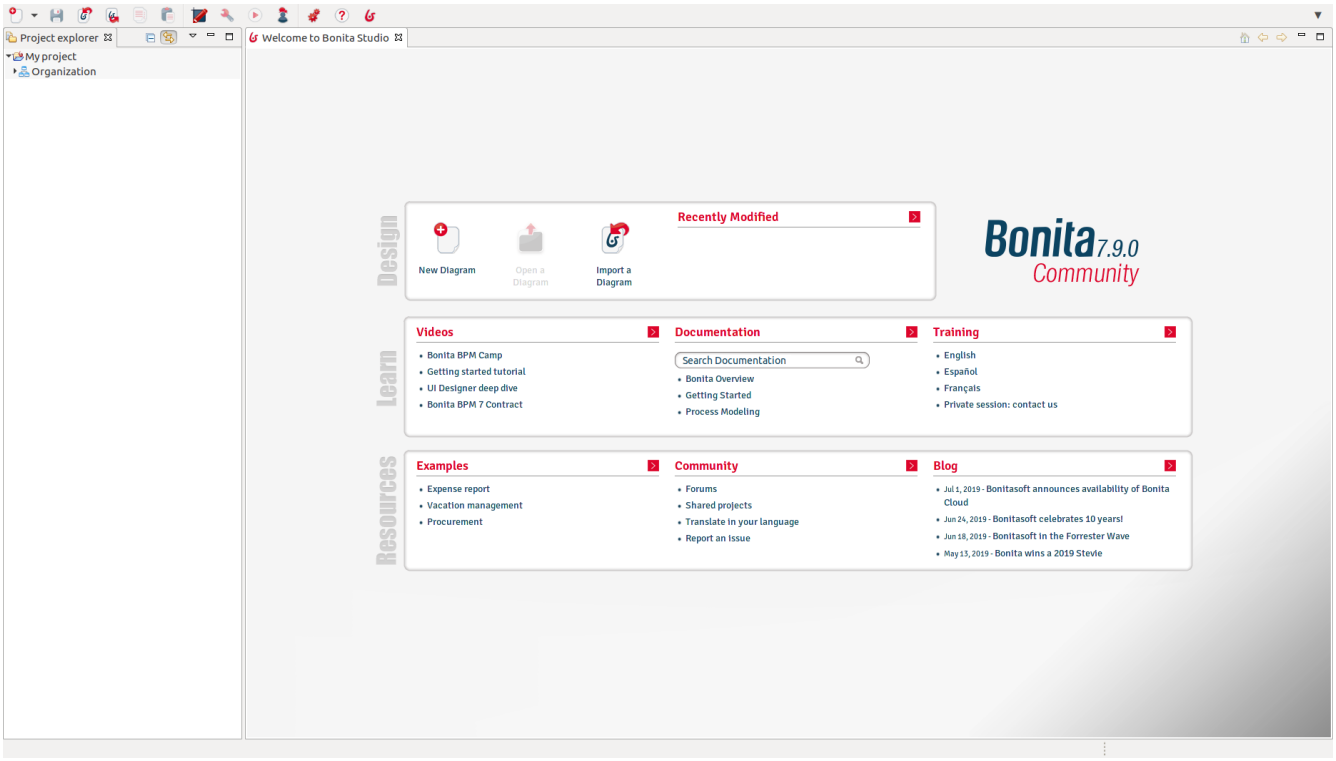
All types of activities in Bonita BPM can have operations. Operations are used to instantiate case data objects, delete case data objects, or assign values to attributes of case data objects. The expressions to calculate values for attributes can be based on SQL queries to the database, use methods of existing Java case data objects, or use custom Groovy scripts that have access to all case data objects. Operations are used to write back form inputs to case data objects after user activities complete, but they can be used in other tasks as well, for example, script tasks, to evaluate a custom Groovy script. When creating contracts for user activities, Bonita BPM can automatically create the operations necessary to write back form information to case data.

Pools

Pools in Bonita BPM represent processes, not business parties, as recommended in “Fundamentals”. As such, the focus on modelling pools in Bonita is on data management and instantiation of process instances. Case data is specified for each pool and is shared across all lanes in that pool. Pools are instantiated to create workflow instances (cases) when a start event is reached. At that point, case data is initialized, using custom initialization scripts, typically in Groovy, to prepare case data.

Getting Started

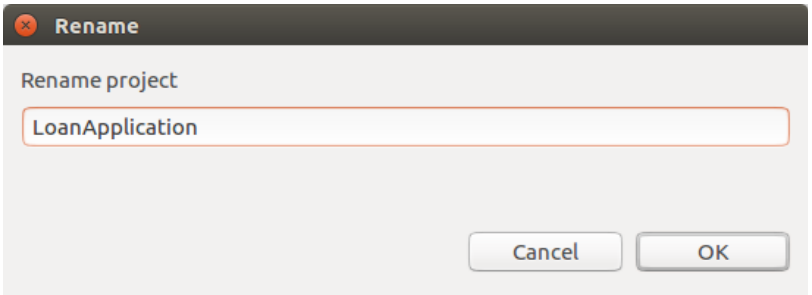
When Bonita is first launched, it presents the Bonita Studio with a welcome tab and a new project name “My Project”.



Close the welcome tab. Bonita community edition allows only a single project in the workspace. Rename the project:

- Right-click on the name “My Project” and selecting “Rename...”.
- Enter the new name for the project
- Press “OK”.

Renaming the project may take a few moments.



Data Management

Begin by modelling the data perspective of the process. Bonita has a rich data perspective that is based on the relational data model to describe *business objects*. Bonita provides a built-in relational database (based on H2[1]) and a modelling tool that makes it easy to define the business objects.

This tutorial is based on the data description in exercise 10.9 in "Fundamentals". To make the modelling easier, the tutorial simplifies the data perspective somewhat and describe the following business objects that are also found in Fig. 10.12 in "Fundamentals"

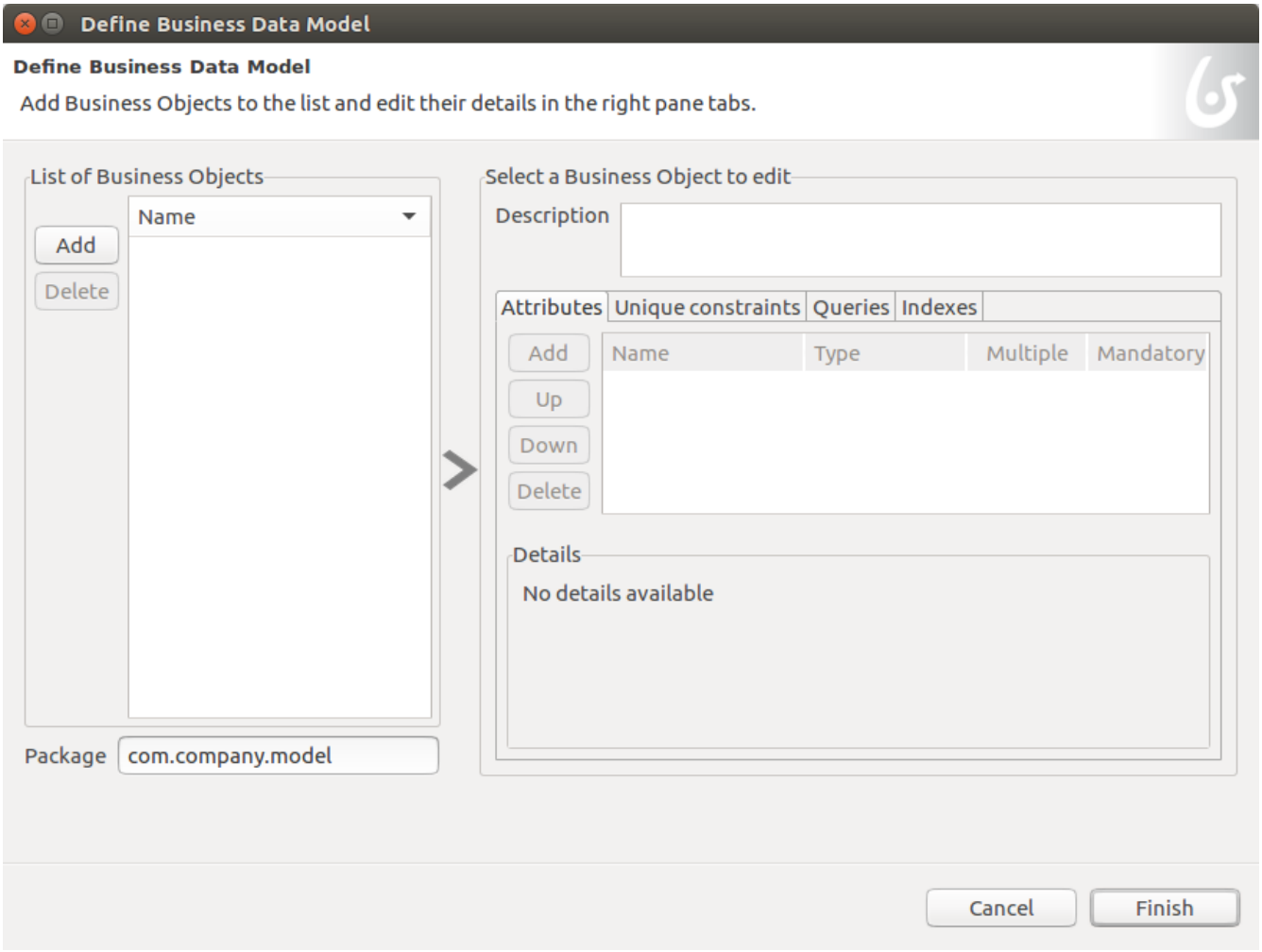
- LoanApplication
- Applicant
- Property
- CreditReport
- RiskAssessment
- PropertyAppraisal

Begin the data modelling

- Select "Development" → "Business Data Model" → "Define ..." from the menu.

The following dialog lets you define the different business objects.

1 https://h2database.com/html/main.html

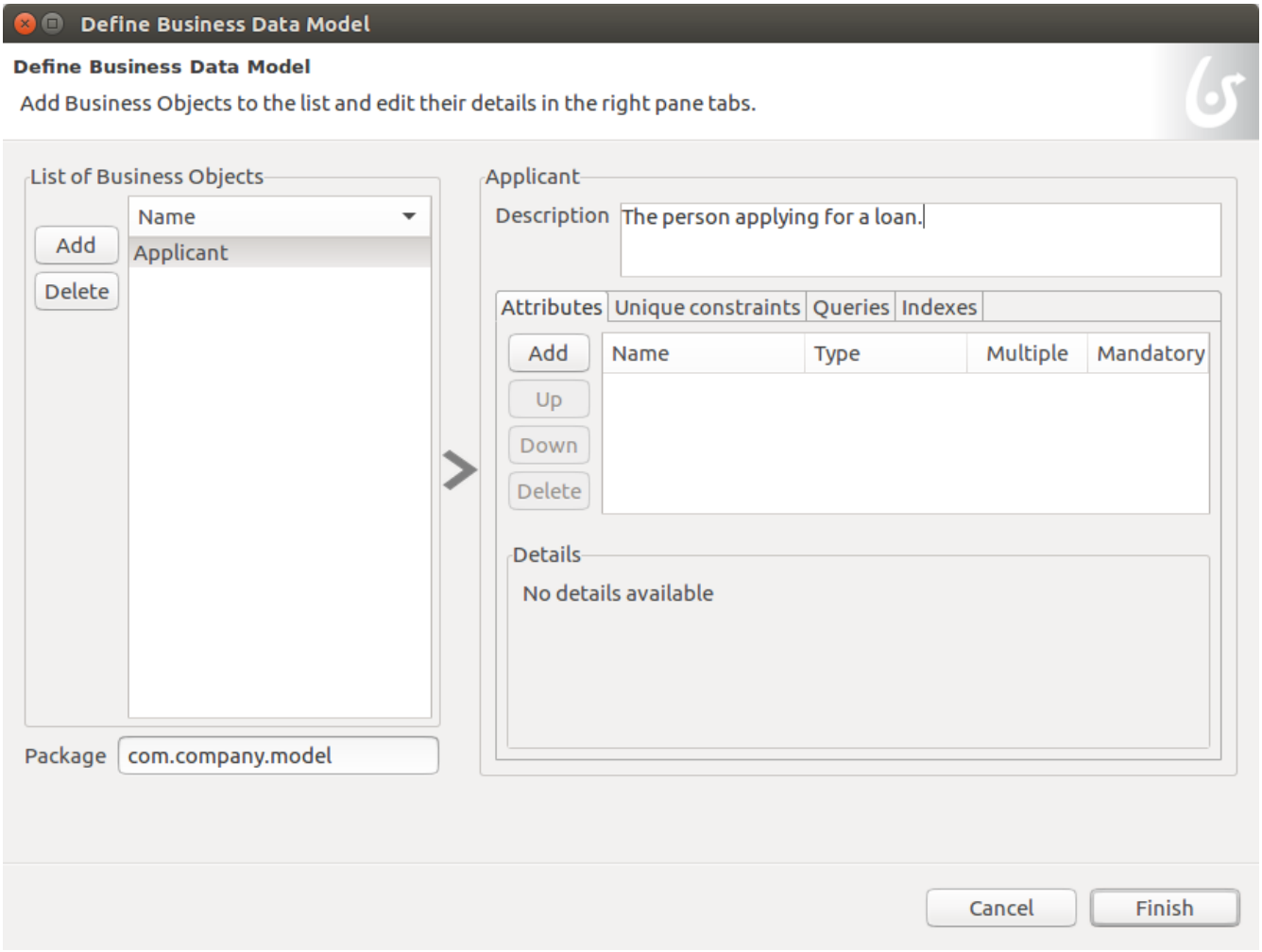


Because Bonita will automatically create Java class definitions for the persisting and retrieving the database contents, it prompts you to define a Java package name for these classes, which defaults to "com.company.model". You do not need to change this.

- Click the "Add" button on the left to create a new business object.

The name of the business object defaults to "BusinessObject" which you can change by selecting the name in the list of business objects and overwriting it.

- Name you first business object "Applicant".



Next, define attributes for the applicant business object. For each attribute,

- Press the "Add" button next to the attribute list.

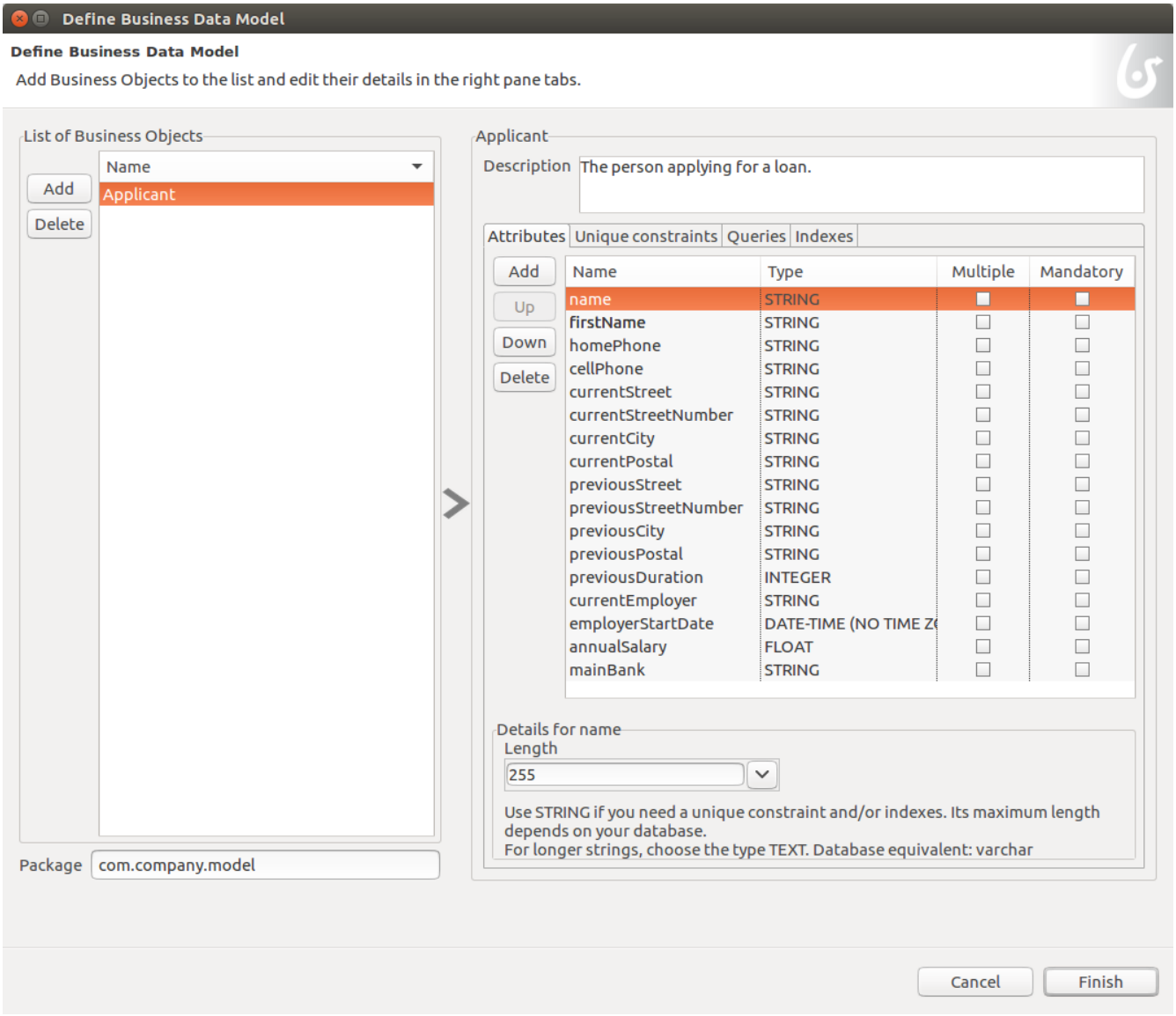
This creates a new attribute, by default the names are numbered consecutively and the data types of attributes are character strings of length 255.

- Click on the attribute name to change it and select an appropriate data type for the attribute.

The following image (next page) shows an example for the applicant business object.

Bonita will create an artificial primary key for each business object called *PersistenceID* (a long data type) and allows versioning of the data definition using the *PersistenceVersion* attribute, which is also automatically created.

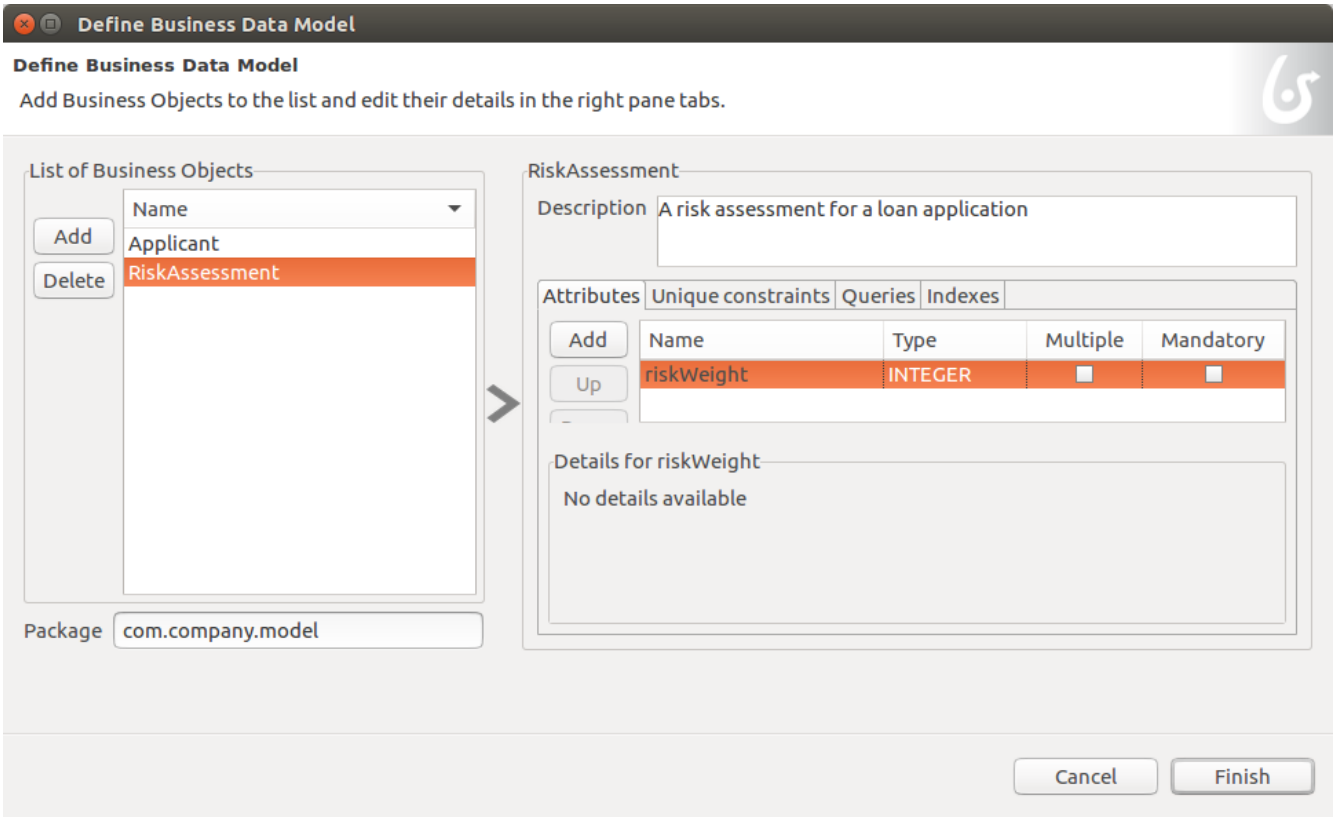
To keep things simple, do not normalize the data schema (for example, by defining an Address business object and linking this as current and previous address). Also do not use multiple or mandatory attributes, unique constraints, pre-defined queries and indexes for query performance improvements. These are advanced data management topics beyond the scope of a process-centered tutorial.



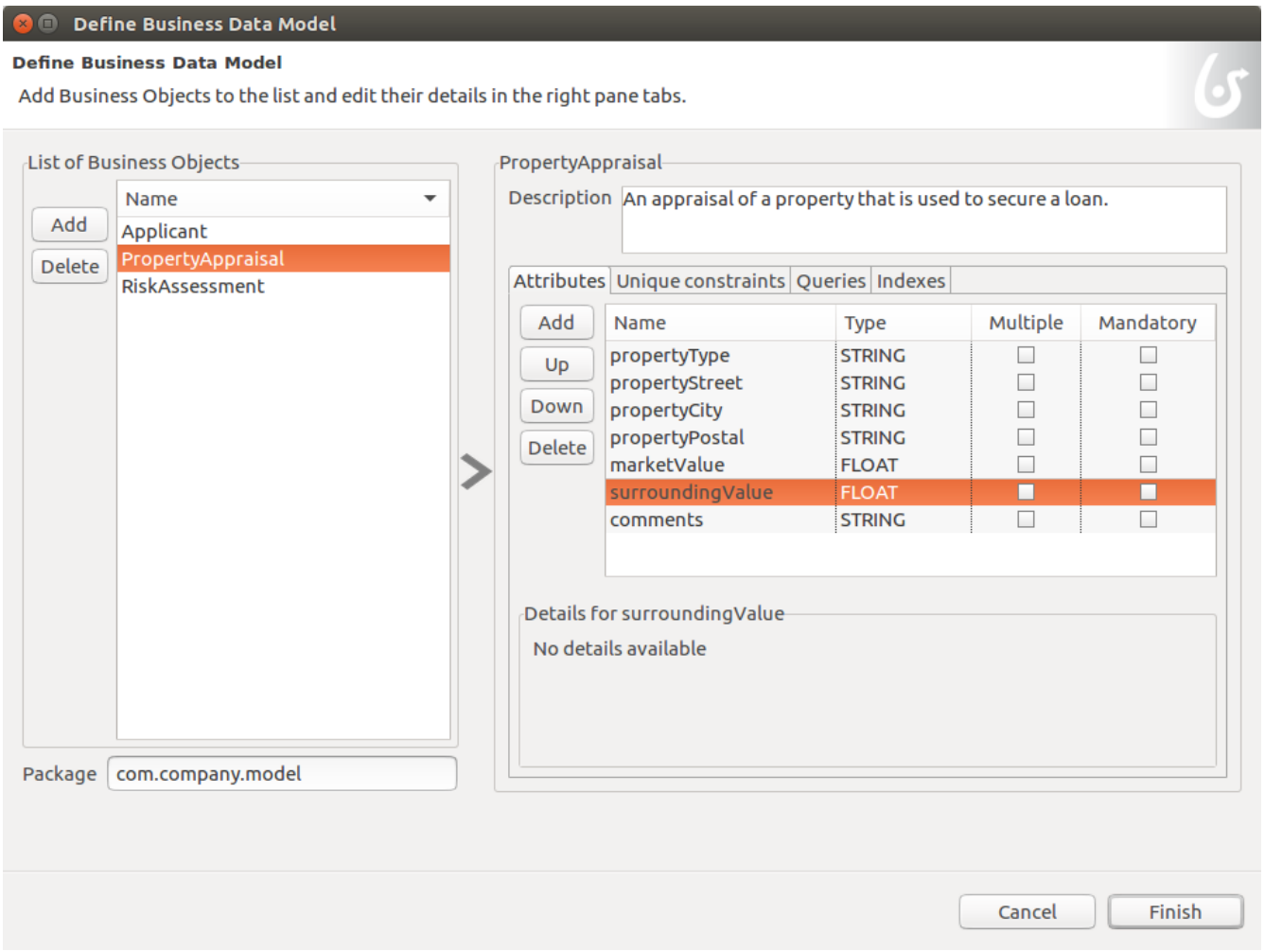
Exercise 10.9 in “Fundamentals” offers some ideas on attributes for the different business objects.

- Define a business object with attributes for each of the business objects listed above.

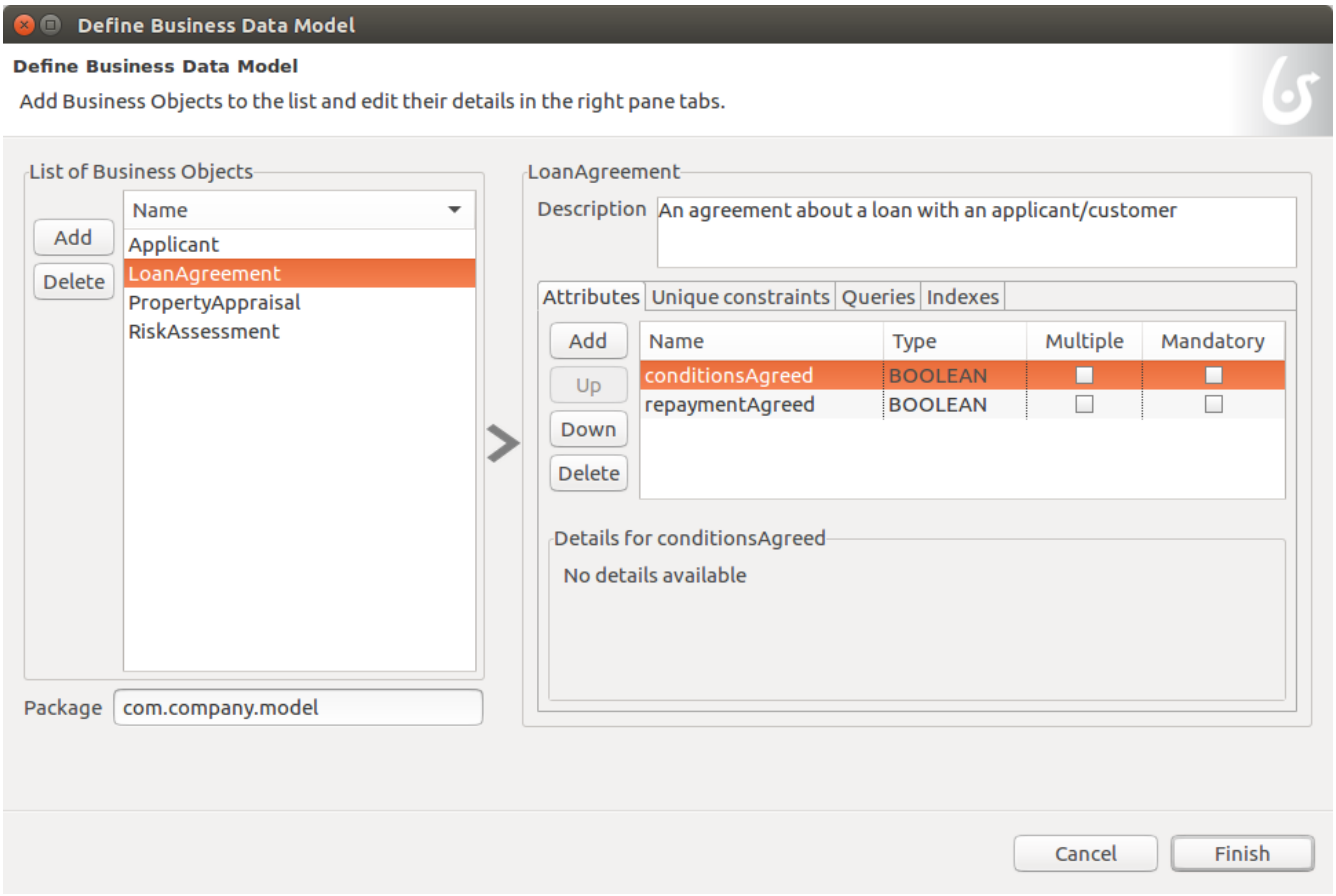
The following image shows a definition of the risk assessment business object. Of the attributes listed in exercise 10.9 in “Fundamentals”, the assessment identifier is not required as Bonita will create an artificial primary key for each business object. Also, the attributes to reference the loan application and credit history report are not required; below, you will make the link from the loan application to the risk assessment, credit history report and other objects.



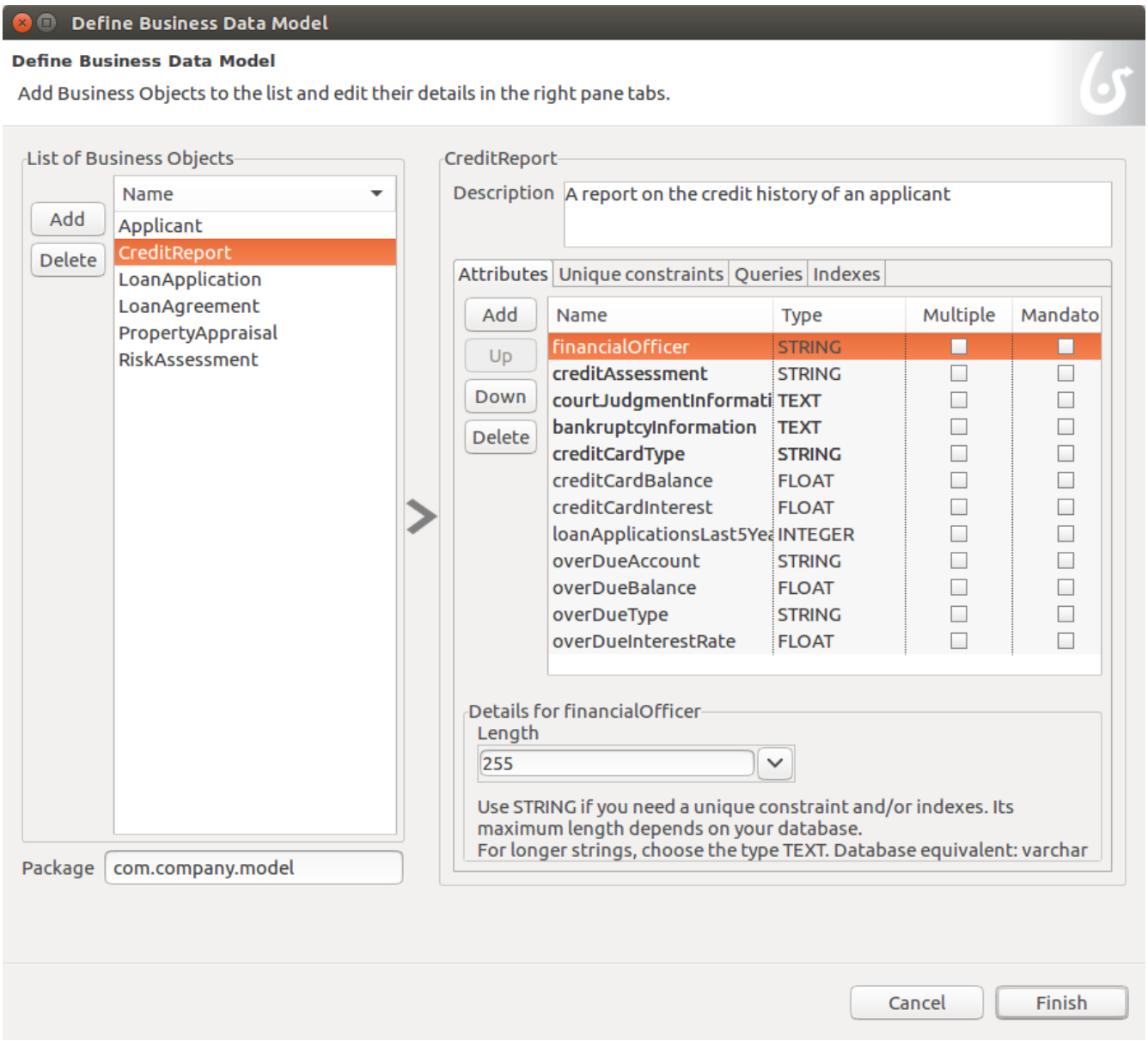
The following image shows the definition of the property appraisal business object. Again, omit identifiers and references to other business objects.



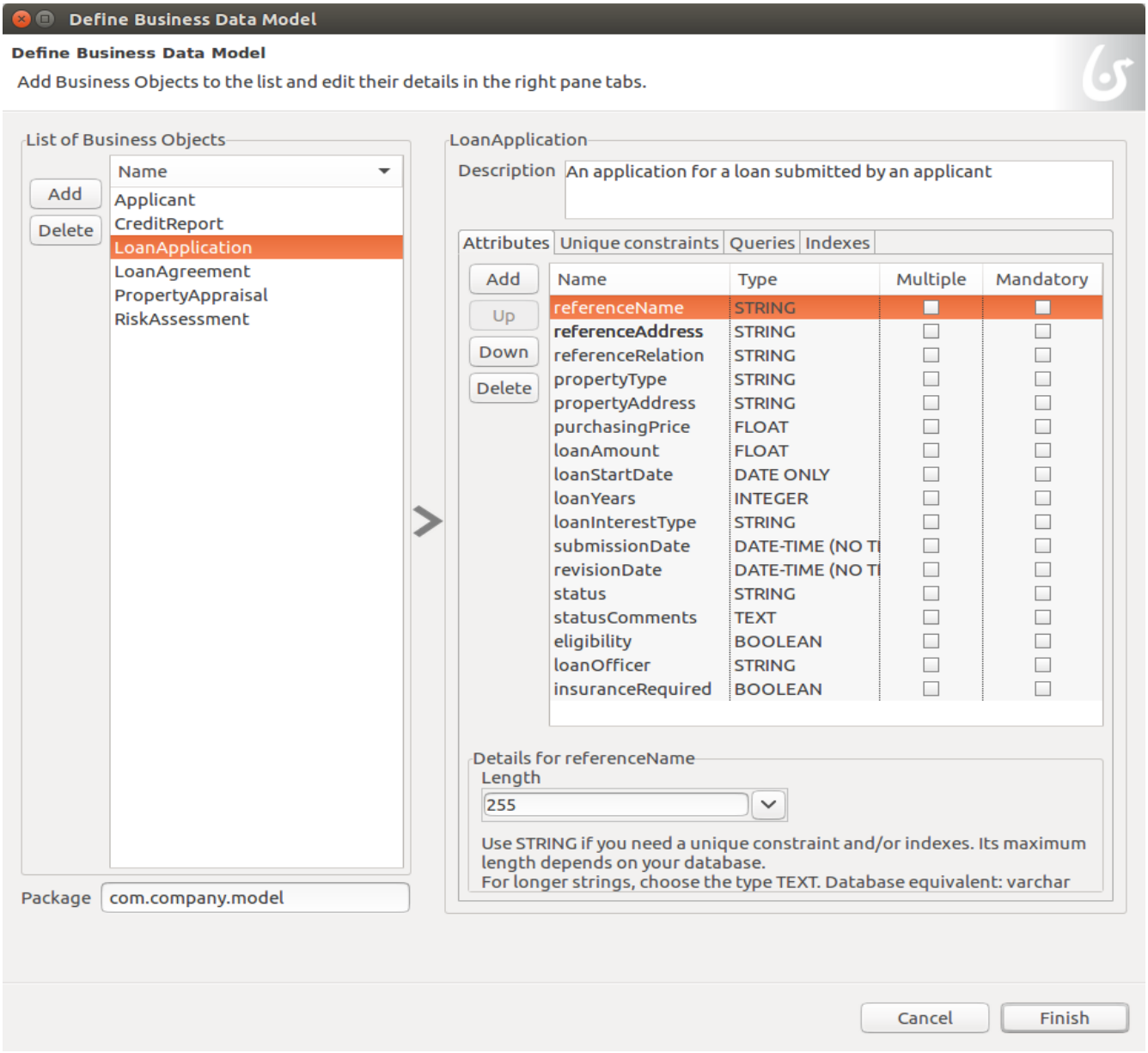
The following image shows the definition of the loan agreement. To simplify the tutorial, omit the digitized copy of the repayment agreement from exercise 10.9 in “Fundamentals”. Bonita provides a way to manage documents that is separate from the business object data perspective. Alternatively, a URL to an external file store could be represented by an appropriate attribute.



The next image shows the definition of a credit report. This is simplified significantly by omitting normalization of entities such as credit account, credit cards, and by omitting multiple entries of these.

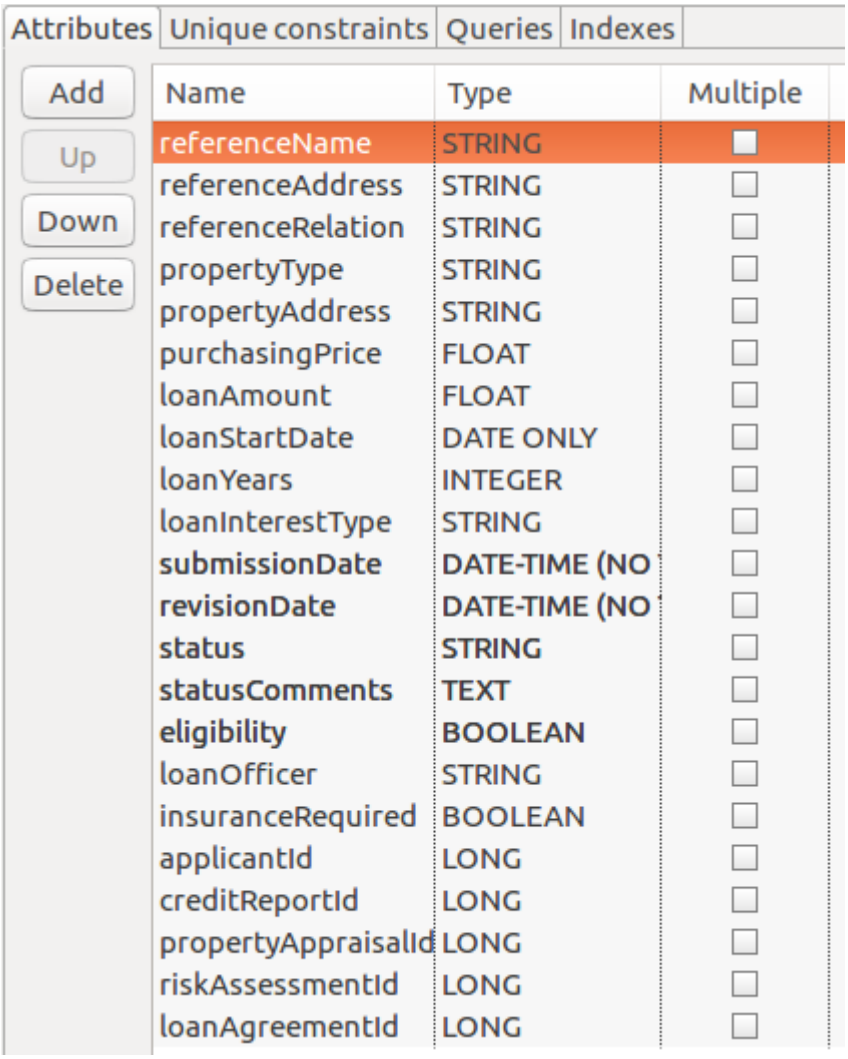


Finally, the following image shows our definition of the loan application. This is also simplified, for example, by omitting detailed address information for the reference and the property.



You will now implement the connections that tie the various business objects to a particular loan application. You will do this by noting the persistence Id of the applicant, the credit history report, the property appraisal, the risk assessment and the repayment agreement summary in the loan application.

For this, you will extend the business data model of the loan application business objects by five attributes of type long, to hold the persistence Ids of each of the associated business objects. Note the last five attributes of the loan application business object in the following figure (next page).

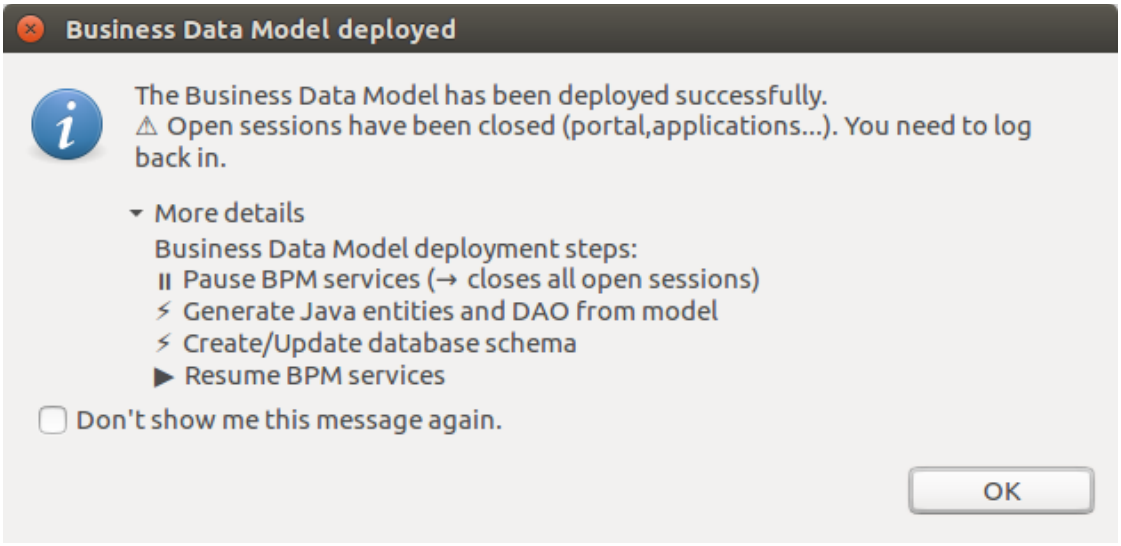


These attributes should never be visible to the user, they should not appear in any user interface form or activity input contract. However, their value will be set whenever a process activity creates a new association between business objects.

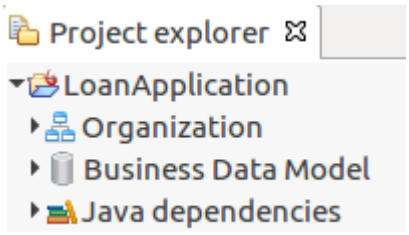
- Create the additional attributes for the loan application business objects as shown in the figure.
- Press the “Finish” button.

Bonita Studio now deploys the business data model to the Bonita workflow server. This creates tables in the relational database system as well as the Java classes for retrieving and persisting the information (DAO, “data access objects”). If you have previously deployed the business data model and have made changes to attribute data types, Bonita may ask if you want to clear the database. This is highly recommended.

Bonita will report when this process is finished:



Additionally, the business data model and Java dependencies are added to the project in the Bonita Studio project:

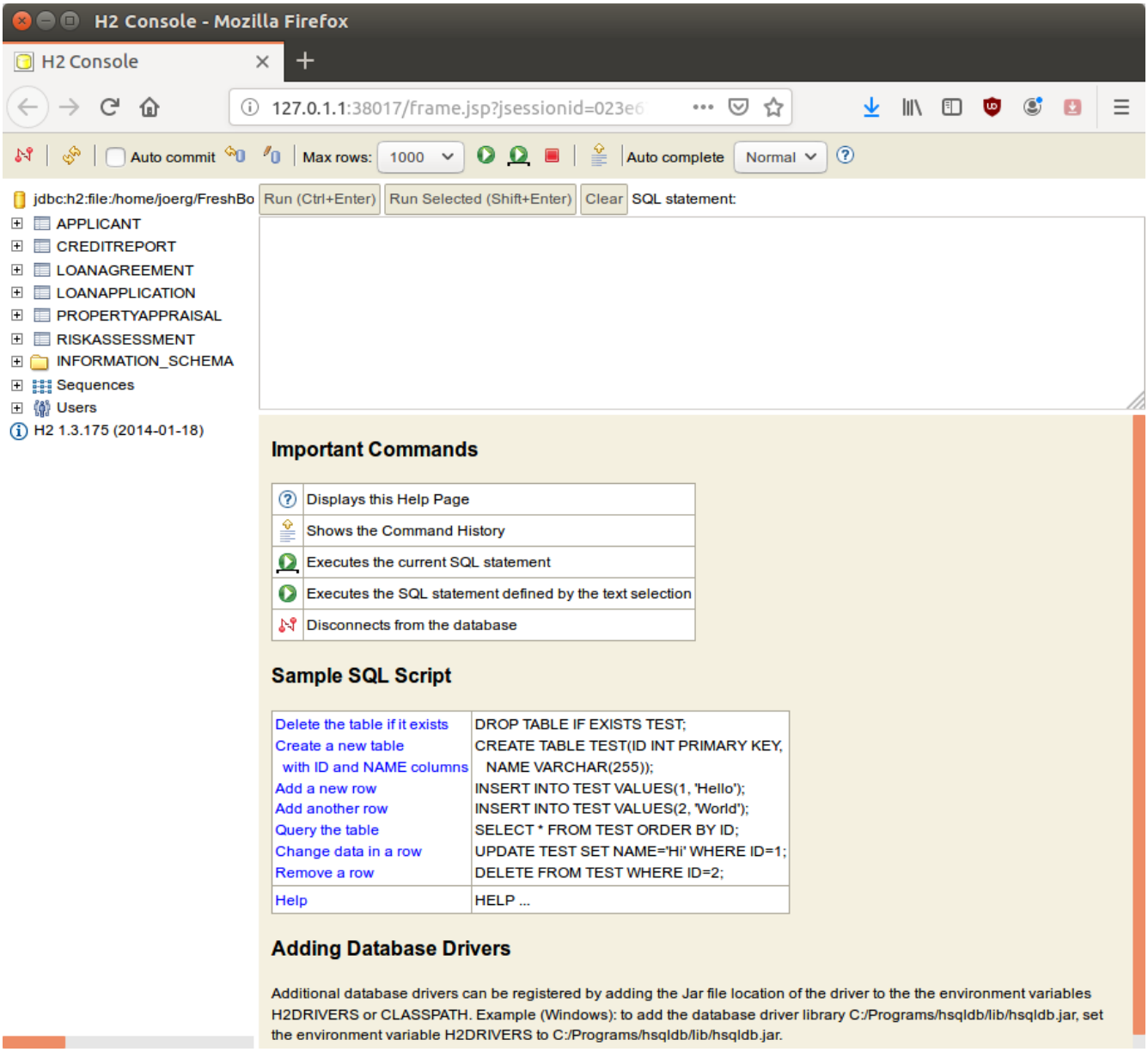


Database Access

At any time, you can gain access to the database and query its contents:

- Select “Development” → “Business Data Model” → “Browse Data (h2 console) ...” from the Bonita Studio menu.

This opens a page in your web browser that lets you interact with the H2 database.



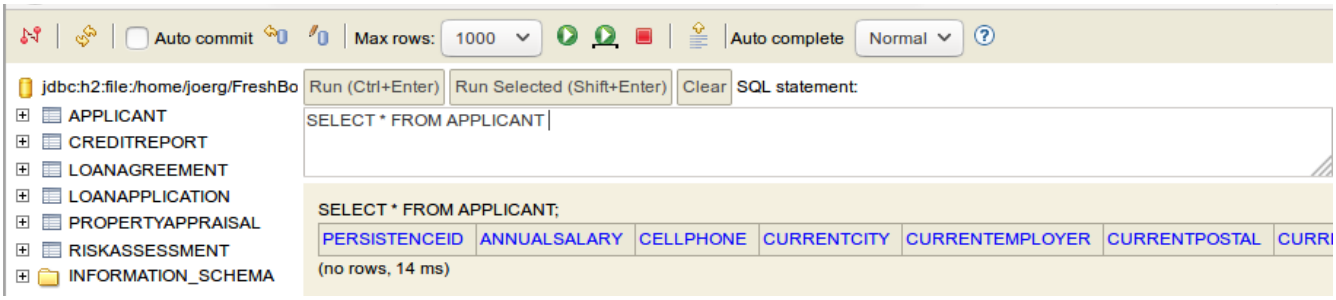
Note the tables corresponding to each business object on the left hand side. You can query the contents of each table with the appropriate SQL statement.

- Select the APPLICANT table on the left

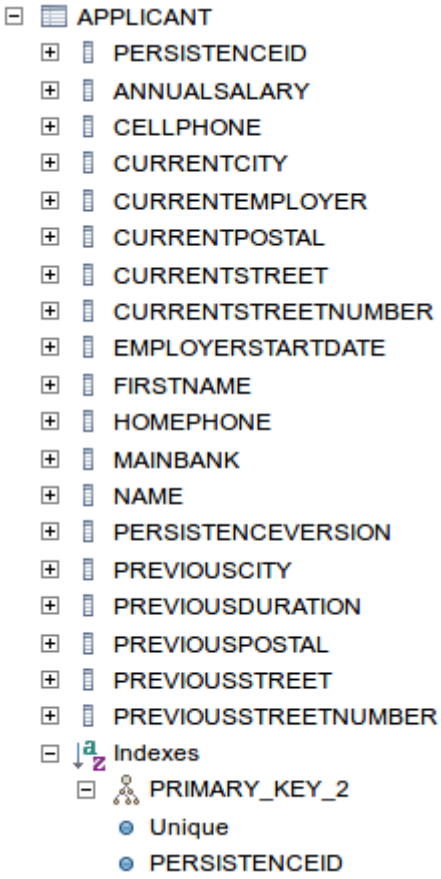
The H2 console will create a generic SQL select statement for you.

- Press the “Run” button to execute the query as in the following example.

Note the PERSISTENCEID column that is the artificial primary key.



Expanding the table name shows all columns, indices, and primary keys defined for the table based on the business object definition. Note the column PERSISTENCEVERSION that is used to version the business objects in the database.



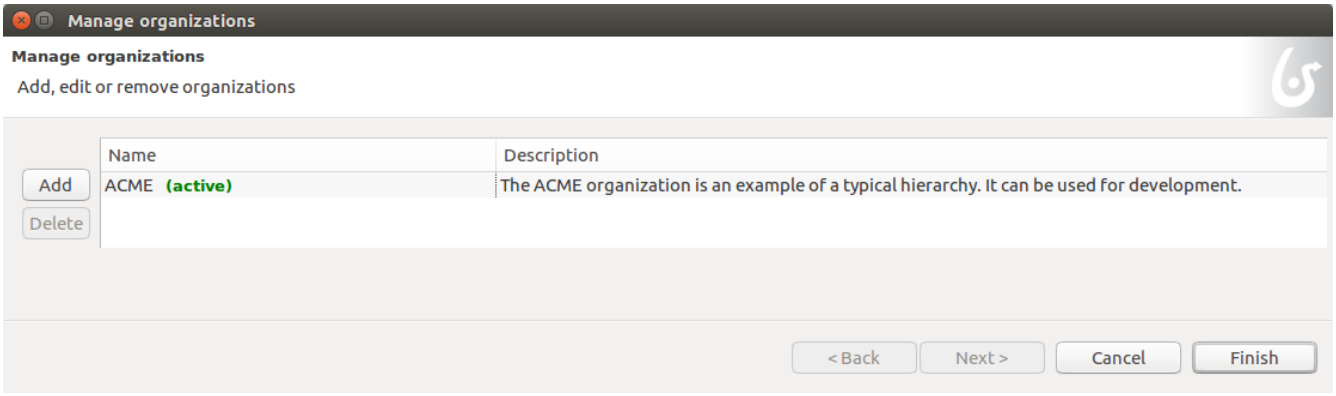
Organizational Management

Bonita separates the organizational management of the workflow server from actor specifications in the BPMN pools and lanes. Organizations are managed on the workflow server by specifying a set of groups and roles to which individual (human) resources can be assigned. Groups and roles are independent of each other. Each actor specified for a pool or lane is later mapped to these groups and roles.

In our simple example, you can look to Figure 10.12 in "Fundamentals" to guide the definition of organizational resources. You will model the workflow server organization to closely resemble the actors specified for the pools and lanes so that the mapping between them will be easier.

- Select "Organization" → "Define ..." from the menu.

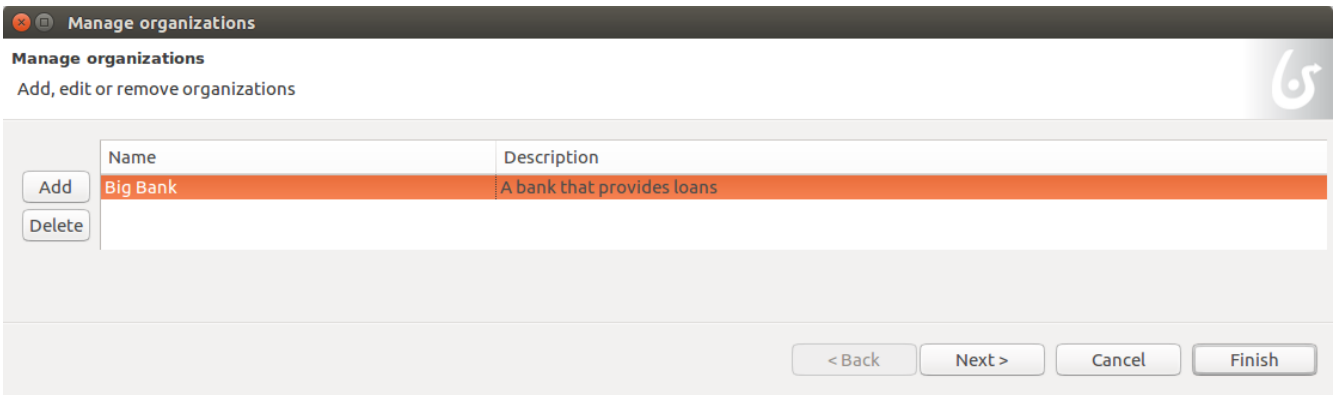
You can see that the "ACME" organization is predefined. Rather than using this, you will define your own organization.



- Select the ACME organization, then push the "Delete" button
- Confirm that you want to delete this organization
- Push the "Add" button to create a new organization.

You can change the default name of "Organization1" by clicking on it and then typing the new name. You can also provide a short description.

- Change the name of the organization to "Big Bank"

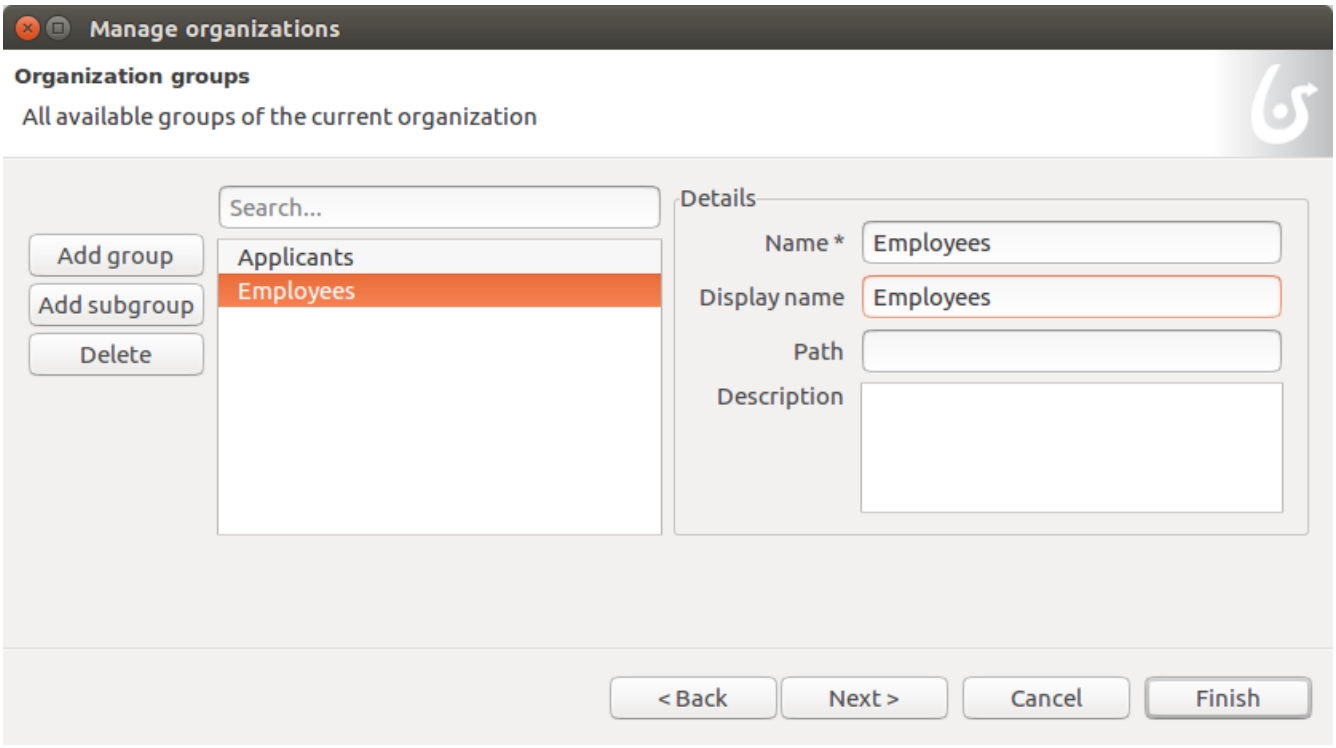


- Make sure the Big Bank organization is selected, then press the "Next >" button.

The following dialog box allows you to create organizational groups. For this tutorial, create two groups, one of Employees and one of Applicants, who will be interacting and carrying out process activities.

- Use the "Add group" button to create a new group, then
- Set its name and display name on the right in the details fields

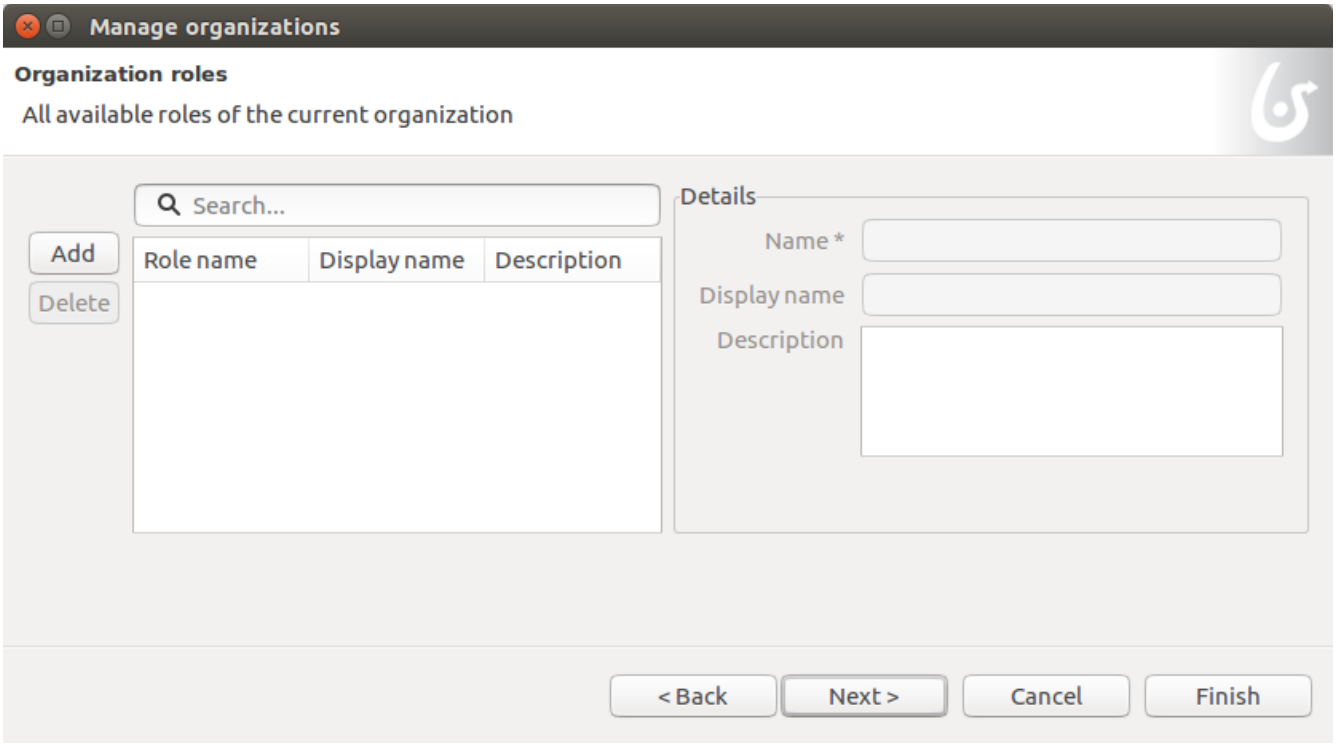
To keep things simple, you will not work with subgroups for this tutorial.



After you have created the two groups,

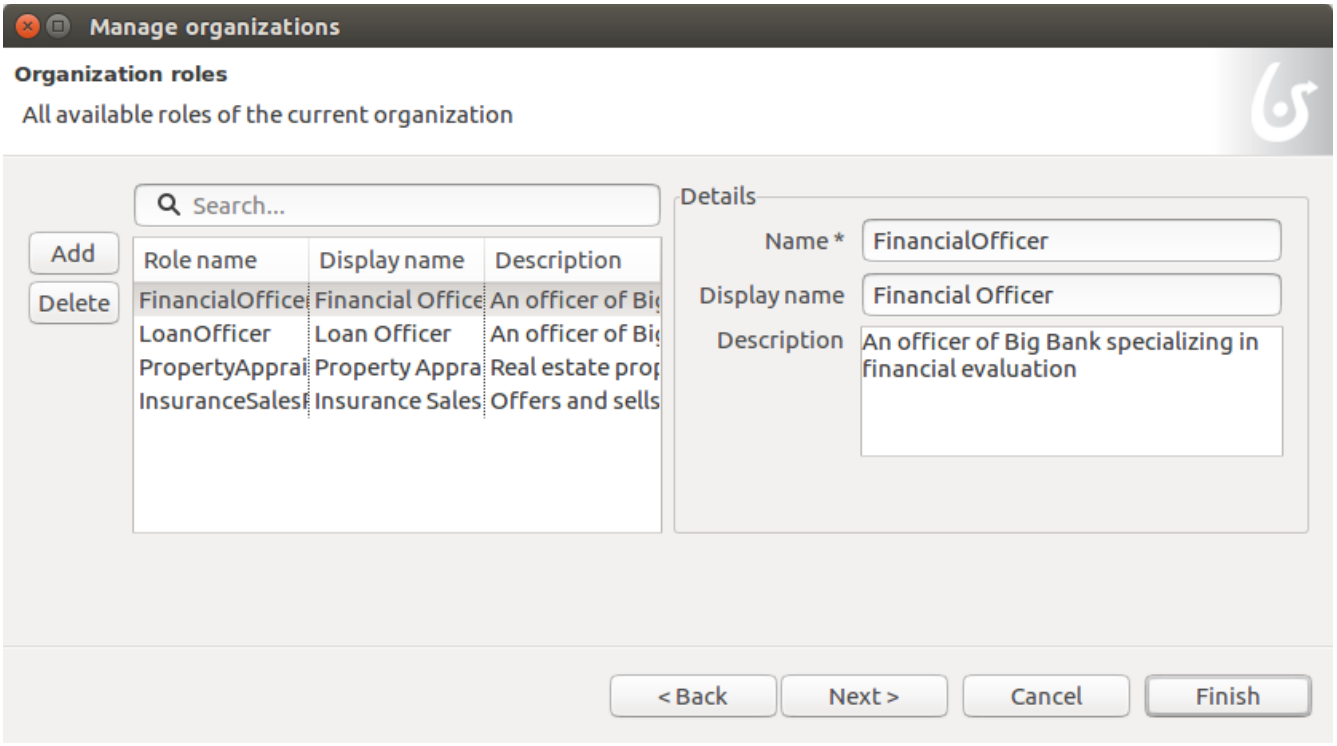
- Press the "Next >" button

to continue with the definition of roles.



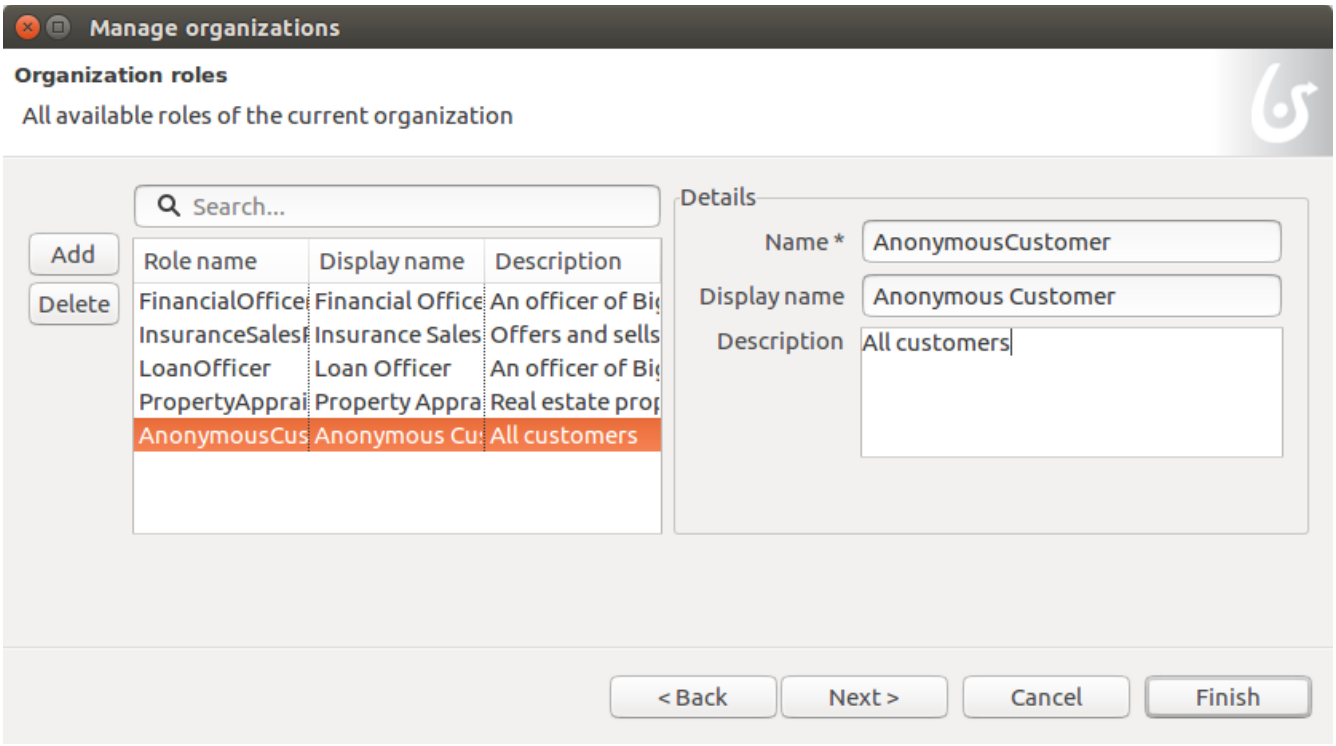
You will define four roles, corresponding to the actors for the four lanes in Fig 10.12 in "Fundamentals". For each role,

- Push the "Add" button to create a new role, then
- Change its name and display name on the right in the details fields.



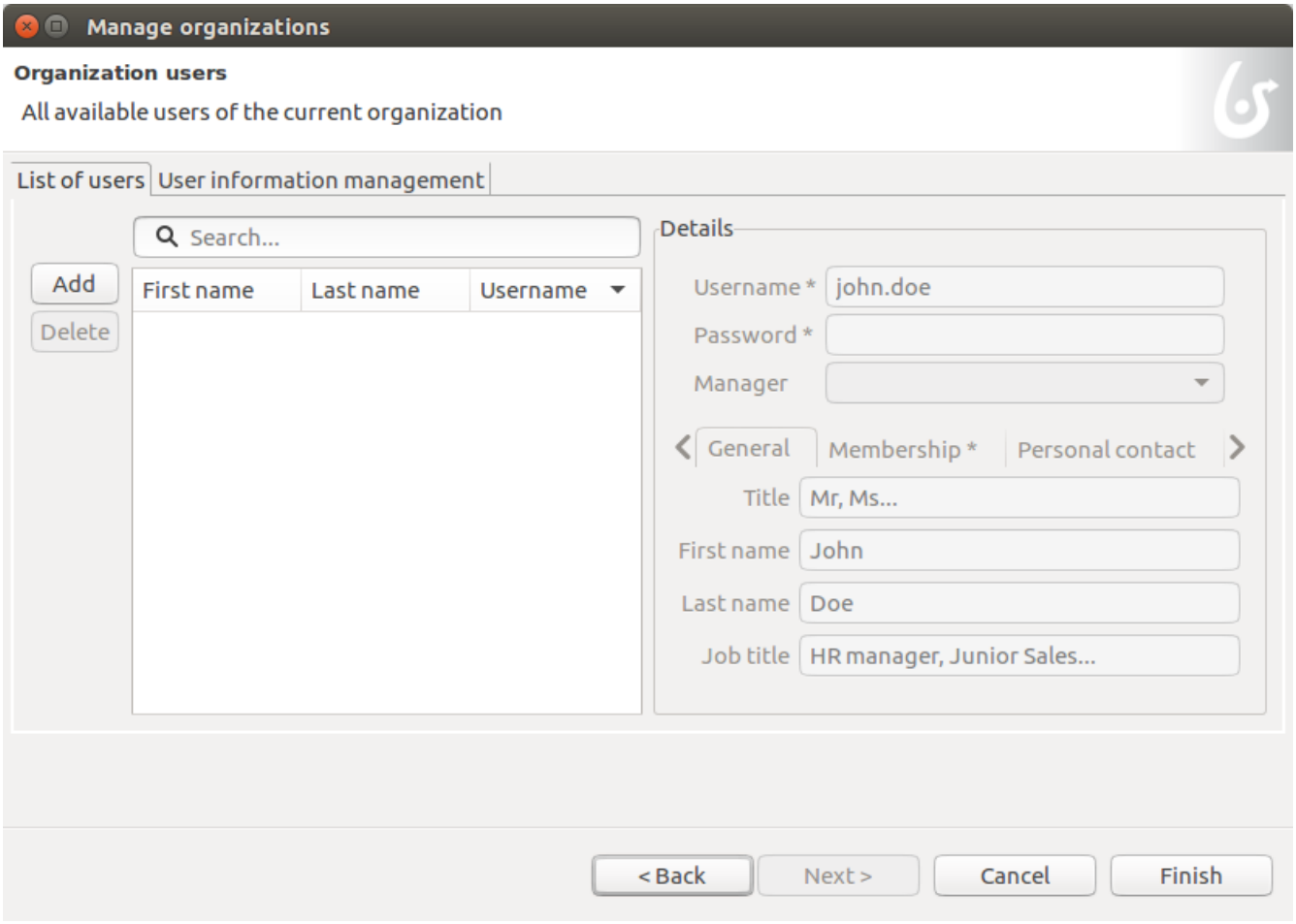
You will also define an additional role for all customers that interact with Big Bank, as they also use the workflow system to initiate and revise loan applications.

- Create the anonymous customer role as shown below



- Push the “Next >” button to continue to the definition of users

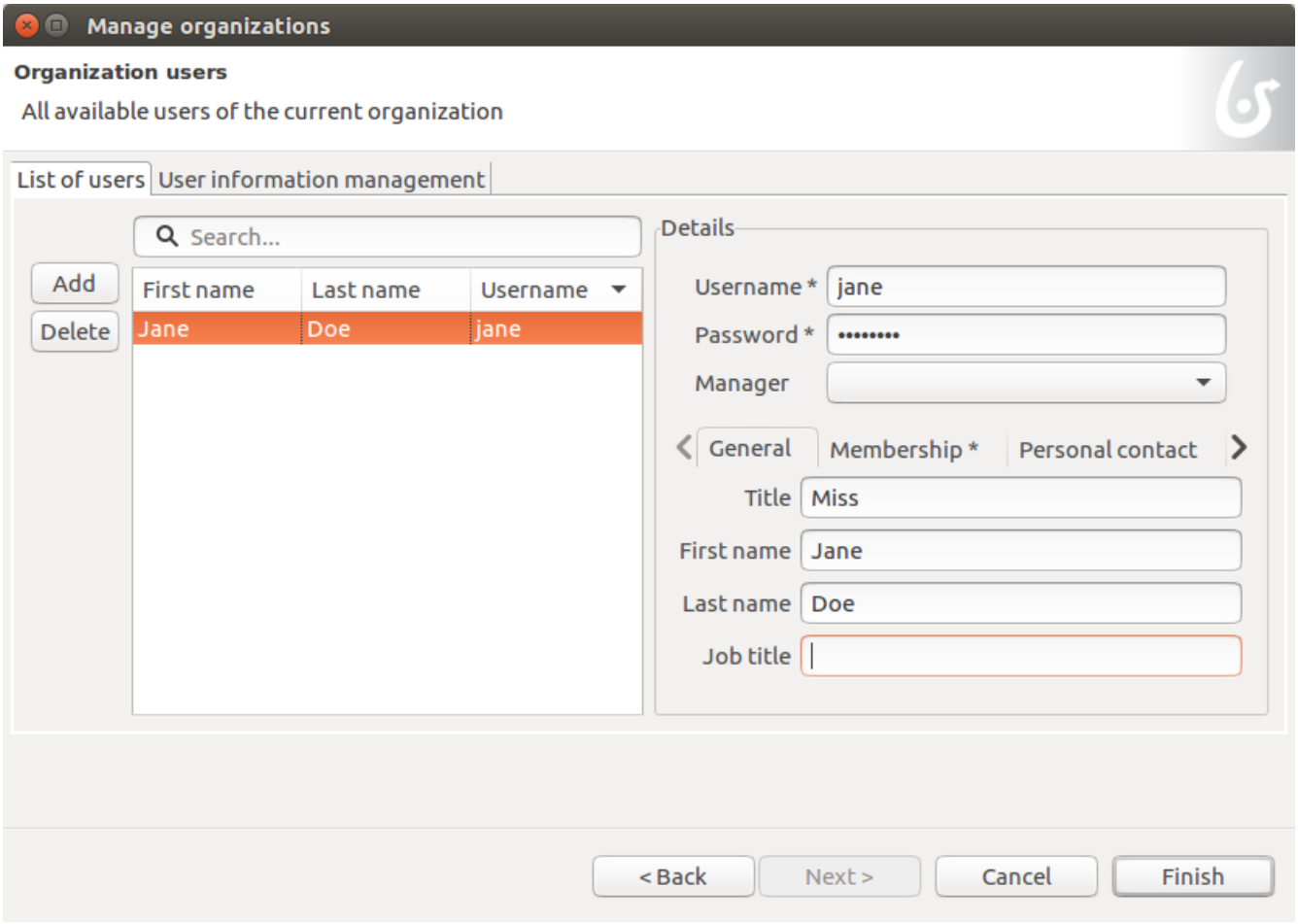
Users represent individuals and their user accounts for the workflow management system. To keep things simple, you will create only two resources, one that represents a customer, and one that represents an employee.



For each user account,

- Push the “Add” button on the left, then
- Change the details on the right of the form as appropriate.

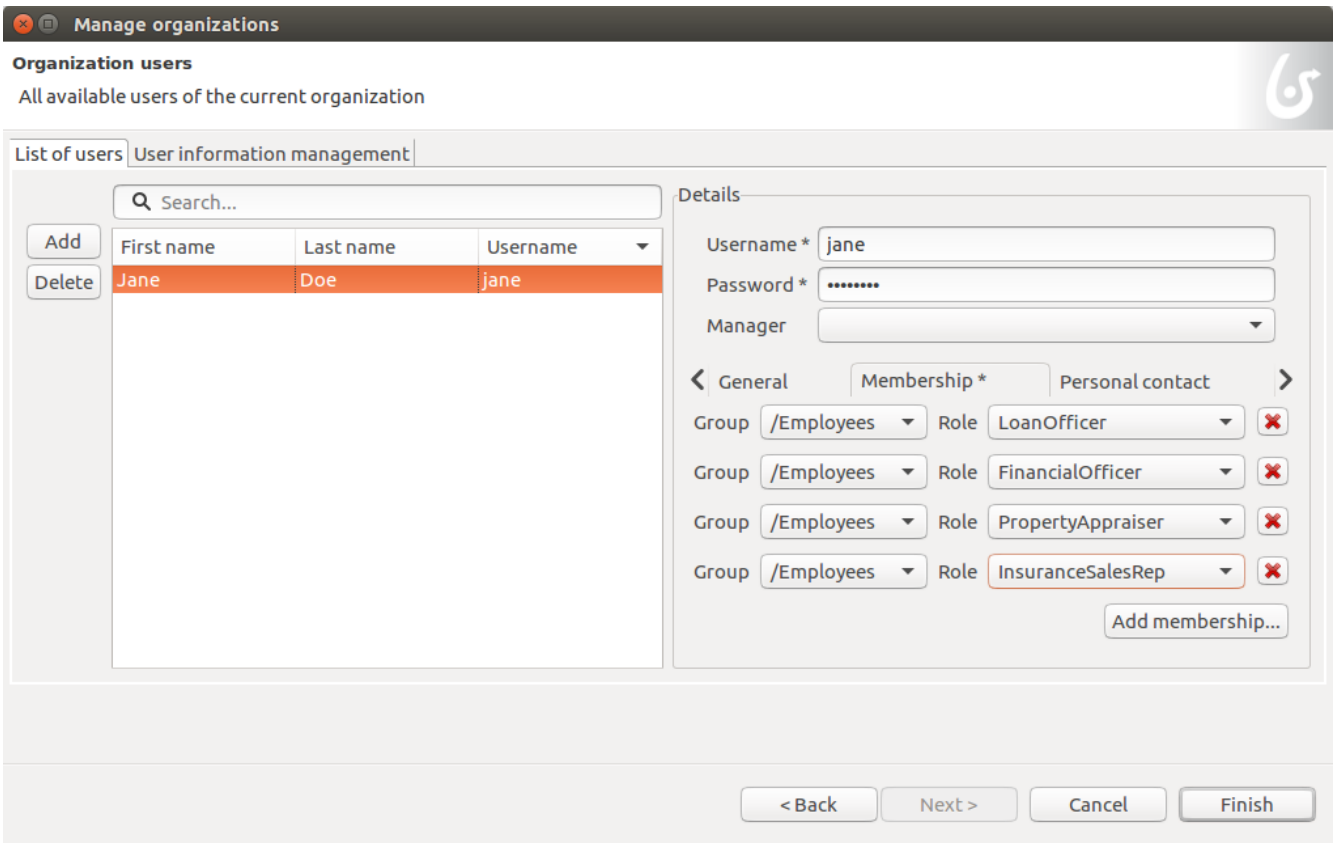
The example in the figure on the next page shows the definition of user Jane Doe with user name “jane” and some password.



For each user account,

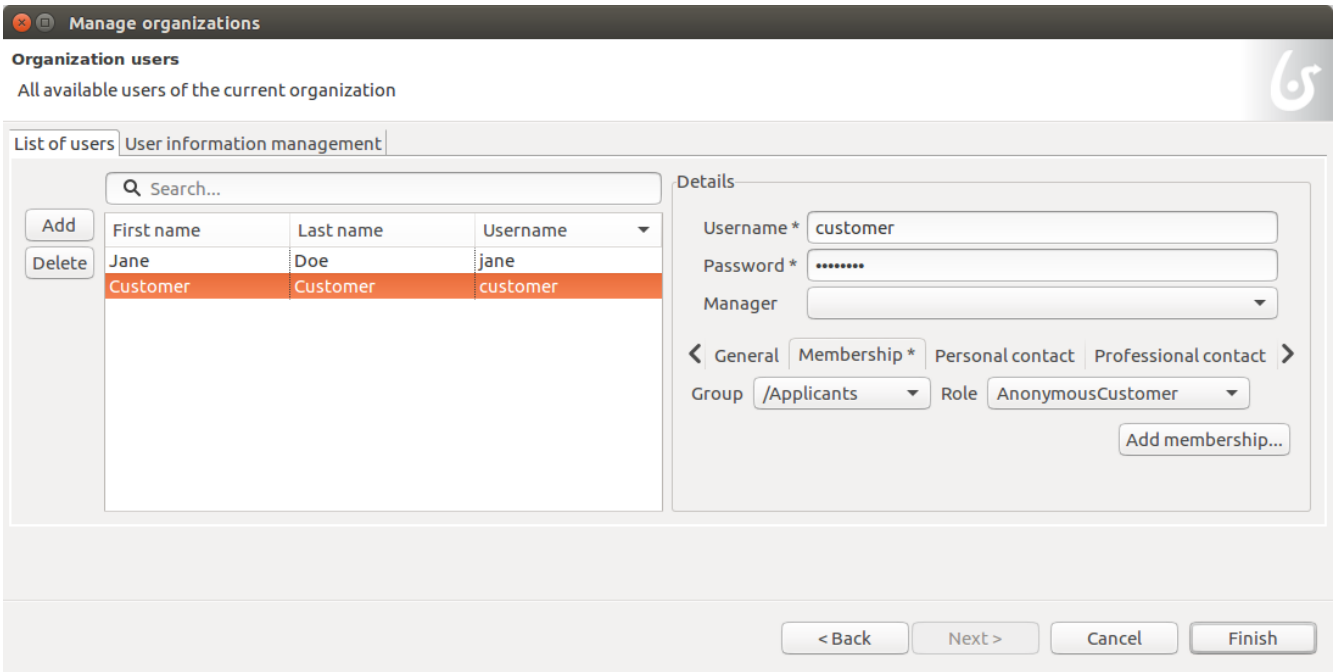
- Select the "Membership" tab in the details, and
- Assign this user to the appropriate groups and resources that you defined earlier.

You can assign multiple groups and roles to each user. Note that groups and roles are independent. In the following example (next page), Jane Doe is a member of all employees who are also loan officers, a member of all employees who are also financial officers, etc.



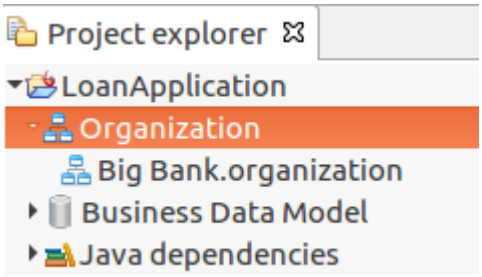
- Define the customer user account in a similar way and ensure that this account is in the applicants group and customers role.

The following figure shows you what the customer account memberships should look like.



- Push the “Finish” button to complete the definition of the organizational model on the workflow management server.

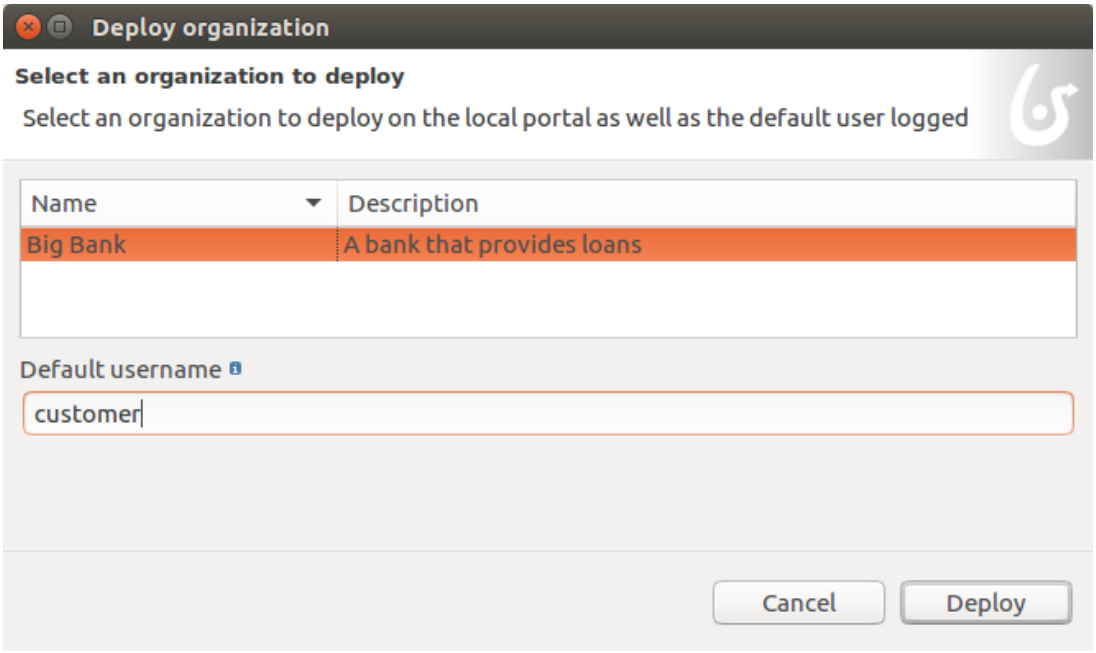
In the Bonita Studio, you will see your organization as part of the current project.



Organizational Deployment

The organization, as defined just now, needs to be deployed to the Bonita workflow management server. In Bonita Studio,

- select “Organization” → “Deploy ....” from the menu.



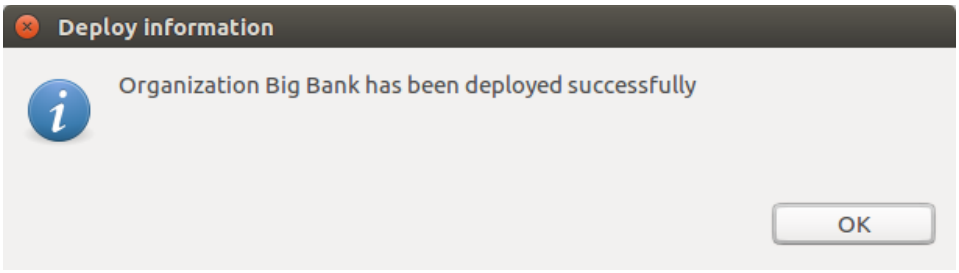
- Select the “Big Bank” organization you have just defined

The default user name is the one that Bonita Studio should use when opening the web portal to the workflow management server. In our case,

- Specify “customer” as the default user name

The customer will launch initiate the loan application process by completing and submitting a loan application as the first task.

Bonita will acknowledge successful deployment:

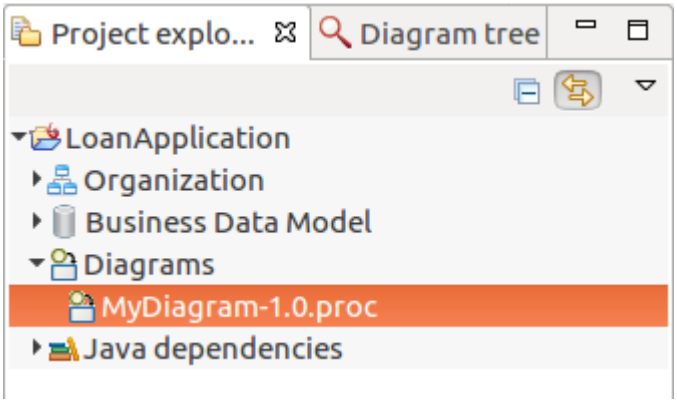


Process Model

To begin describing the process in BPMN, create a new diagram in Bonita Studio.

- Select “File” → “Create new diagram ...” in the menu.

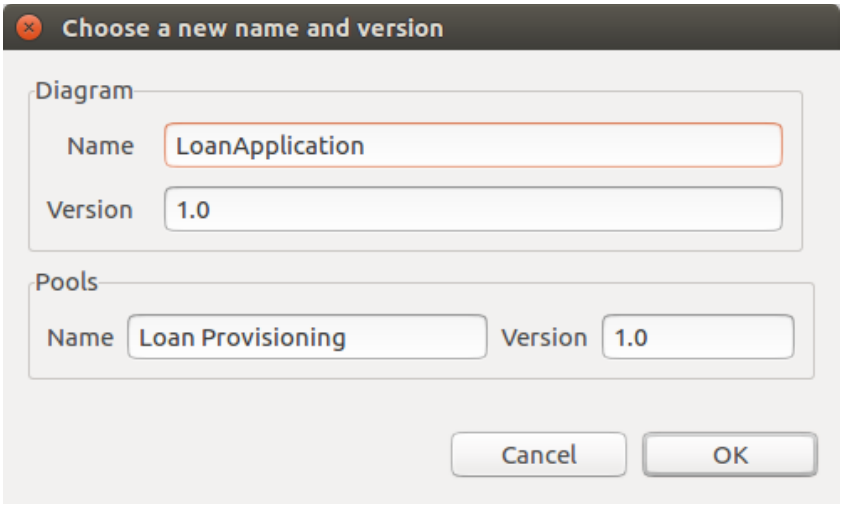
Once the diagram has been created, you can select it in the project view on the left of Bonita Studio.



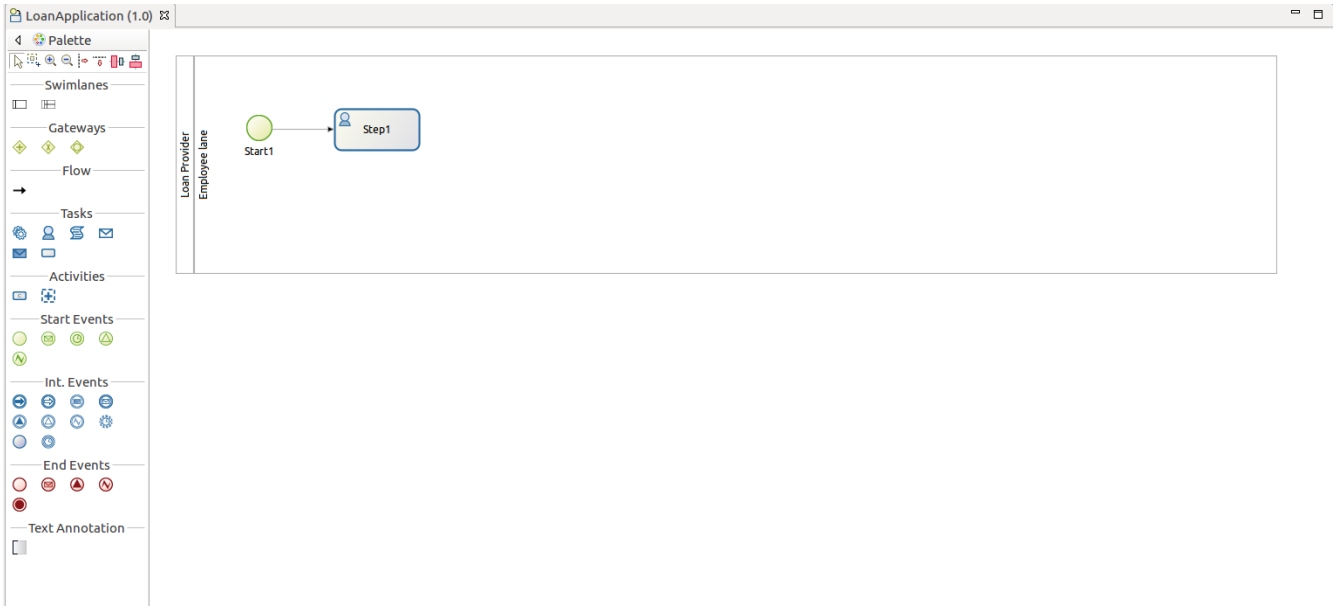
You can rename the diagram, as well as the pool(s) in the diagram.

- Select the new diagram and right-click and select “Rename ...” to provide a better name for it.
- Name the pool “Loan Provisioning”.

While Fig. 10.12 in “Fundamentals” labels the pool after the organizational unit performing the work (“loan provider”), Bonita uses the convention that the pool name designates the process, not the organization, hence you should label it “Loan Provisioning”.



The diagram should now show a single pool with a single lane and the beginnings of a process, as in the following image (next page). Note the palette of BPMN symbols to the left of the drawing area.



Pools, Lanes, and Actors

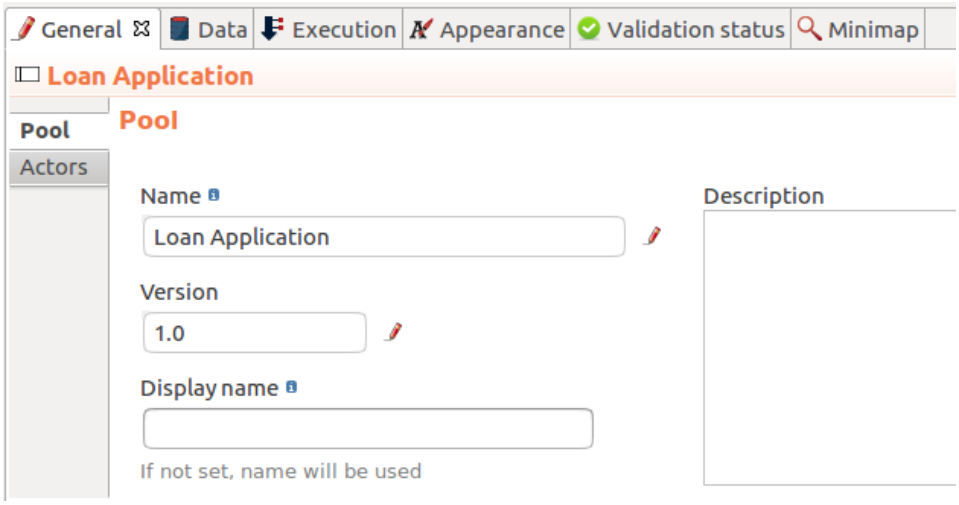
Because we want to model the message passing coordination between the loan application and the loan provisioning processes that is shown in the collaboration diagram in Fig. 10.12 in “Fundamentals”, you will add another pool to the diagram.

- Select the pool symbol in the palette, then
- Add a new pool above the loan provisioning.

You can name this new pool in the general properties below the drawing area.

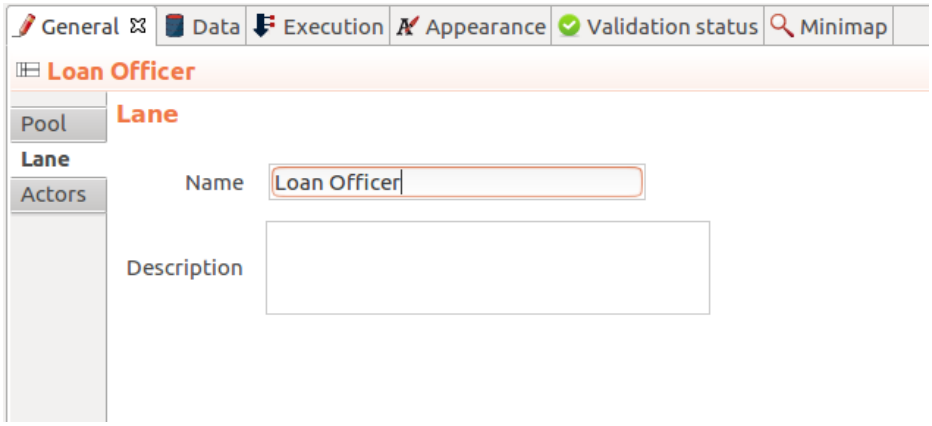
To rename the pool,

- Push the button with the pencil icon to enable editing of the pool’s name.
- Change the name to “Loan Application”.



The loan provisioning pool has four lanes, one each for the loan officer, the financial officer, the property appraiser, and the insurance sales rep.

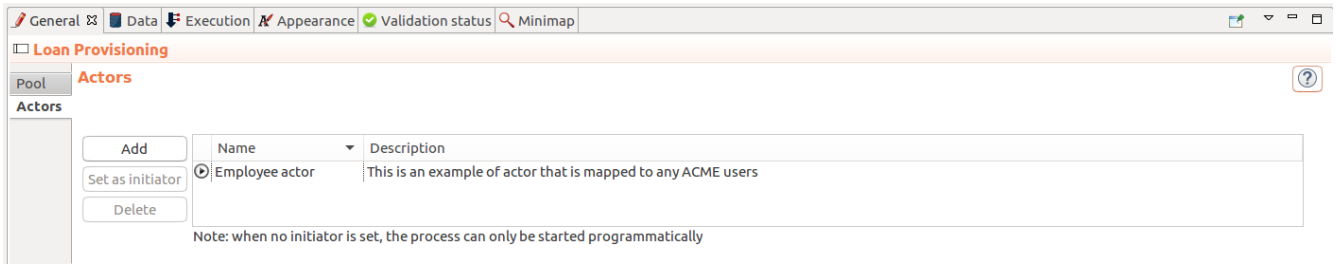
- Select the "Employee lane" pool, and
- In the "Lane" tab in the properties pane, rename the lane to "Loan Officer"



- Select a new lane from the drawing palette, and
- Add the new lane below the "Loan Officer" lane.
- Rename the new lane to "Financial Officer".
- Add lanes for the property appraiser and insurance sales rep in the same way.
- Add a lane to the Loan Application pool. Label this lane "Applicant".

Next, you will assign actors to each pool and lane. Actors in the BPMN diagram are distinct and separate from the organizational resources you have defined earlier for the workflow server. They will be mapped to these resources when the process is deployed for execution.

- Select the loan provisioning pool and the "Actor" tab in the properties pane.



A default actor is already entered here.

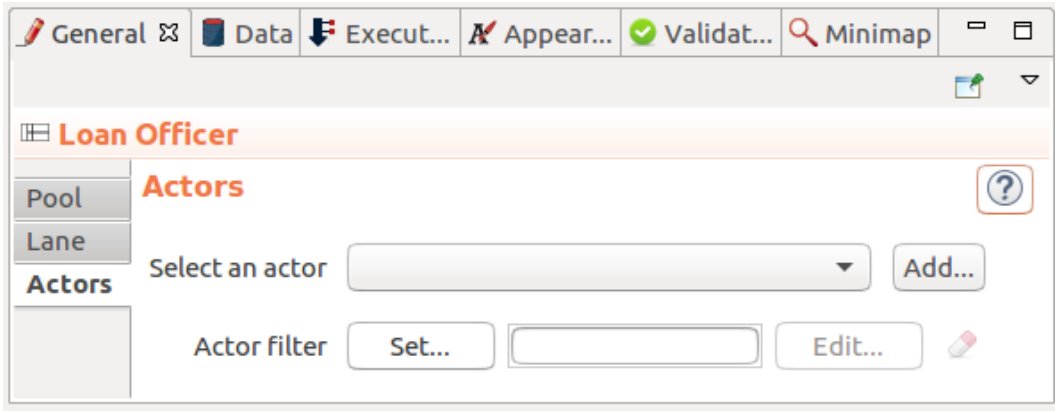
- Select the default "Employee actor", unset it as initiator, and then delete it.

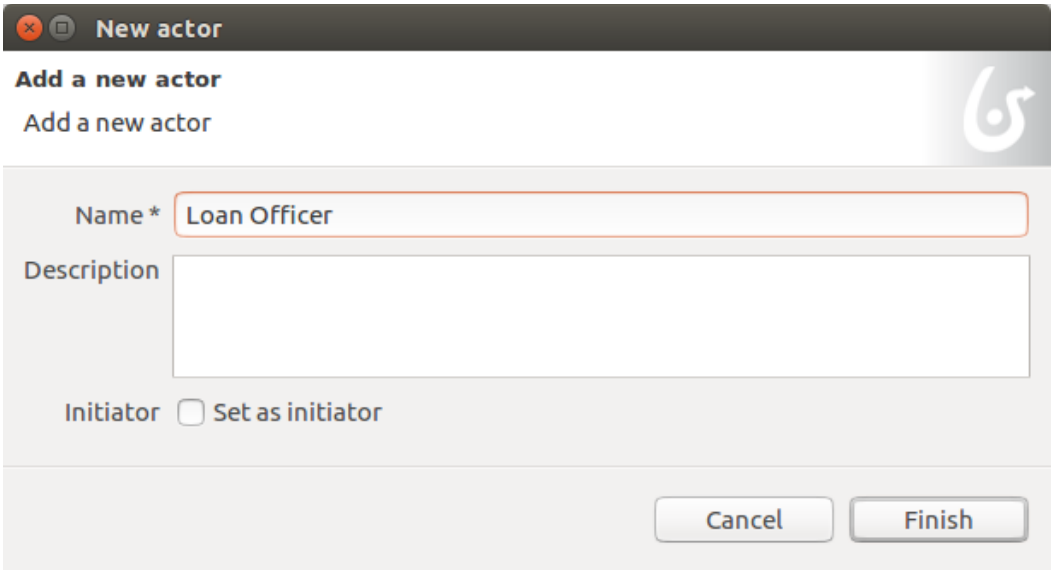
Confirm that you really want to delete this actor. Next,

- Use the "Add" button to add a new actor,
- Rename the default "Actor1" name by selecting it and replacing it with "Big Bank Employee". Do not set this actor as initiator.

Next, you will define actors for each lane within the loan provisioning pool.

- Select the loan officer lane, then
- Select the "Actors" tab in the properties pane and press the "Add ..." button
- In the following dialog box, enter the name "Loan Officer" and press finish.

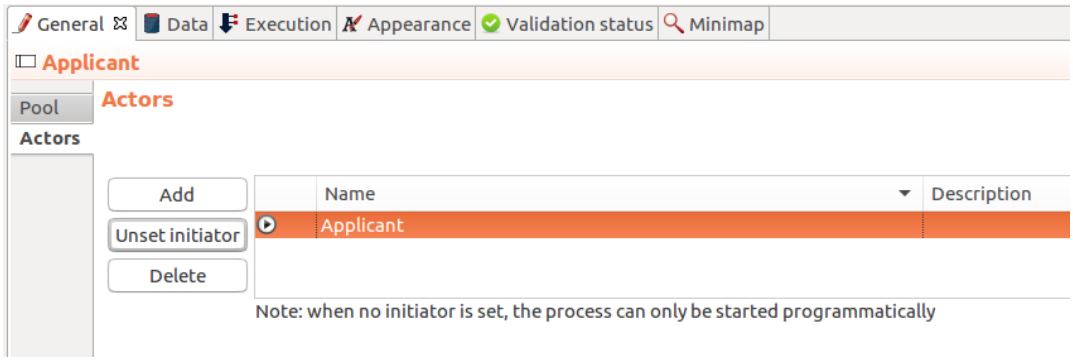




- Create new actors for the remaining three lanes in the same way.
- Create a new actor for the loan application pool in the same way.

The actor of the loan application pool will manually initiates the process of applying for a loan. In contrast, the loan provisioning actors react only to inbound application messages. They should not be able to instantiate the loan provisioning actors manually.

- Set this actor as initiator



You must also specify an actor for the Applicant lane in the loan application pool. Select this lane. Instead of creating a new actor,

- Use the drop-down list to select the "Applicant" actor defined for the loan application pool.

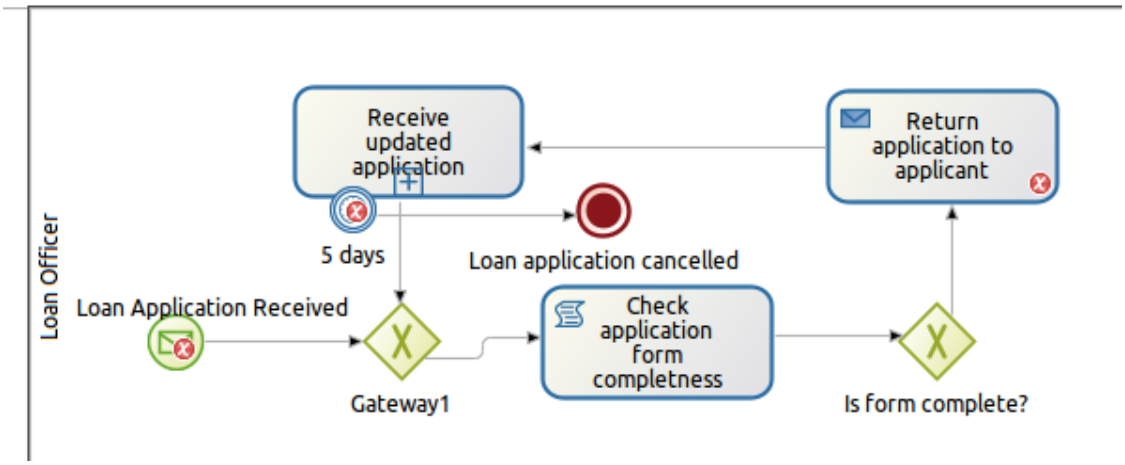
Control Flow

Next, you will model the process, staying close to Fig. 10.12 in "Fundamentals".

Examining that figure, you will notice that Bonita does not offer symbols for data objects and data stores. These can still be added to each task, as you will see later, but are not represented graphically. Also, when modelling the messages in the collaboration with the applicant pool, Bonita requires these message connections to originate or terminate at a message event or message task, rather than at the edge of an opaque ("black") pool as in "Fundamentals". Note also that message connections cannot be drawn graphically, but are added automatically by Bonita once the message details have been specified.

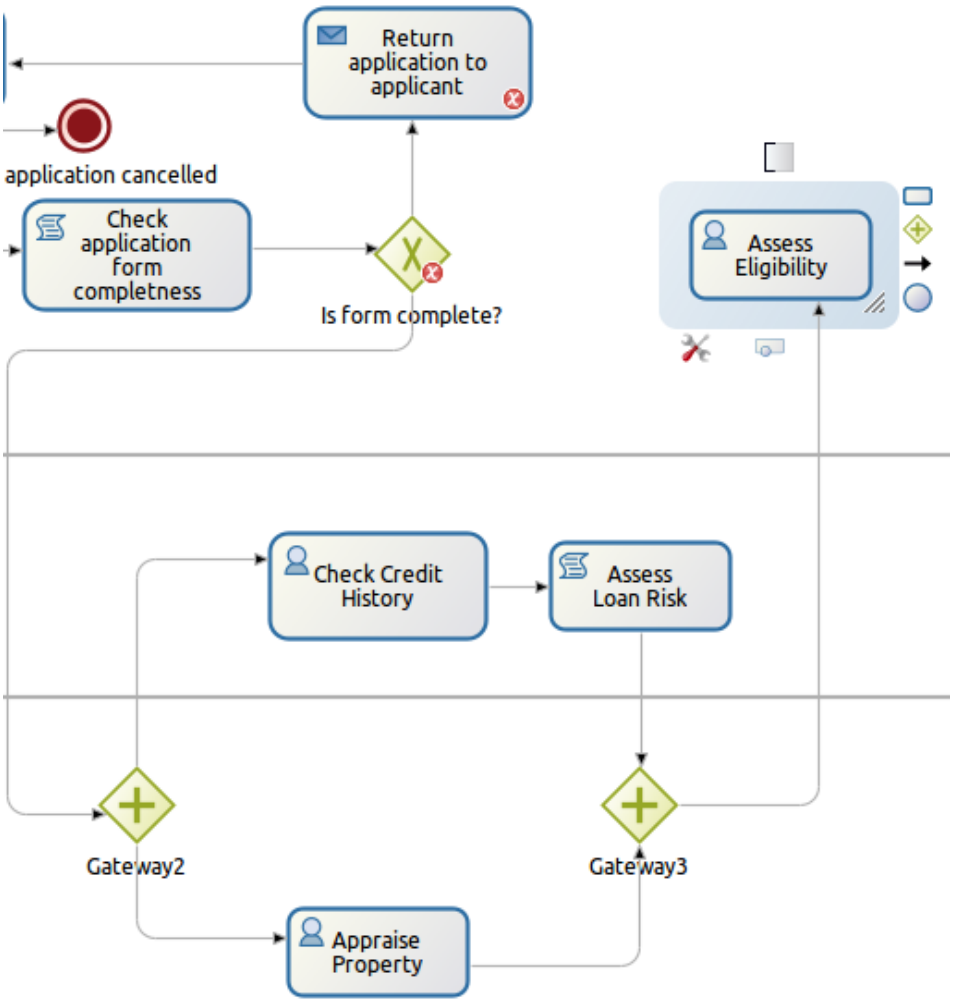
Begin with the basic BPMN elements for the loan officer.

- Model the following process fragment in the Loan Officer lane



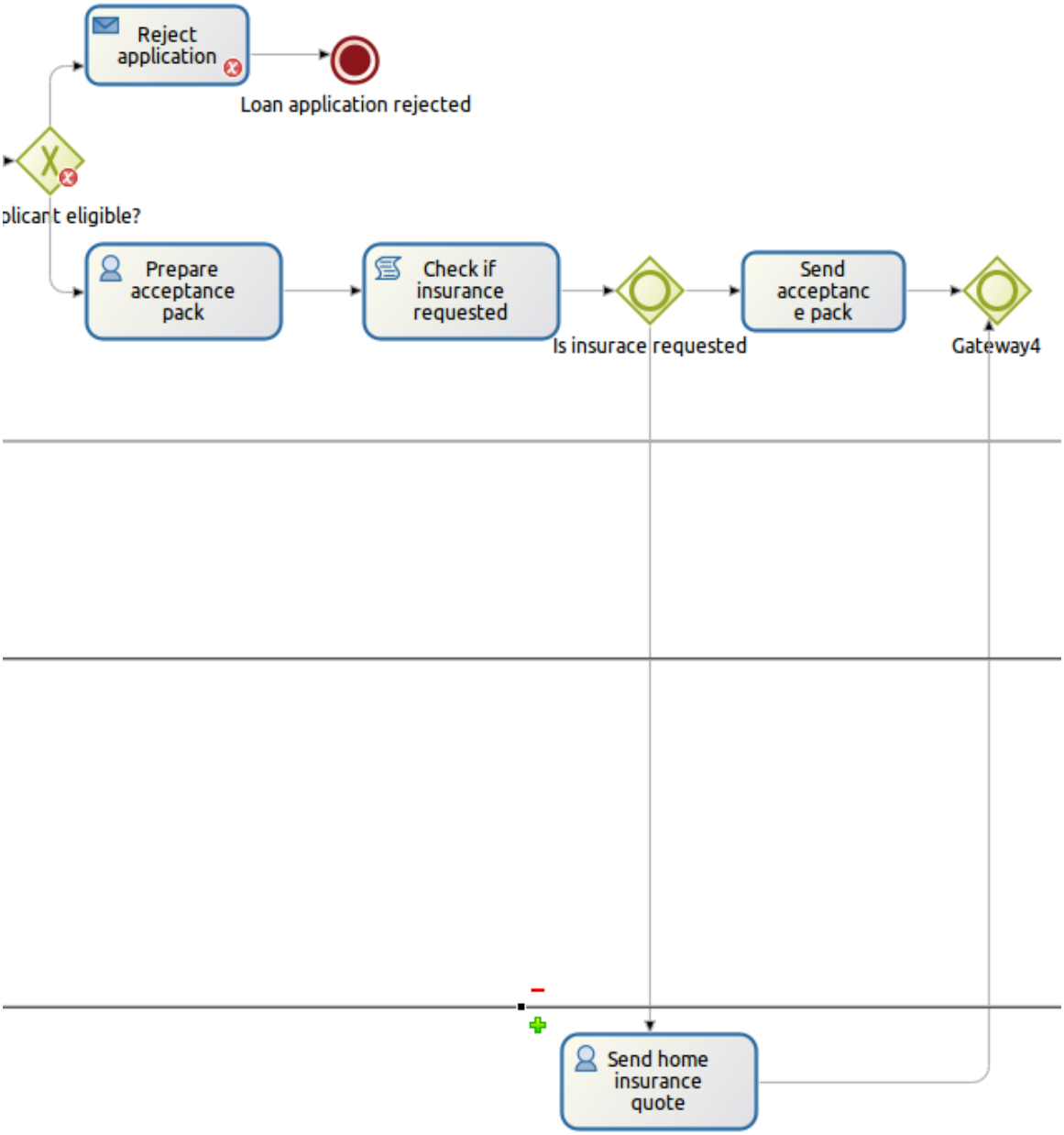
Explanation: Because Bonita does not allow using timer events as boundary events on message receive tasks, as is done in Fig. 10.12 in "Fundamentals", you will use a sub-process, which is called *Call Action* in Bonita, for the activity "Receive updated application". You will place the message receiving activity into that sub-process. An interrupting timer event can be added to the call action.

- Continue modelling the process for the financial officer and property appraisers as shown in the following diagram (next page)

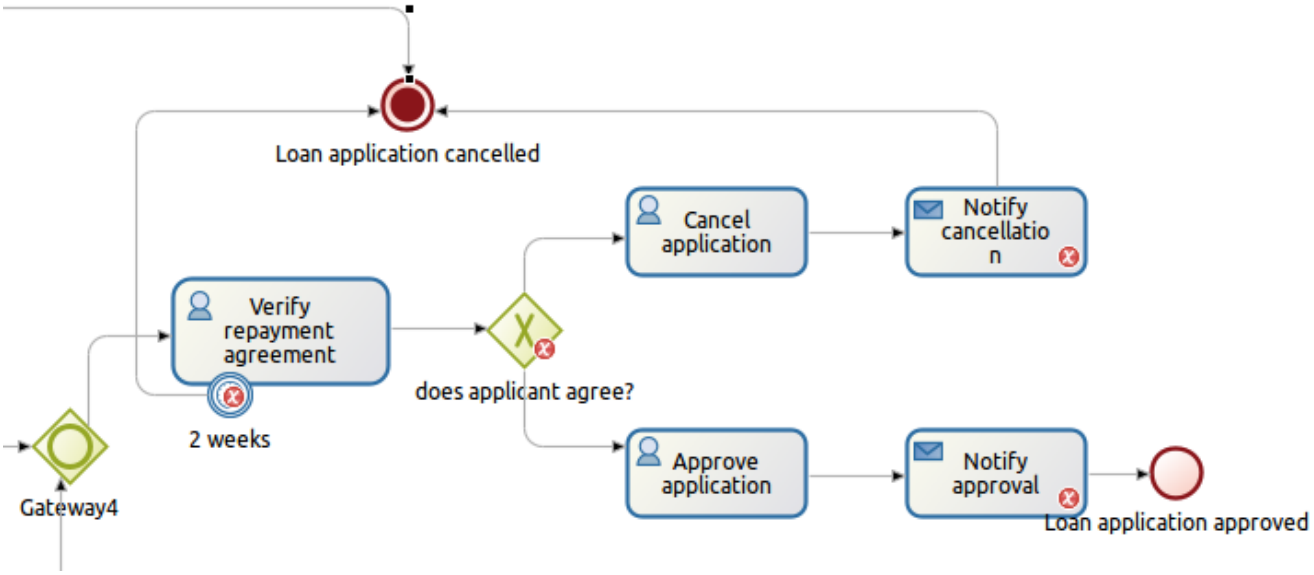


Explanation: Use a script task, instead of a service task, for the activity "Assess Loan Risk", as our executable process in this tutorial will not have access to an actual credit rating service.

- Continue modelling the next parts of the process for the loan officer and the insurance sales rep, as shown in the following diagram (next page)

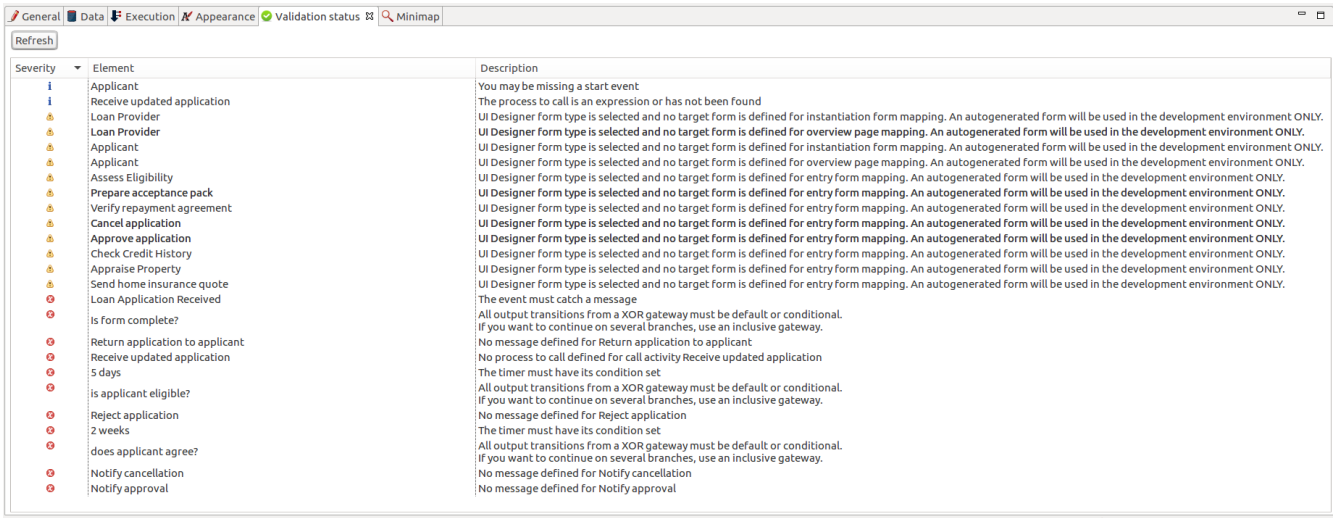


- Model the remainder of the process for the loan officer, as shown in the following diagram (next page)



Explanation: In contrast to Fig. 10.12 in "Fundamentals", this model contains only one terminating event "Loan application cancelled" to which the "5 day" boundary timer event, the "2 weeks" boundary timer event, and the "Notify cancellation" sending activity are connected. This is because the Bonita does not allow multiple events with the same name.

At this point, you will notice that many of the elements are labelled with a red 'x' symbol, indicating that there are errors in the diagram. You can find out more about these errors in the "Validation Status" tab of the properties pane below the diagram. You may need to refresh this view to show the most current information.


General | Data | Execution | Appearance | Validation status | Minimap

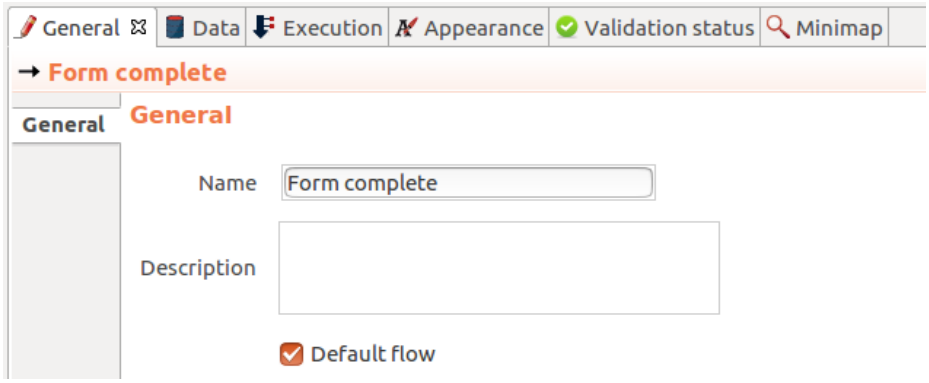
Refresh

Severity	Element	Description
i	Applicant	You may be missing a start event
i	Receive updated application	The process to call is an expression or has not been found
⚠	Loan Provider	UI Designer form type is selected and no target form is defined for instantiation form mapping. An autogenerated form will be used in the development environment ONLY.
⚠	**Loan Provider**	**UI Designer form type is selected and no target form is defined for overview page mapping. An autogenerated form will be used in the development environment ONLY.**
⚠	Applicant	UI Designer form type is selected and no target form is defined for instantiation form mapping. An autogenerated form will be used in the development environment ONLY.
⚠	Applicant	UI Designer form type is selected and no target form is defined for overview page mapping. An autogenerated form will be used in the development environment ONLY.
⚠	Assess Eligibility	UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.
⚠	**Prepare acceptance pack**	**UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.**
⚠	Verify repayment agreement	UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.
⚠	**Cancel application**	**UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.**
⚠	**Approve application**	**UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.**
⚠	Check Credit History	UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.
⚠	Appraise Property	UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.
⚠	Send home insurance quote	UI Designer form type is selected and no target form is defined for entry form mapping. An autogenerated form will be used in the development environment ONLY.
⊗	Loan Application Received	The event must catch a message
⊗	Is form complete?	All output transitions from a XOR gateway must be default or conditional. If you want to continue on several branches, use an inclusive gateway.
⊗	Return application to applicant	No message defined for Return application to applicant
⊗	Receive updated application	No process to call defined for call activity Receive updated application
⊗	5 days	The timer must have its condition set
⊗	is applicant eligible?	All output transitions from a XOR gateway must be default or conditional. If you want to continue on several branches, use an inclusive gateway.
⊗	Reject application	No message defined for Reject application
⊗	2 weeks	The timer must have its condition set
⊗	does applicant agree?	All output transitions from a XOR gateway must be default or conditional. If you want to continue on several branches, use an inclusive gateway.
⊗	Notify cancellation	No message defined for Notify cancellation
⊗	Notify approval	No message defined for Notify approval

You will now attend to each of these problem issues in turn.

First, for each of the three branching exclusive (XOR) gateways, name the flows and indicate the default flow. To do this,

- Select the default flow from each gateway.
- In the properties pane, provide a name and indicate that it is the default flow.



- Do this for all flows from branching exclusive (XOR) gateways.

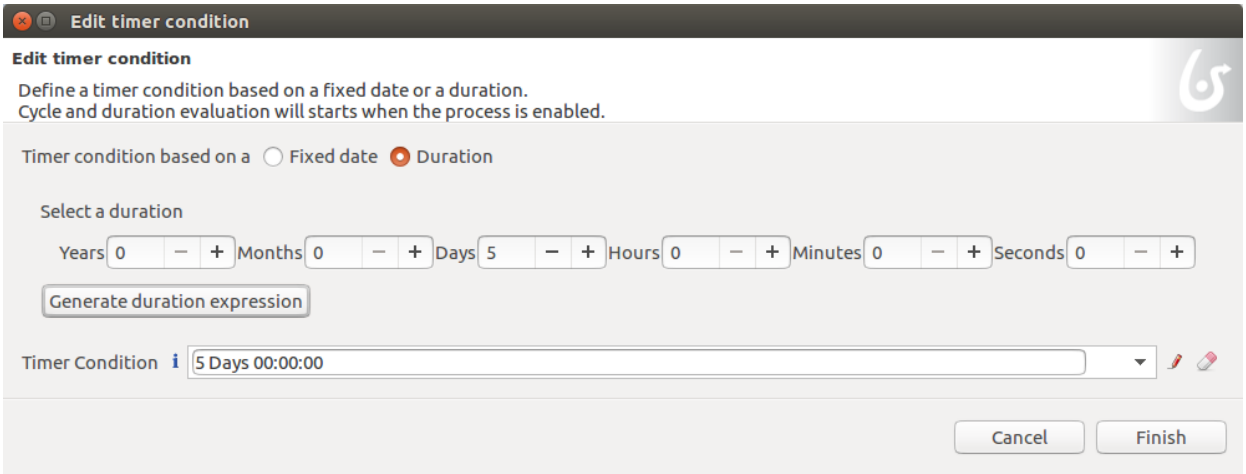
Your changes should be reflected in the diagram by showing the labels as well as the indications for default flows.

Next, you will specify the two timer events in the model.

- Select the "5 days" timer event.
- In the properties pane, push the "Edit ..." button to specify the timer.

In the following dialog window,

- Select "Duration" for the type of timer, enter "5" for the number of days, and push the button "Generate duration expression" to generate the timer condition.
- Push the "Finish" button.

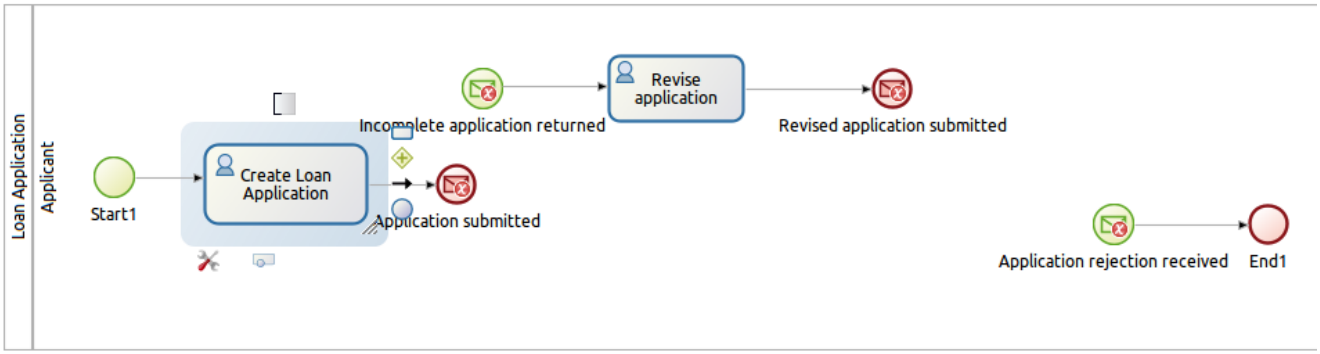


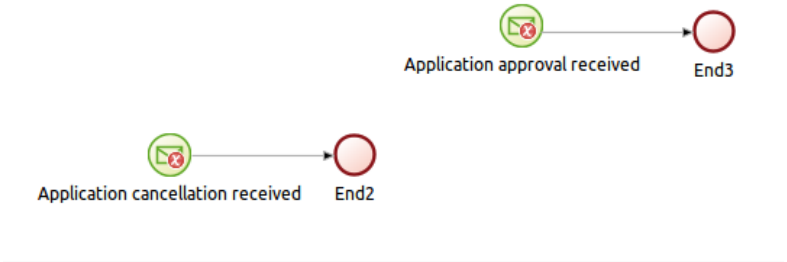
- Repeat this for the "2 week" timer, using 14 days as the duration.

To specify the details of messages, you need to explicitly model those activities of the applicant that the applicant will complete on the workflow management system itself. While in a production environment, these messages may be sent from external systems to start a process instance, for this tutorial you should assume that applicants create and revise loan applications also on the workflow management system. Recall that you have created a customer/applicant user account for this purpose.

The applicant initiates the process by submitting a loan application, and may be requested to revise or edit loan applications to be resubmitted. The applicant also receives rejection, cancellation, and approval messages.

- Model the following process for the loan application pool.

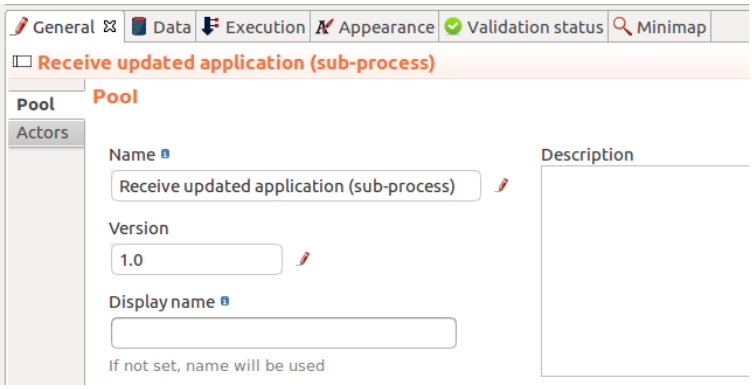




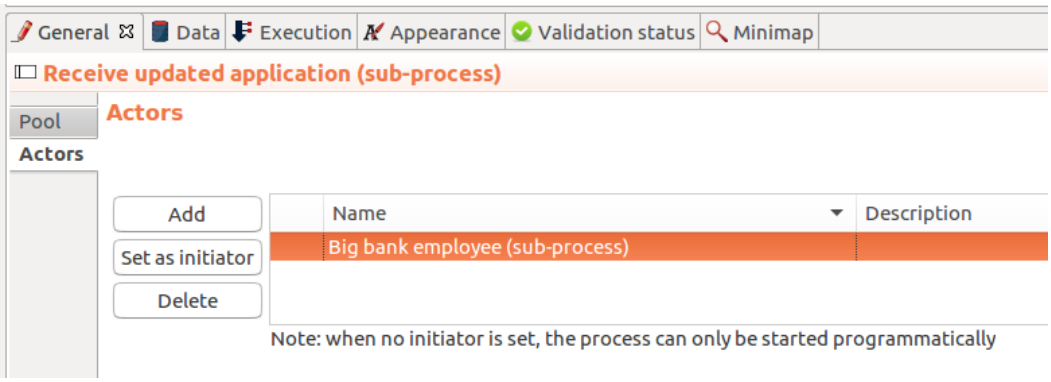
You do not need to provide any further details on the activities of the applicant, as the focus is on the loan provisioning process.

Next, you will specify the details of the call activity “Receive updated application” by creating another process pool. Bonita will not properly create message connections between pools in separate diagrams, hence you will create this sub-process pool in the same diagram.

- Select the pool symbol from the drawing symbol palette to the left of the drawing area.
- Create another pool below the loan provisioning pool.
- Rename the pool to “Receive updated application (sub-process)”.

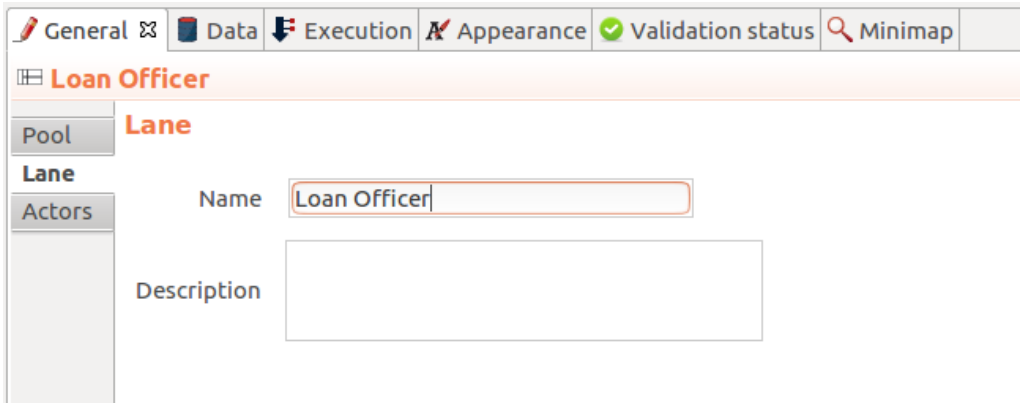


- Select the “Actors” tab in the properties pane below the diagram.
- Add a new actor to the pool, named “Big bank employee (sub-rpcoess)”.

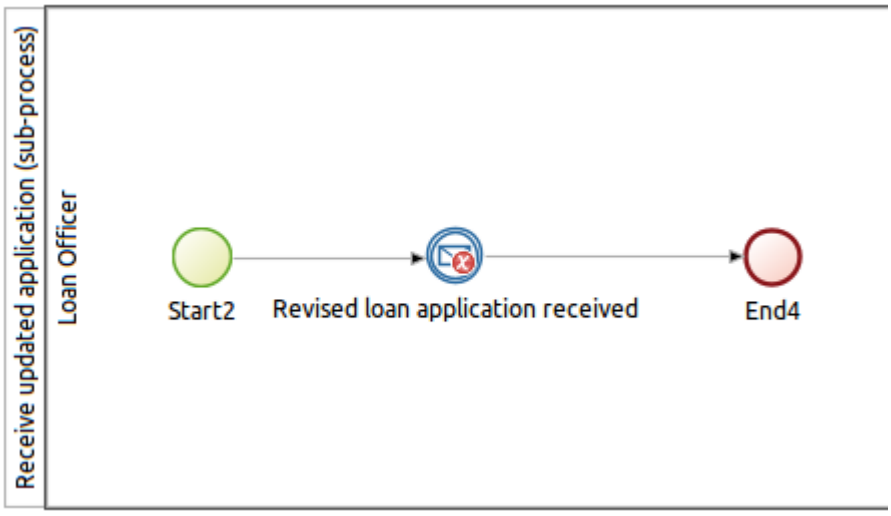


Next, use the lane symbol from the drawing symbol palette and

- Create a single lane within this pool.
- Name this lane “Loan Officer”, as in the following figure (next page).



- Model the following sub-process for the loan officer.

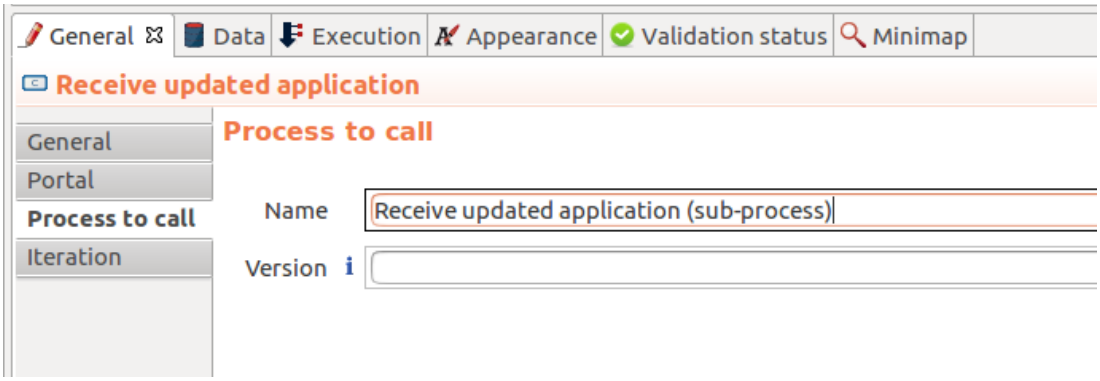


Explanation: Beginning the sub-process with a message start activity does not work in this case. In Bonita, a sub-process with a message start event will be skipped instead of waiting for the message. Hence, the sub-process must begin with a regular start event so it is instantiated by the parent process's call activity. It must then await the message arrival using the intermediate message event.

Explanation: Note also that in Fig. 10.12 in "Fundamentals", the task "receive update application" is a message task. Instead, the sub-process simply uses a message event as nothing else needs to be done other than await the message arrival.

- Select the call activity "Receive updated application" in the loan provisioning pool.
- Select the "Process to call" tab in the properties pane below the diagram.
- From the drop-down list for the name, select "Receive update application (sub-process)".

You may leave the version blank in order that the latest version of the process definition will be used when the sub-process is instantiated. Your sub-process definition should look as in the following figure (next page).



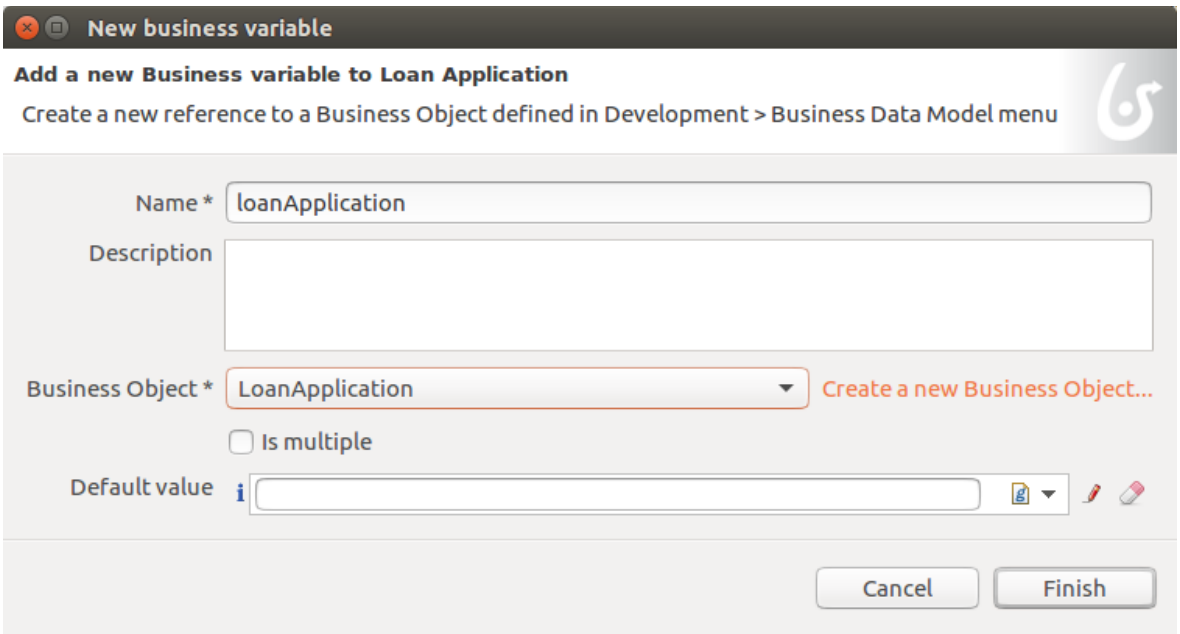
Process Data

It is useful to specify the data for each pool before specifying messages, because message sending activities or events will assemble message content from this data, and message receiving activities or events will populate this data from message content.

- Select the loan application pool, then select the "Data" tab in the properties pane.

As applicants will work with loan applications and applicant information, you will create two business variables. Additionally, you will create business variables that hold the loan application ID and the applicant ID.

- Select the "Add..." button for business variables.
- In the following dialog window, enter "loanApplication" for the name of the variable, and select the "LoanApplication" business object from the drop-down list.
- Do not enter a default value

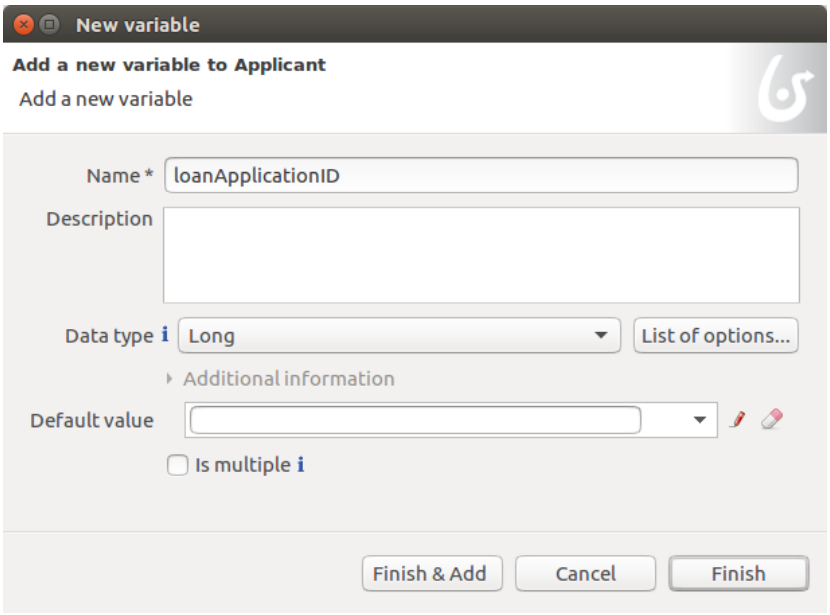


Explanation: There is no need to instantiate or set an default/initial value for the loan application variable at the pool level. Instantiation and initialization is done by the task "Create loan application". Also, there are many start events in this pool. Each time a start event occurs, a new instance of the pool's process and data is created, which would create another instance of the loan application object. You avoid this by passing the loan application into the new instance as message content.

- Complete this definition by pushing the "Finish" button.
- Repeat this procedure to define a business variable for the applicant data. Do not define a default/initial value or instantiation for this variable either.

Next, add two process variables that contain the loan application ID and the applicant ID.

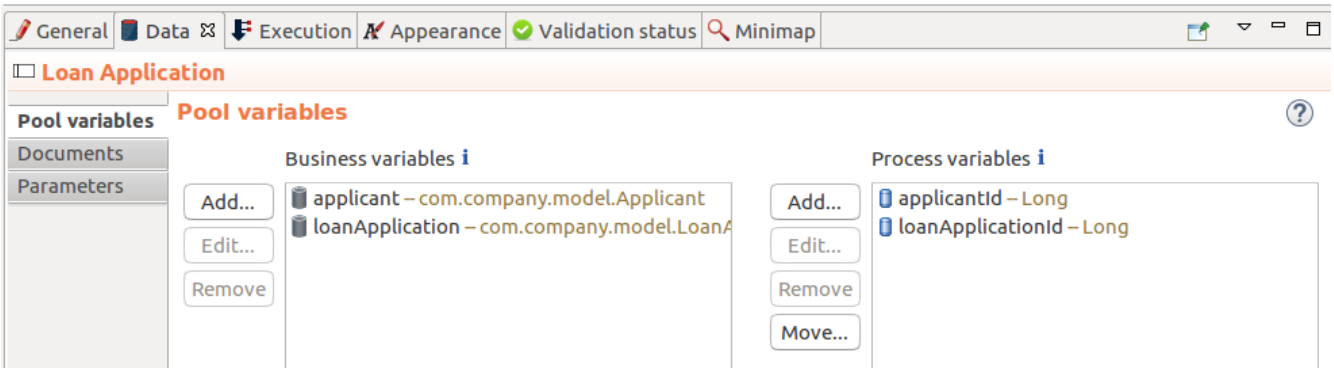
- Use the "Add..." button for the process variables to create these. Both of these should have the long data type, as shown in the following figure



Explanation: The main distinction in Bonita between business and process variables is that of persistence. Business variables are persisted beyond the execution of the workflow instance, while process variables are not. Moreover, as you have noticed when creating these variables, business variables represent business objects defined earlier (for which database tables are created) while process variables use Java data types and have no corresponding database tables.

Explanation: Bonita Community 7.9 does not send complex business objects as message content. You will circumvent this problem by sending the *PersistenceId* of the business object instead, whereupon the complete object can be read from the database after the message has been received. The *persistenceIds* of the applicant and loan application object are represented as process variables as they will become part of the message content of outgoing messages. If it were only for sending messages, you could omit the process variables and retrieve the values directly from the business objects. However, more importantly, the process variables will be used to receive message content from incoming messages.

When done, your data definitions for the loan application pool should look as follows:

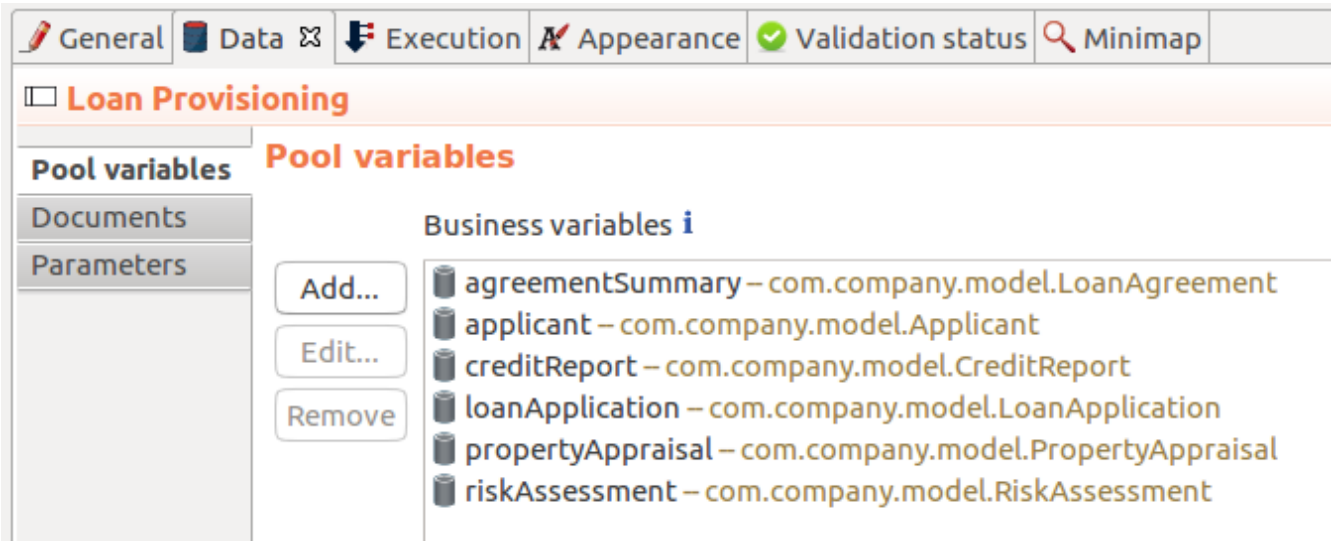


You will also need to model data for the loan provisioning pool. The loan provider also works with applicant and loan application data, so this pool requires the same business and process variables as the applicant pool. Additionally, you know from Fig. 10.12 in "Fundamentals" that the loan provider also works with a risk assessment data object, an agreement summary data object, a credit report data object, and a property appraisal data object. As these data objects do not need to be transferred by message passing from or to another pool, you do not require process variables to hold their identifiers.

- Create business and process variables for the loan provisioning pool using the same procedure as above.

Again, you do not need to provide default/initial value expressions or instantiation.

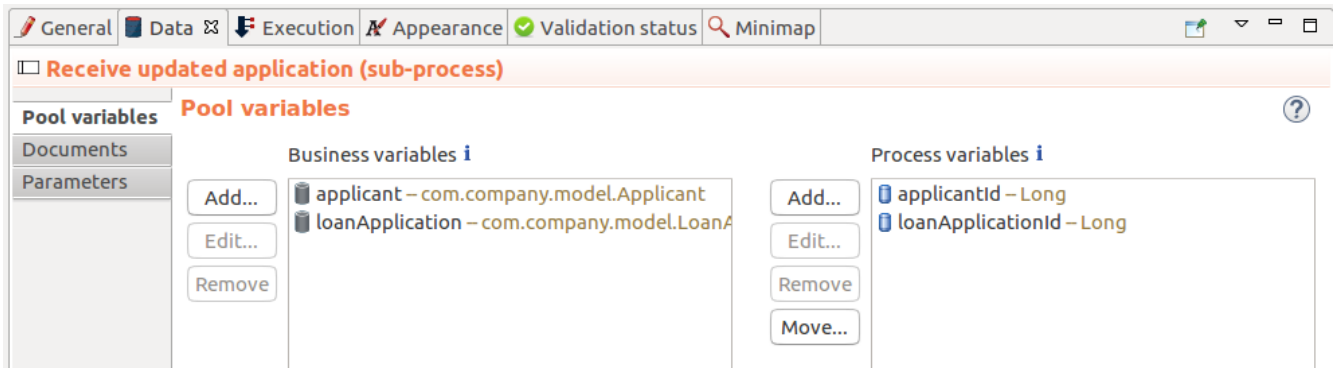
When done, the business variables for the loan provisioning pool should look as in the following figure. Additionally, you should have the *applicantId* and *loanApplicationId* as for the loan application pool.



You also need to model data for the loan provider (sub-process) pool. The activity in this pool requires access to the loan application and the applicant data.

- Model this data in the same way as you have done for the applicant pool.

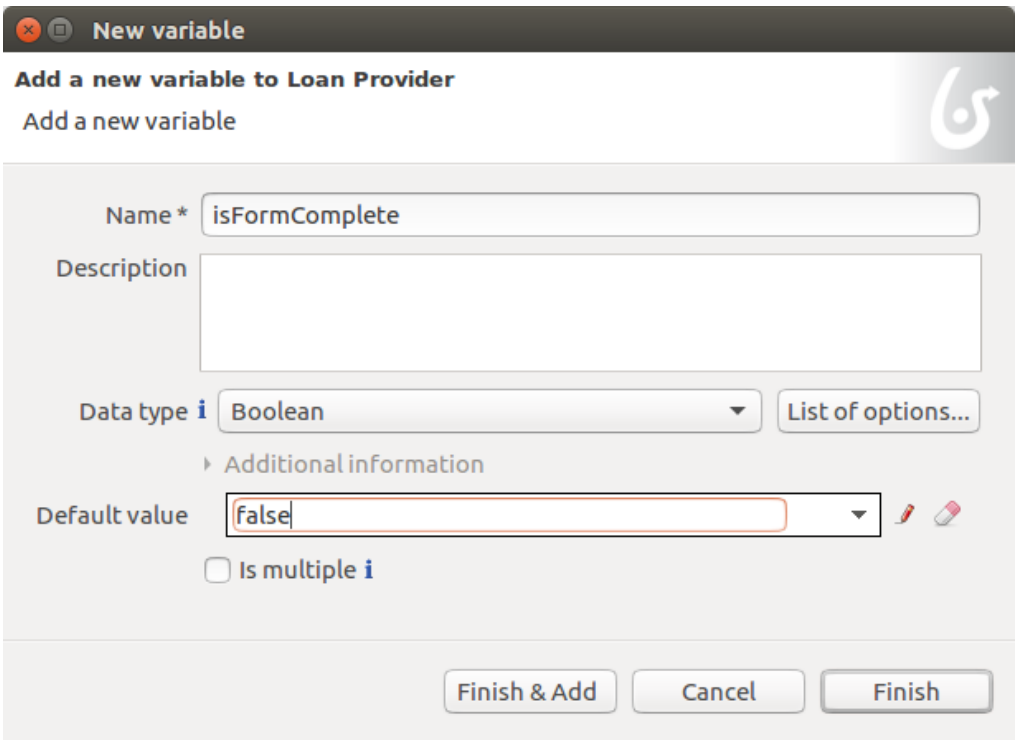
Your data definitions for the loan provider (sub-process) pool should look as follows:



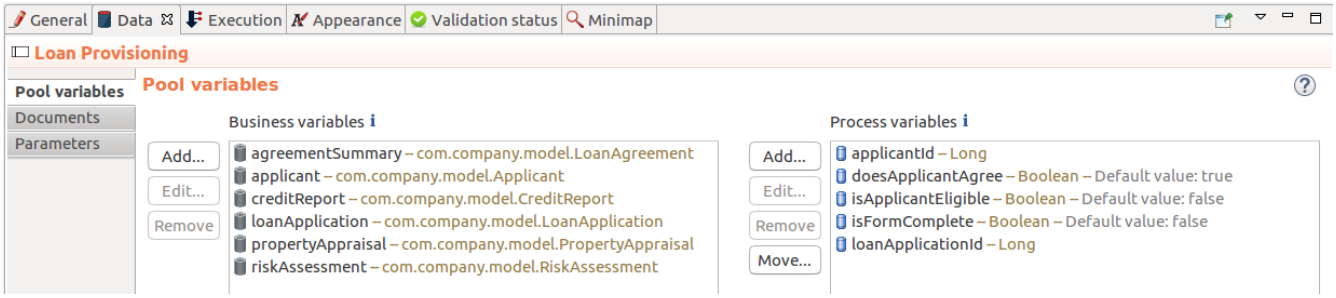
Next, note that the decisions for the exclusive (XOR) branching gateways require decision information. There are three such gateways in the process model. The first checks whether the form is complete, the second whether applicant is eligible and the third whether the applicant agrees with the repayment plan.

While the decisions can be based on data in the business variables, the model in Fig. 10.12 in "Fundamentals" contains explicit tasks to create this information ("check application form completeness", "assess eligibility", and "verify repayment agreement"). Therefore, you will model three boolean (true/false) variables that capture this information as process variables for the loan provisioning pool. The following image shows the dialog box for defining one of these variables.

- Define all three variables in the same way.

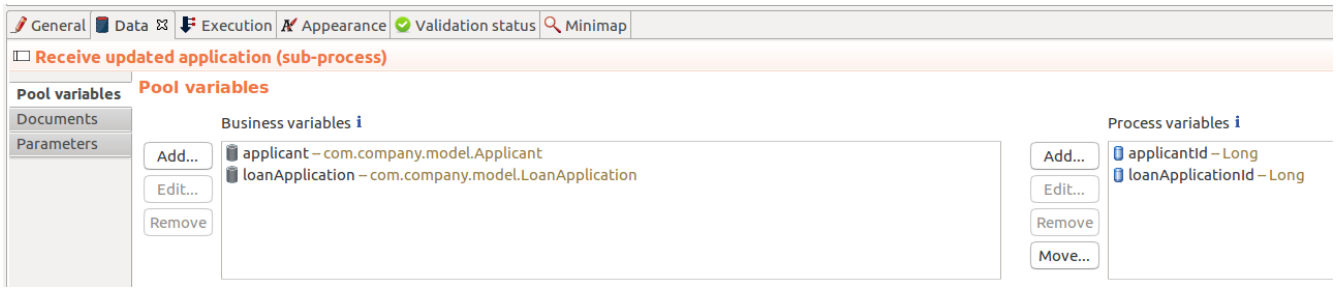


When done, your data definitions for the pool should like as follows. Note that the process variables also include the loan application identifier and the applicant identifier you have defined to be used for message passing.



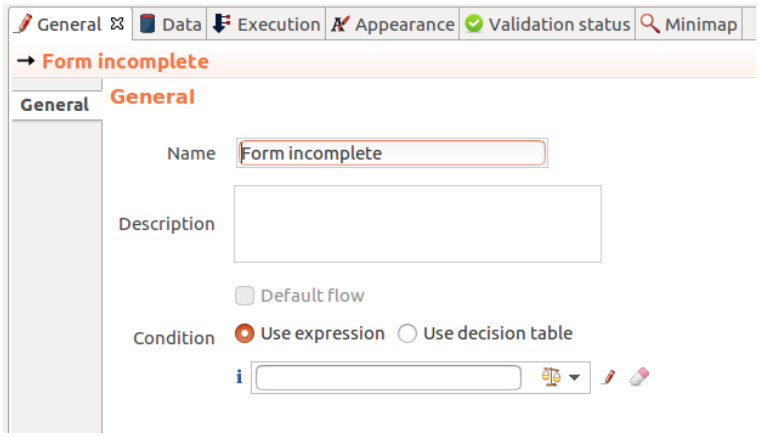
Finally, add business variables and process variables to the "Receive updated application (sub-process)" pool. This sub-process works with the applicant and loan application information, so you model these as business variables. As above, you do not need to provide instantiation and initial value expressions. You also model the *applicantId* and *loanApplicationId* as process variables of type long. Note that you do not need to provide initial values for the the applicant and loan application. The process pool will receive these from the inbound message.

- Create data definitions for the sub-process pool to look as follows:

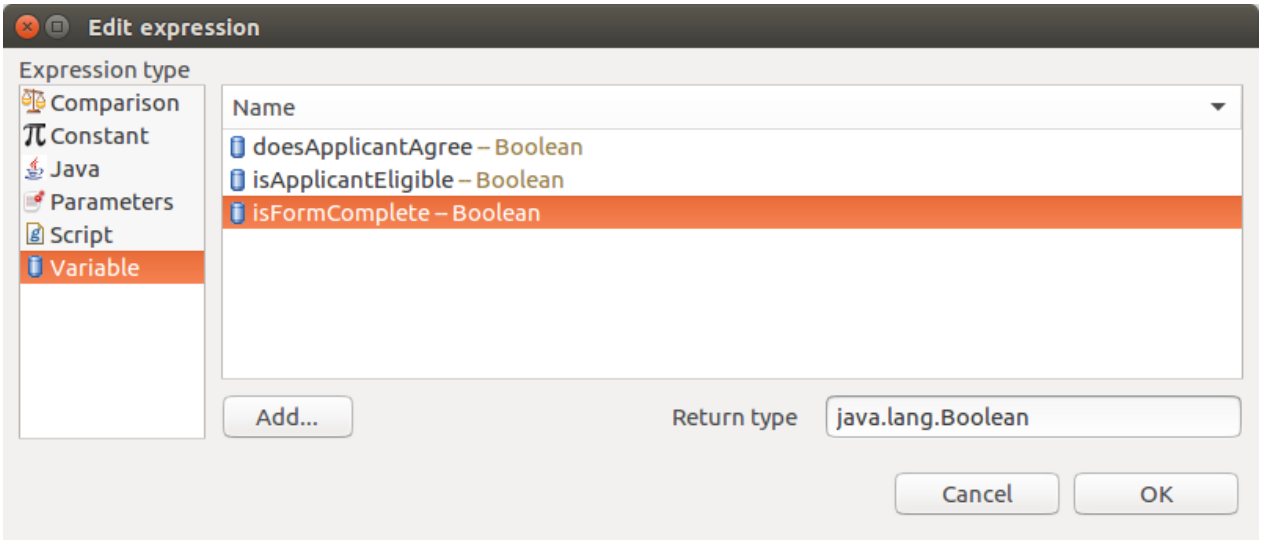


Branching Conditions

With the pool data, in particular the process variables, defined, you can specify the flow conditions from each of the branching exclusive (XOR) gateways. Select a non-default (conditional) flow. The properties pane below the diagram allows you to enter a condition expression that determines when the process will follow this flow.



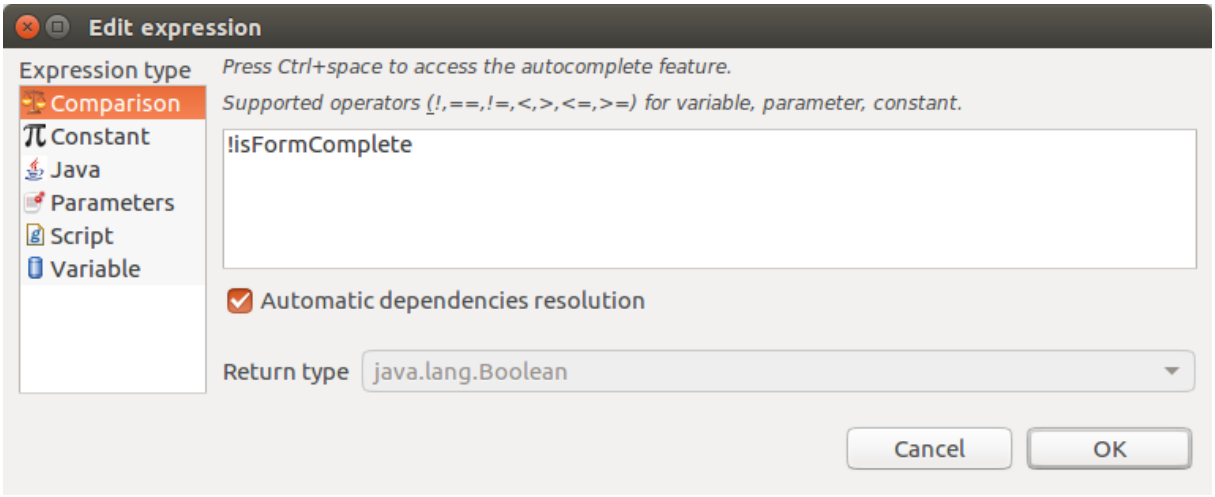
- Use the pencil icon to edit the expression.
- In the following dialog box (next page), select “Variable” in the list of expression types on the left. Select one of the three boolean process variables you have just created, but do not push the “OK” button yet.



- Select the “Comparison” expression type in the list on the left.

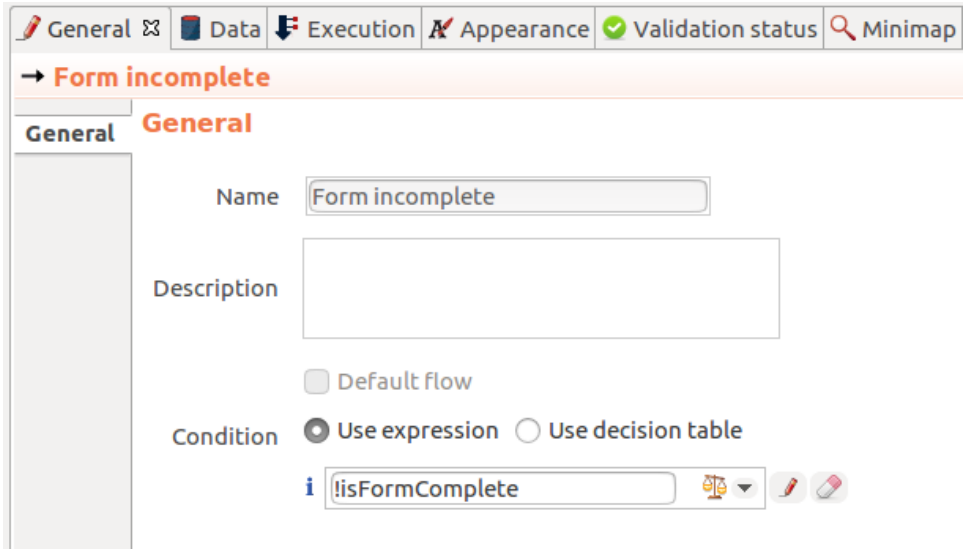
You should see the boolean variable you have selected is now entered.

- Change the comparison expression by adding an exclamation mark (“!”) before the variable name (as this is the flow that we wish to follow when the form is NOT complete).



- Push the “OK” button.

Your expression should be reflected in the properties pane for this flow:



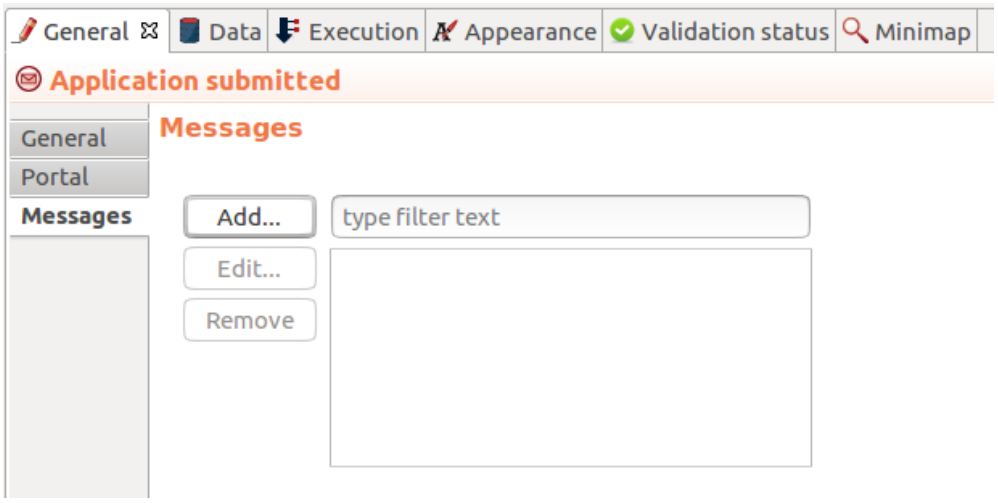
- Do the same for the conditional (non-default) flows from the other two branching exclusive (XOR) gateways.

Messages

Now that you have specified business and process variables for all pools, you can define messages and their contents. You will begin with the message that is sent from the loan application pool to the loan provisioning pool to initiate the loan provisioning.

- Select the end event "Application submitted" in the loan application pool.
- Select the "Messages" tab in the "General" properties pane.

This is where you will define the message that is being sent at this event:

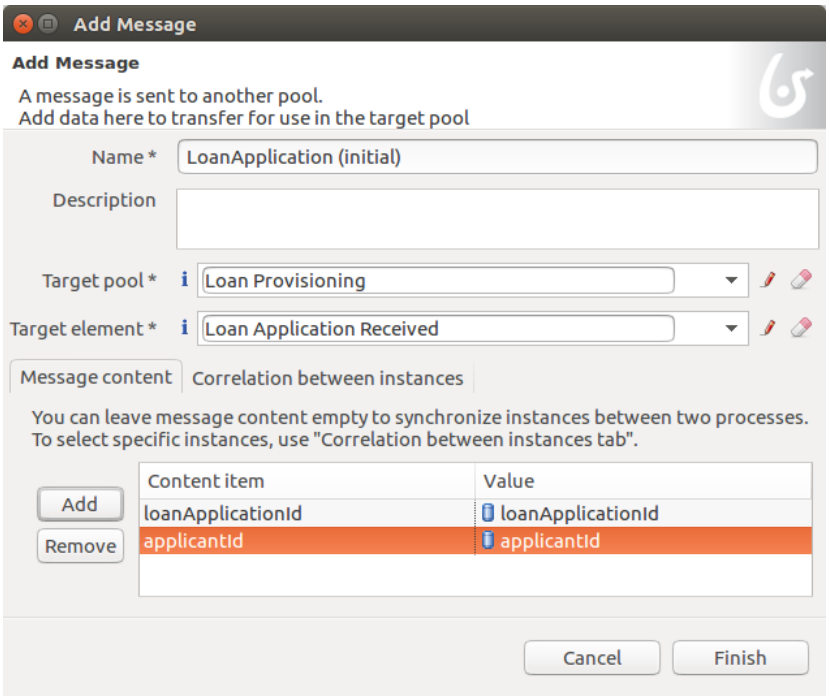


- Push the "Add ..." button to define a new message.

- Specify “Loan Application (initial)” for the message name.
- Select “Loan Provisioning” as the target pool from the drop-down list.
- Select “Loan Application Received” as the target element from the drop-down list.

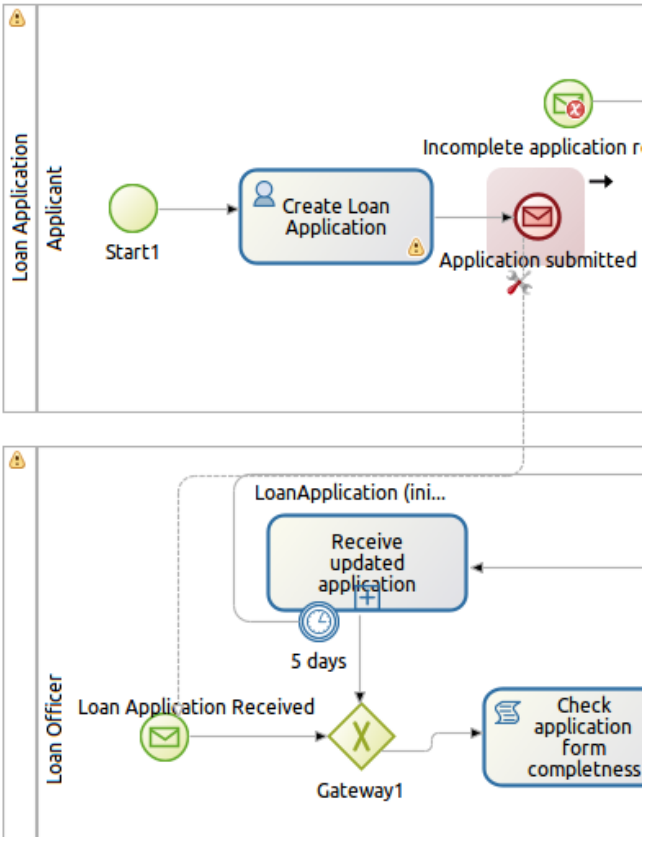
You will need to transfer the *loanApplicationId* and the *applicantId* as message content.

- Push the “Add” button in the message content pane.
- Enter “loanApplicationId” as the content item.
- Select the value field and begin typing “loanApplication”. Bonita will present you with suggestions that match what you have typed. Select the *loanApplicationId* process variable from the list of suggestions.
- Do the same for the applicant ID. Your message definition should look as follows:



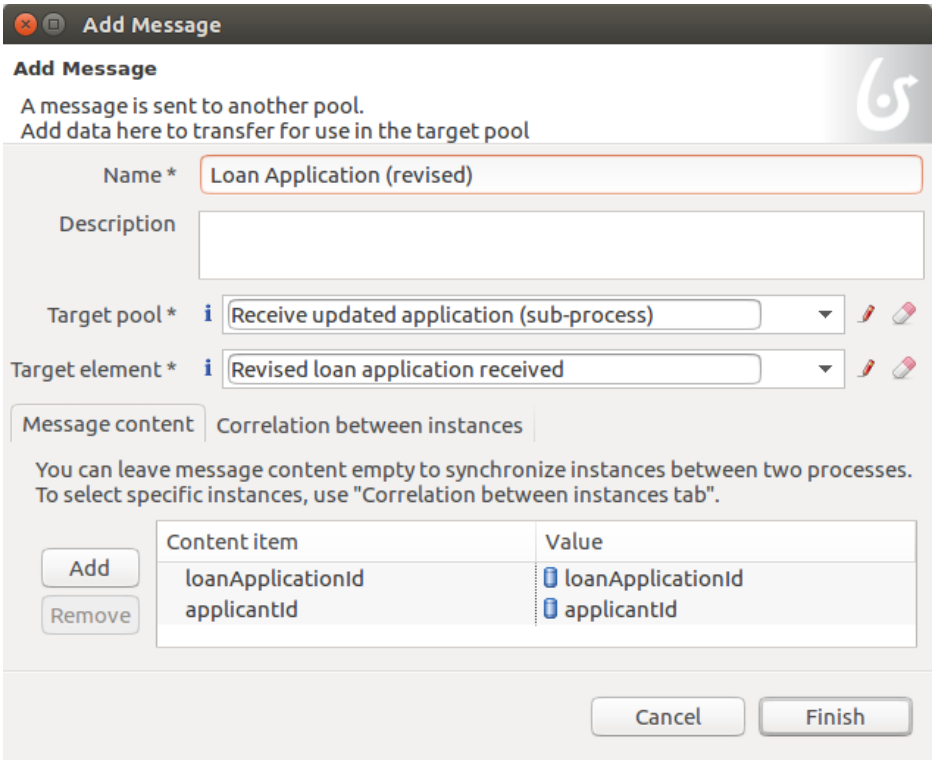
- Press the “Finish” button.

Your message should now be reflected in the process diagram by a message flow between the two events.



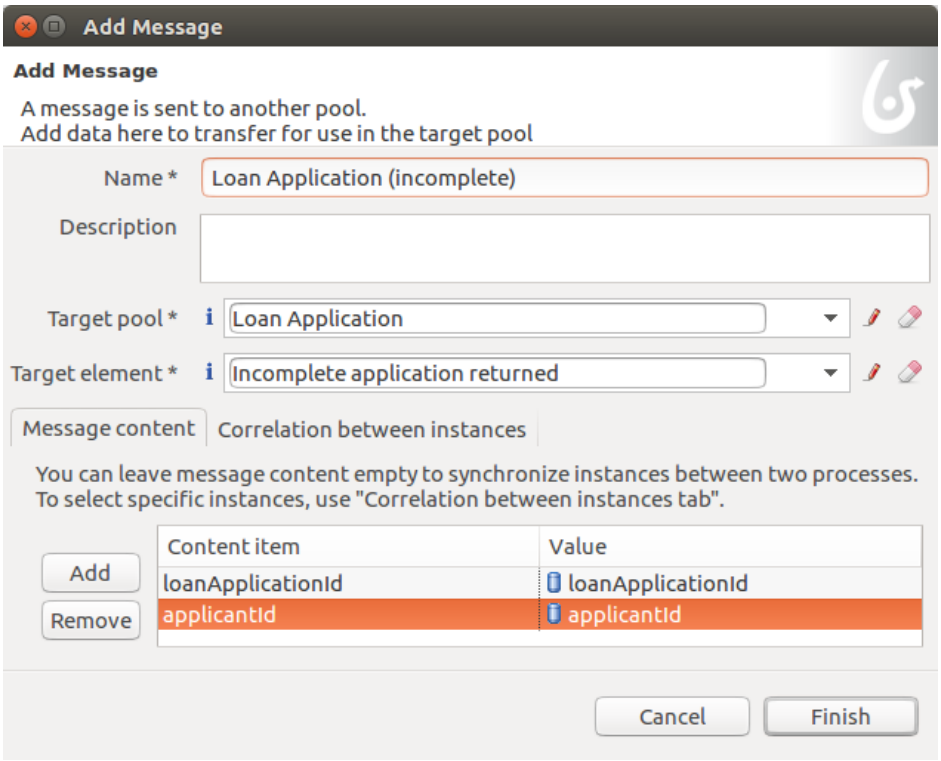
Next, you will define the message that is sent from the "Revised application submitted" end event in the loan application pool.

- Define the message as in the following figure (next page).

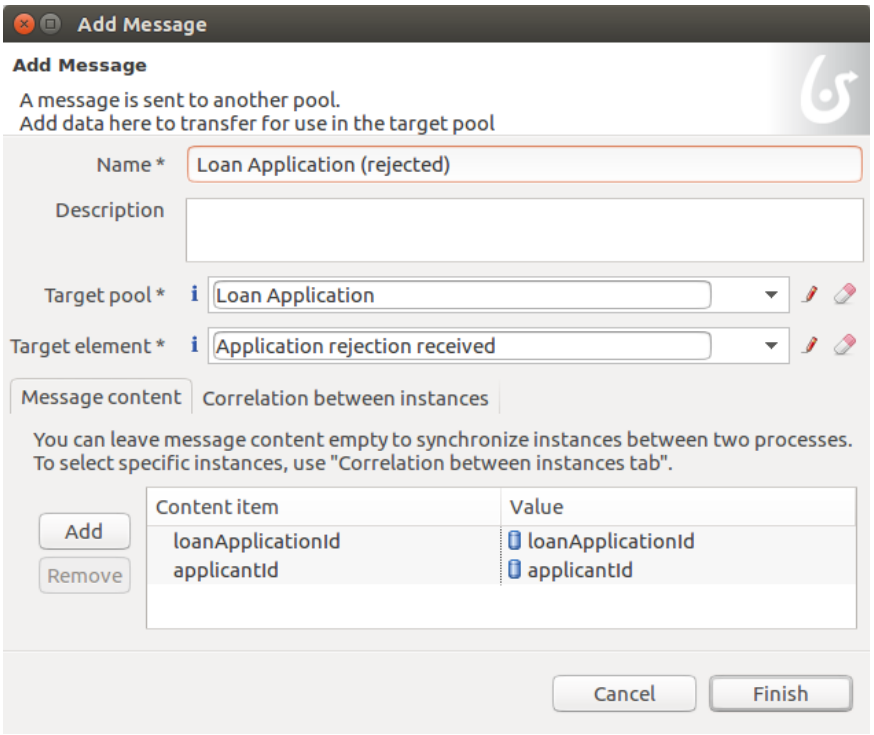


Next, define the message that is being sent from the message activity “Return application to applicant”.

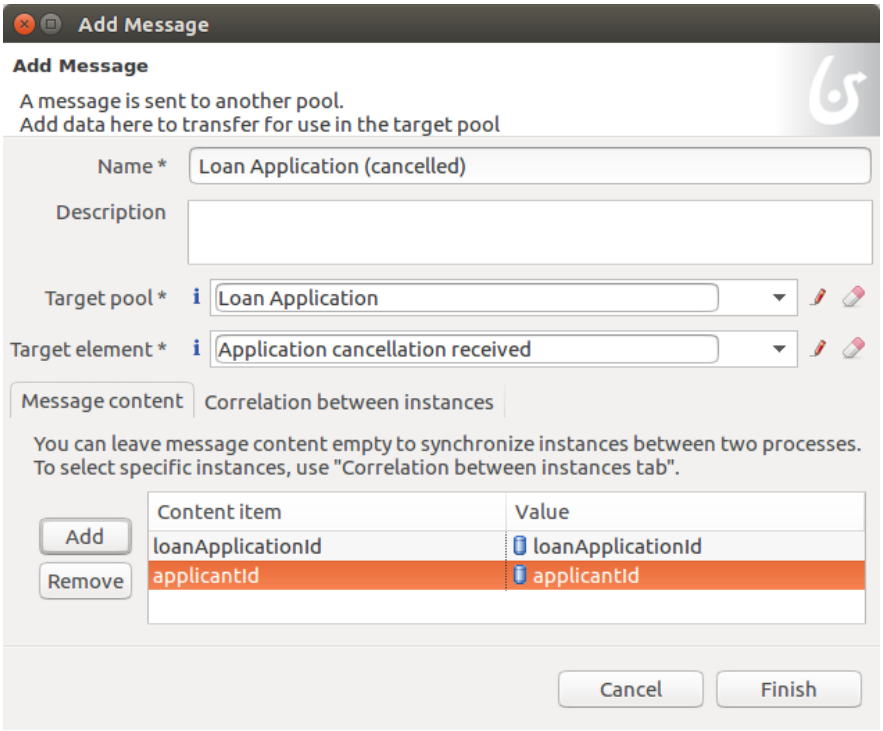
- Define this message as follows



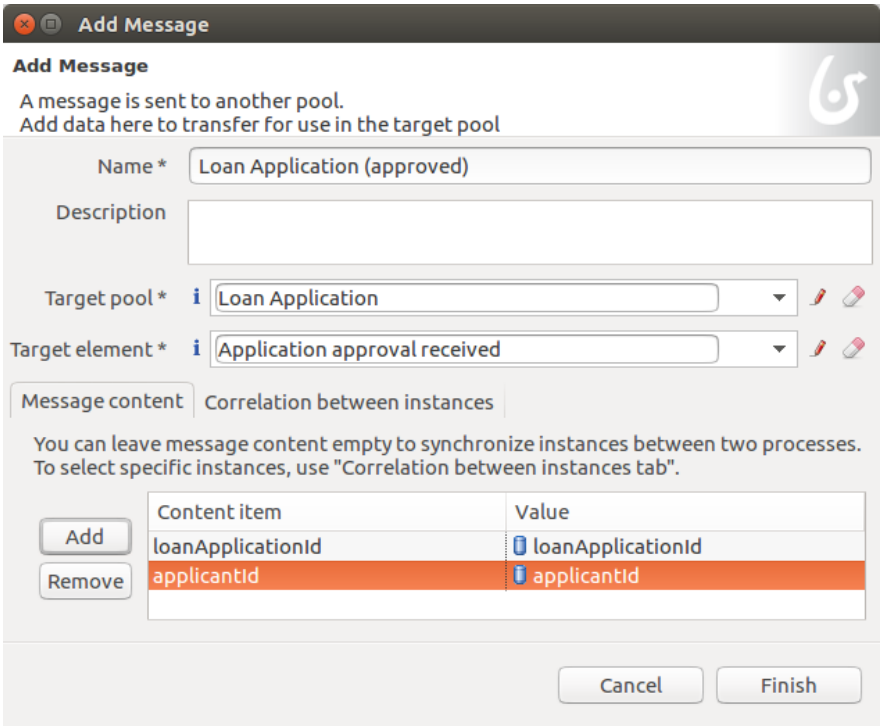
- Define the message sent from the message sending activity “Reject application” as follows:



- Define the message from the message sending activity “Notify cancellation” as follows:



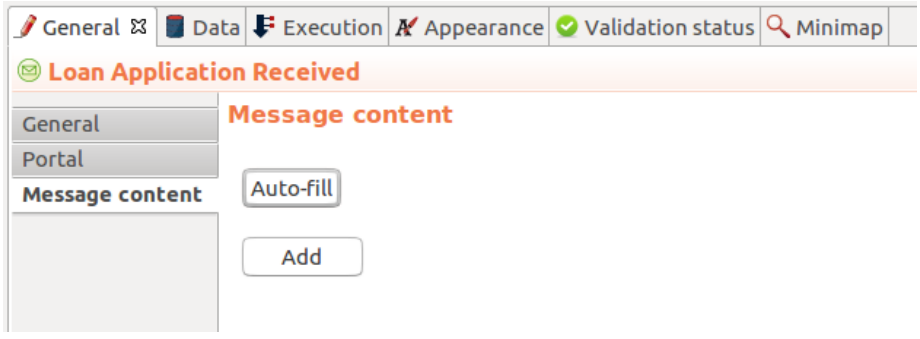
- Finally, define the message sent form the message sending activity “Notify approval” as follows:



Message Handling

You have specified the messages that are being sent. You must now specify what happens when messages are received. First, upon receipt of a message, you must specify what happens to the message content. Second, because our messages do not transfer complete business objects, you must add activities that restore/read the business object from the database, given its identifier. You will begin with the “Loan Application Received” event in the loan provisioning pool.

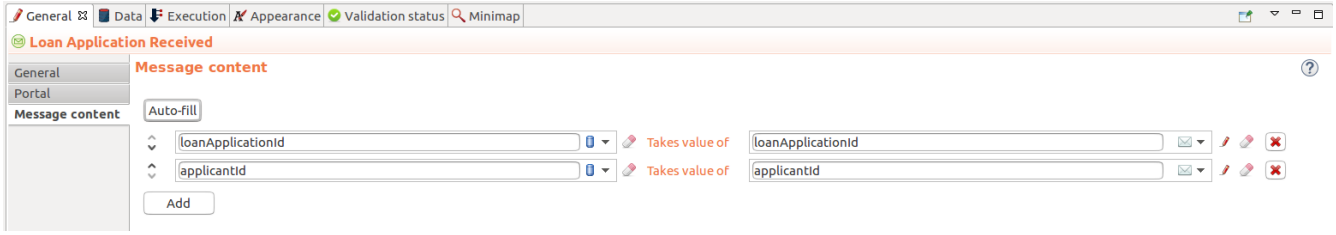
- Select the “Loan Application Received” message event, then select the message content pane in the general tab of the properties panel:



- Push the “Auto-fill” button.

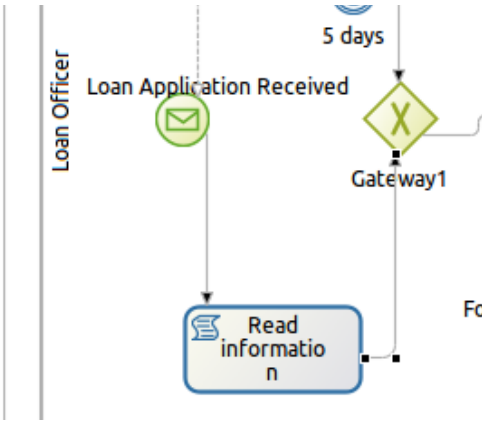
Bonita will attempt to map message content to business or process variables based on names and datatypes. As you have named your message content elements the same as your process variables, this

automatic mapping works for you. In other cases, use the "Add" button to manually create these mappings. The generated mappings should look as follows:

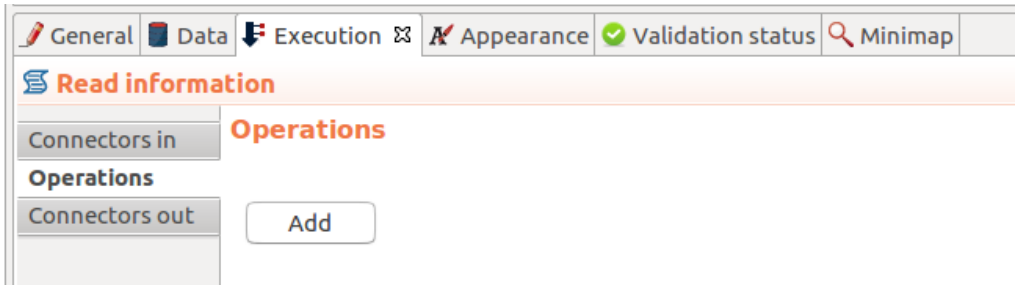


Next, you will need to read/restore the full business object of applicant and loan application after receipt of the message.

- Insert a script task named "Read information" after the message start event, as follows:.



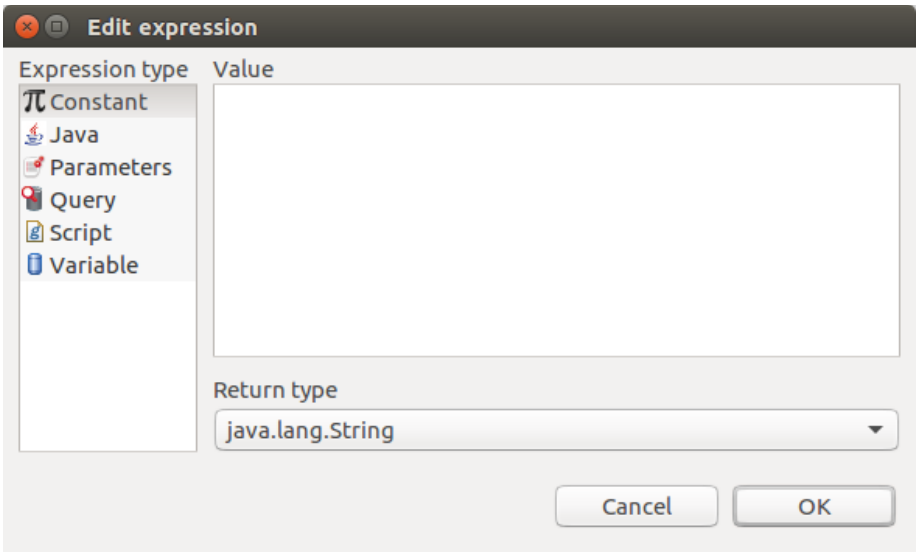
- Select this task, then select the "Execution" tab in the properties pane, then select the "Operations" pane:



Operations in Bonita are data variable assignments that are carried out after a task has completed execution. In this case, you will specify how to restore the applicant and loan application business objects.

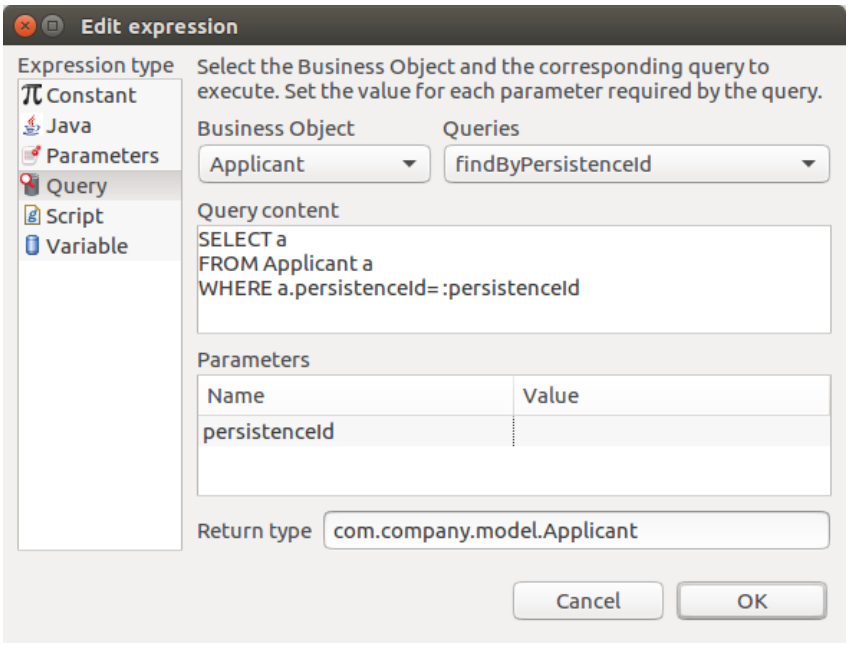
- Push the "Add" button.
- Select the "Applicant" business variable as the target of the operation.
- Select the edit button (pencil icon) for the expression on the right.

You should see the following dialog window:



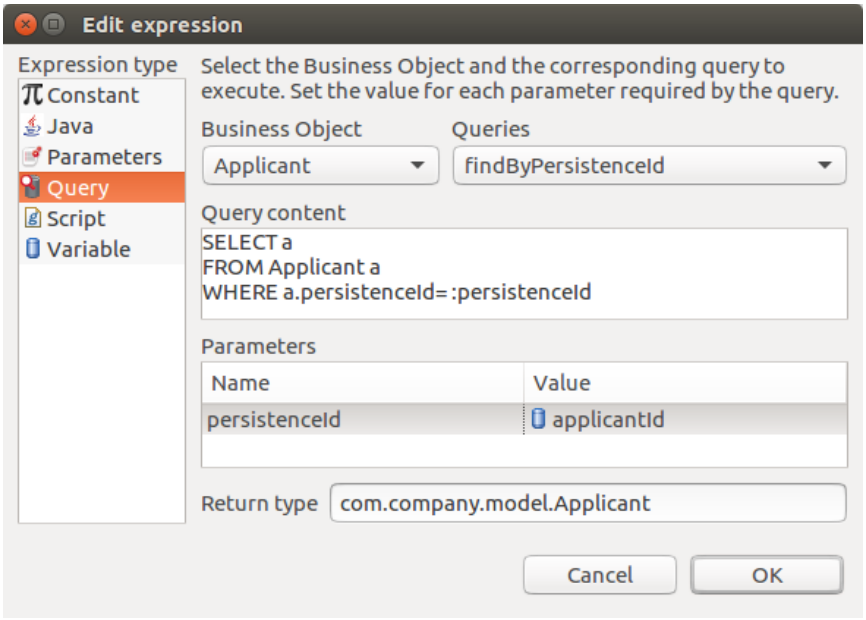
- Select “Query” for the expression type on the left.

Bonita will automatically propose a database query to read the applicant information from the database based on its *persistenceId*.



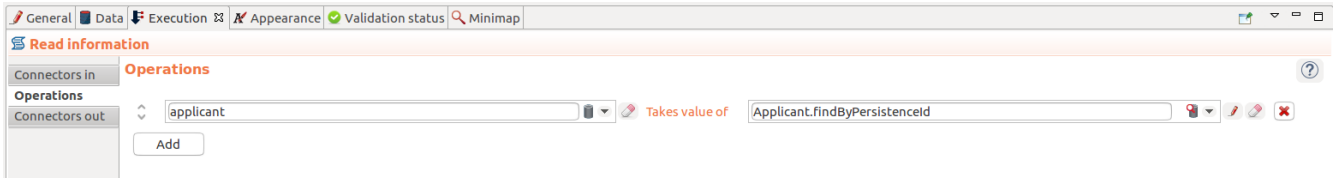
To execute this script, you need to provide where the persistence Id script parameter comes from.

- Select the “Value” column for the *persistenceId* parameter.
- Set this value to the *applicantId* process variable

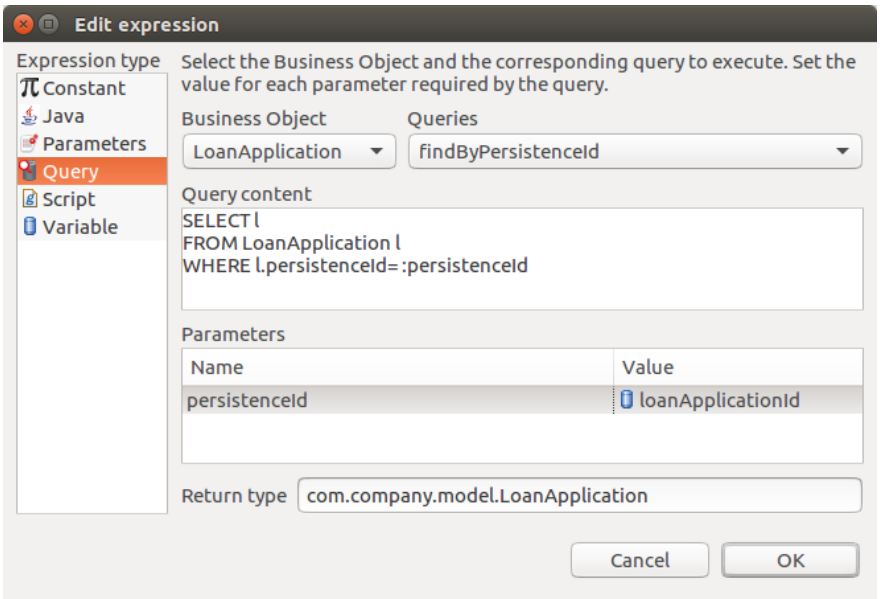


- Push the "OK" button.

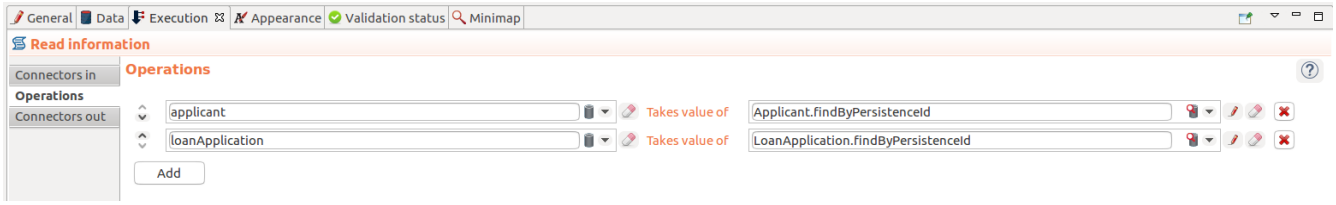
The query expression you have specified should be reflected in the list of operations for the activity:



- Do the same to retrieve the information of the loan application. The query specification should look as follows:



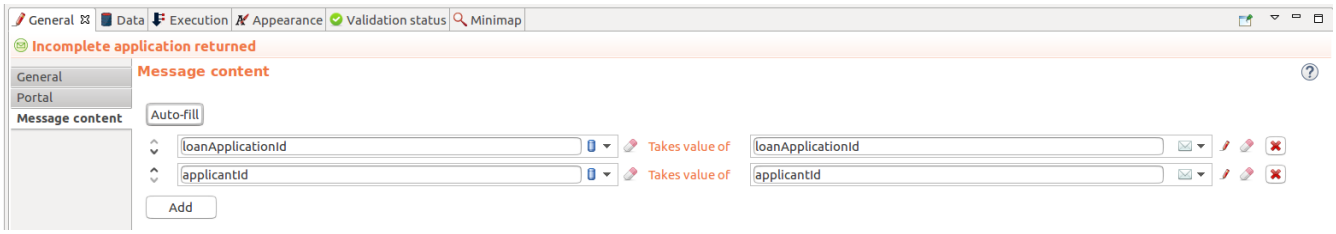
The full list of operations for the task “Read information” should look as follows:



Next, you will specify the message handling for the receive message start event “Incomplete application returned” in the loan application pool.

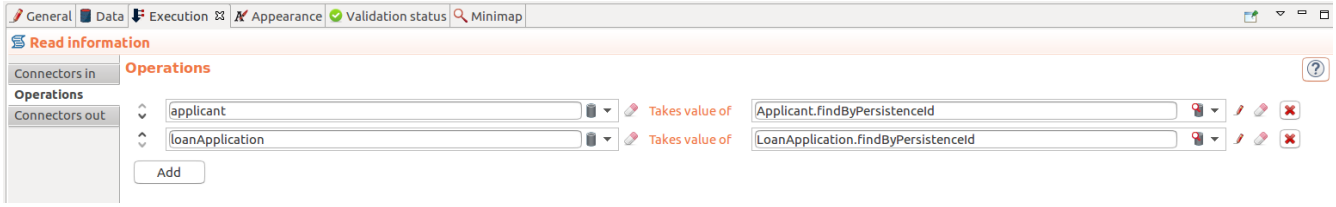
- Select the “Incomplete application returned” event, then select the “Message content” panel in the “General” tab of the properties pane, and push the “Auto fill” button.

The message content should be mapped to process variables as follows:



- Include an activity to read the information from the database in the loan application pool, after the receiving message start event “receive incomplete application”

- Specify the operations for the task “Read information” in the same way as earlier:



- Perform the automatic mapping of message content to process variables for the remaining three receive message start events in the loan application pool, “application rejection received,” “application cancellation received,” and “application approval received”.

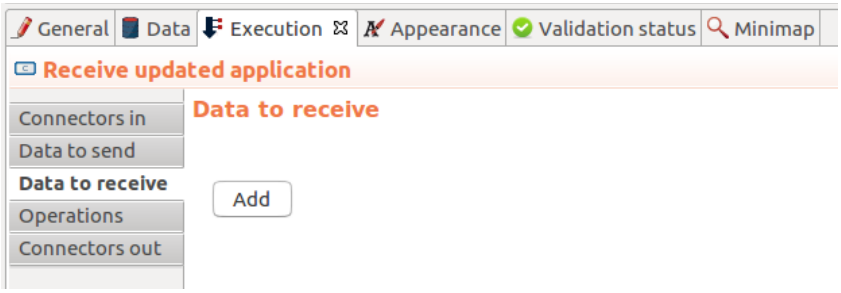
Because there are no further process activities specified following these message events, you need not bother reading the updated information from the database.

The intermediate message receive event “revised loan application received” in the receive updated application (sub-process) pool also requires a mapping of message content to process variables.

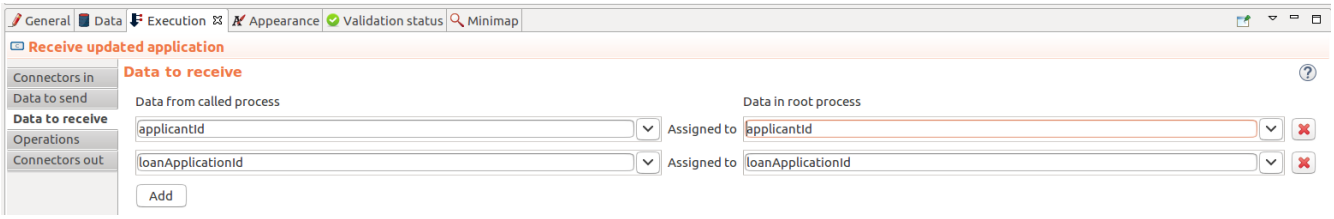
- Use the auto-fill feature to perform this mapping.

However, rather than reading the information from the database in the sub-process and then passing it back to the parent activity, you will pass only the applicant Id and loan application Id back to the parent process and will perform the data updates after the call activity task in the loan provisioning pool is complete.

- Select the "Receive updated application" call activity.
- Select the "Execution" tab of the properties panel below the drawing area, then select the "Data to receive" pane

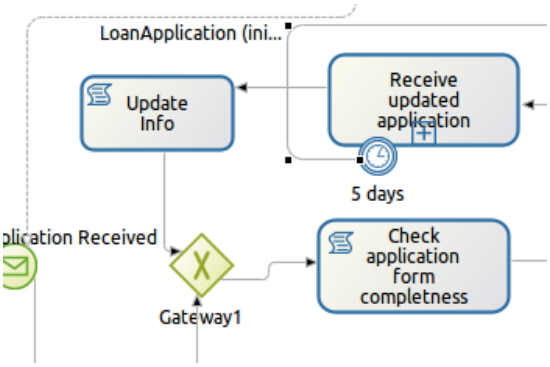


- Press the "Add" button to define the data that the call activity receives from the specified sub-process.
- Map the *applicantId* and *loanApplicationId* of the parent and sub-process as follows:



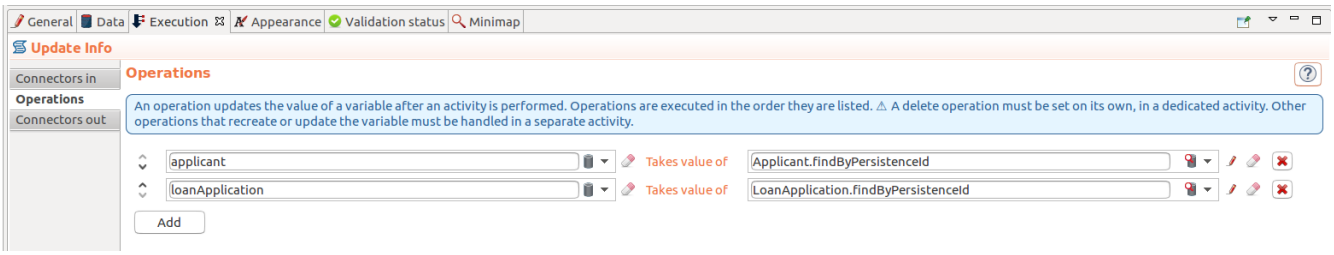
Next, add a task after the call activity that updates the business variables from the database using the *applicantId* and *loanApplicationId* that is received from the sub-process.

- Insert a script task called "Update info" immediately after the call activity



- Specify the operations for this task as database queries as you have done for the "Read information" task earlier.

The operations should be defined as follows:

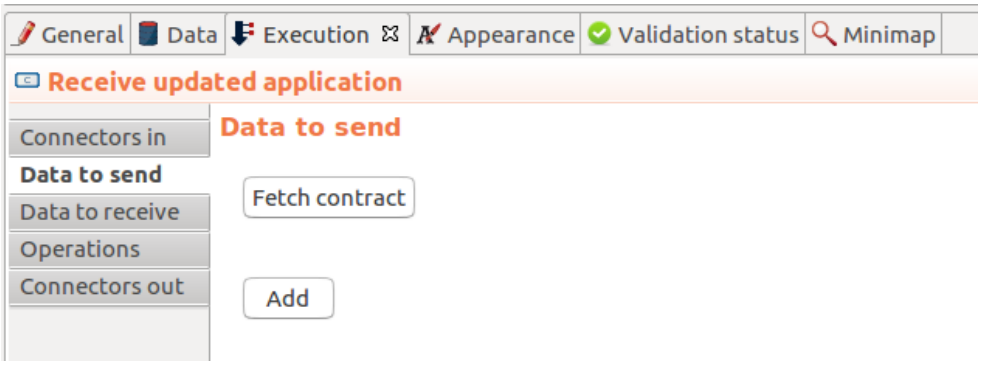


Message Correlations

An important consideration in a collaboration of multiple processes is to ensure that messages are sent to the right *instance* of a process. In a typical request/response pattern, as you have modelled here where the loan provisioning process (message "Loan Application (incomplete)") and the loan application process responds (message "Loan Application (revised)", to the "receive updated application (sub-process)" process), you need to ensure that the message is being sent to that instance of the receive updated application sub-process that was called from the instance of the loan provisioning process that sent the loan application (incomplete) message to the loan application process.

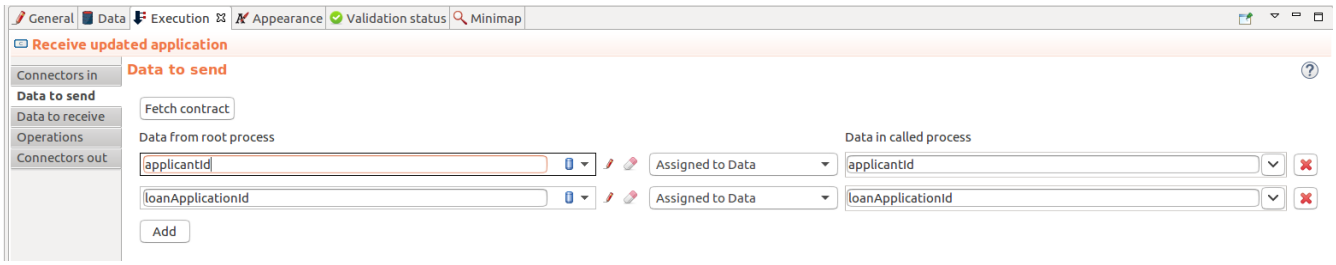
You will do this through message correlation, i.e by specifying information that uniquely characterizes each process instance. In your case, this information is the combination of *applicantId* and *loanApplicationId*. To enable message correlation, you first need to ensure that an instance of the "receive updated application" sub-process, upon instantiation, is provided with the *applicantId* and *loanApplicationId* of its parent process instance.

- Select the "receive updated application" call activity in the loan provisioning process.
- Select the "Execution" tab and the "Data to send" panel in the properties pane.



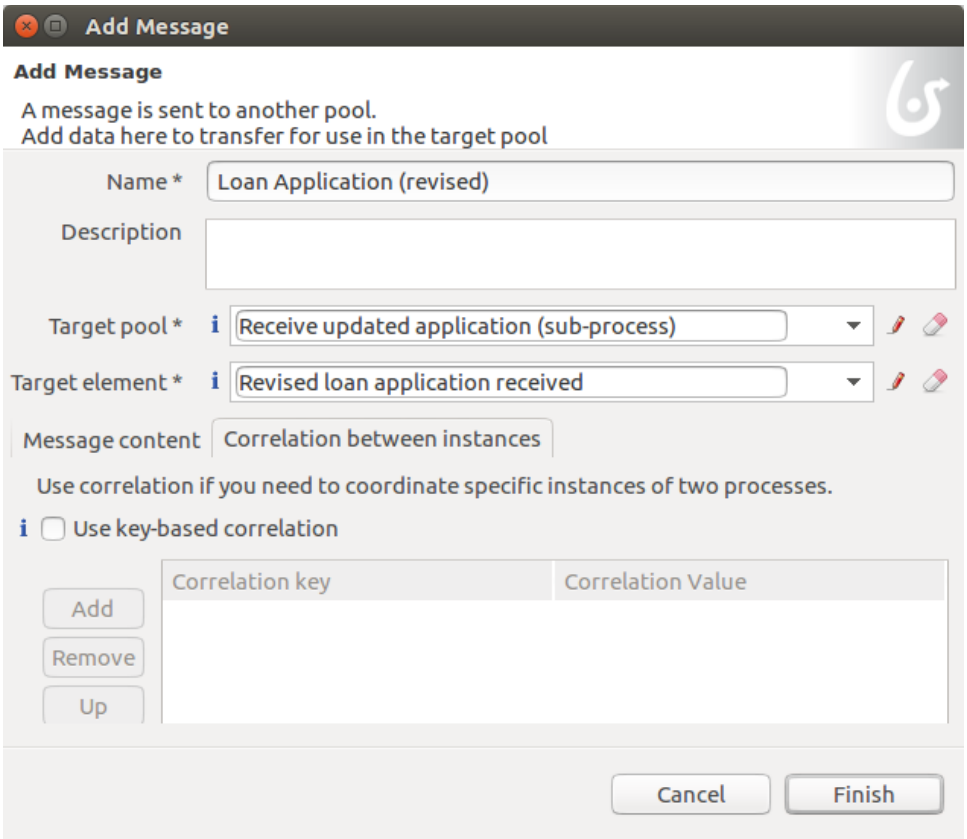
- Press the "Add" button to specify the data to be passed from the calling to the called process instance.
- Assign the *applicantId* of the root process to the *applicantId* in the called process. Make sure you select "Assigned to Data" as the type of data sending.
- Assign the *loanApplicationId* from the root process to the *loanApplicationId* in the called process.

Your data to send should look as follows:

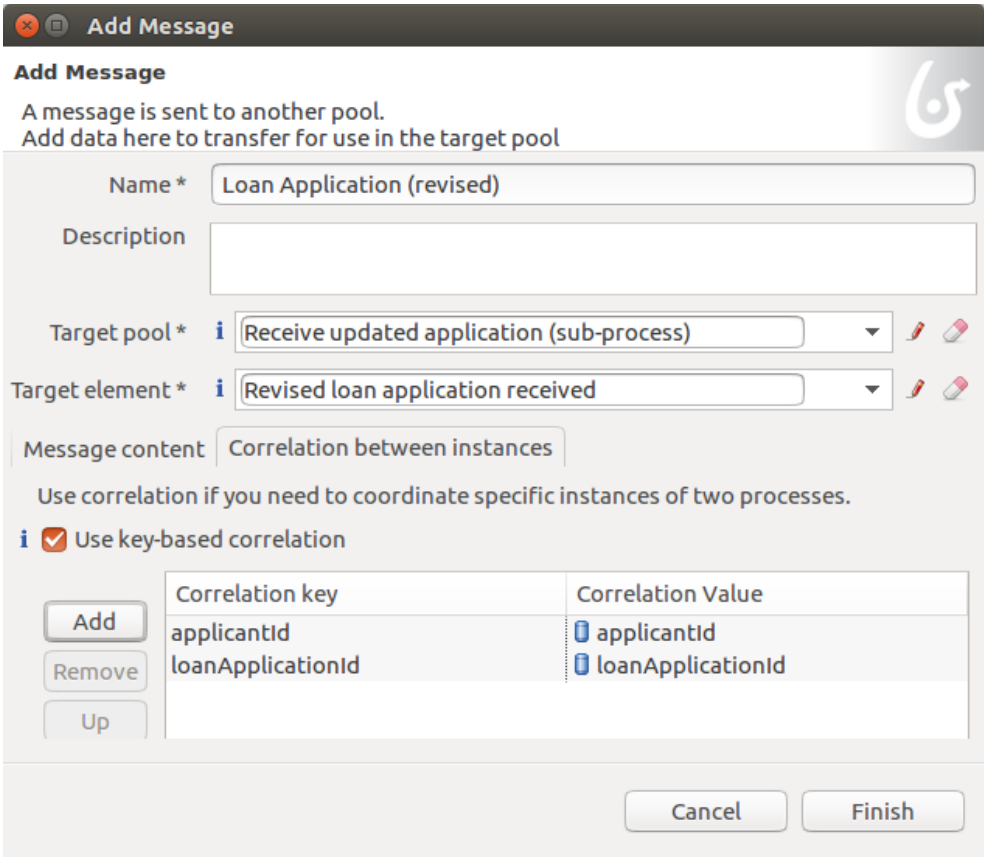


Next, revisit the message “Loan application (revised)” that is being sent from the “Revised application submitted” message end event in the loan application pool.

- Select that end event, then select the “Messages” pane in the “General” properties tab.
- Select the message and press the “Edit ...” button.
- In the following dialog window, select the “Correlation between instances” tab:



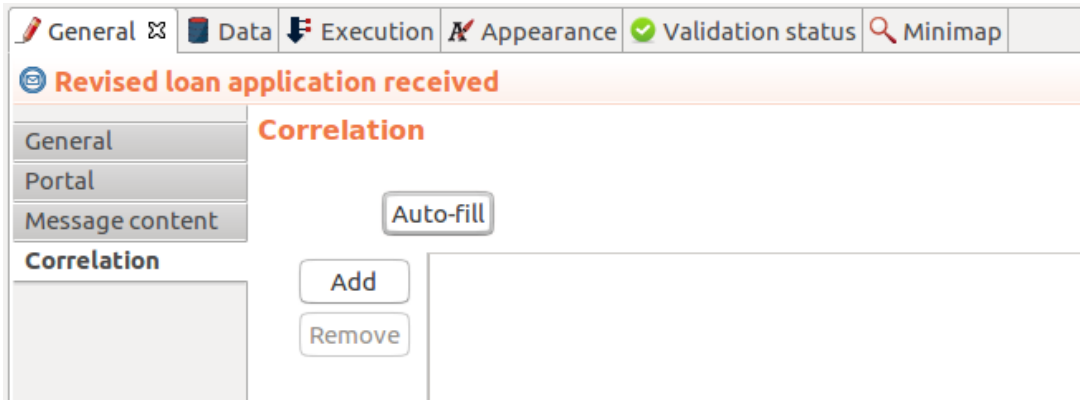
- Tick the checkbox “Use key-based correlation” and set up the two correlation keys as shown in the following figure (next page)



- Press the "Finish" button to complete the changes.

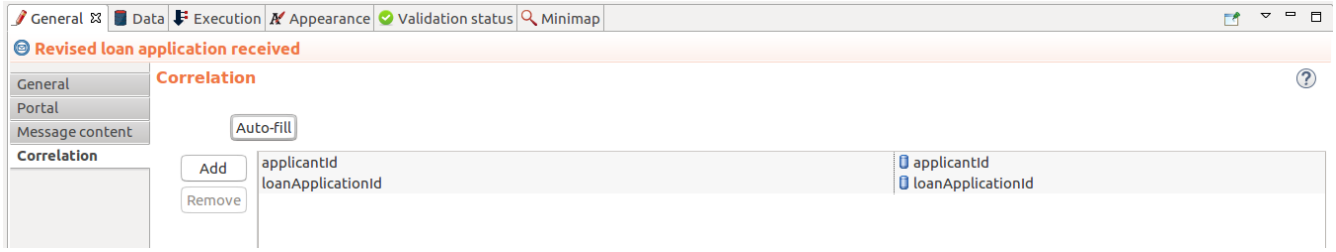
The corresponding message correlation keys also need to specified for the receive message event.

- Select the intermediate message receive event "Revised loan application received" in the receive updated loan application sub-process pool.
- In the "General" tab of the properties pane, select the "Correlation" panel.



- Use the "Auto-fill" button to have Bonita Studio try to derive message correlation information based on naming and data types of correlation keys and process variables.

Alternatively you can use the "Add" button to manually set up message correlation. Your correlations should map each correlation key to the process variable with the same name, as follows:



In summary, you have specified that upon sending the message, the two correlation keys are filled with the values of process variables of the sending instance, and the message is delivered to that instance of the receiving process where the values match the values of the respective process variables.

Comment: With the correlation keys set up in this way, actually sending the applicant Id and loan application Id as part of the message is redundant and the message content could be entirely omitted.

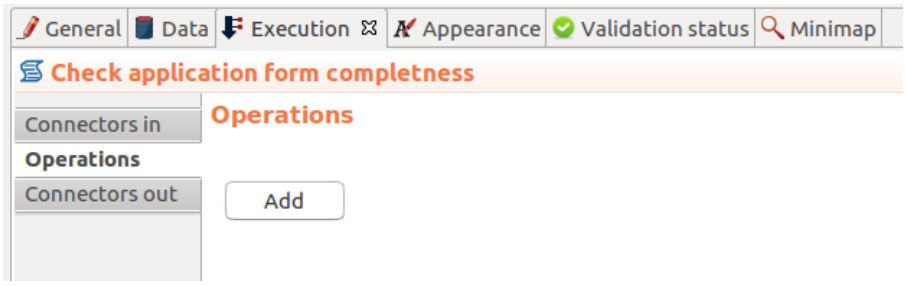
Comment: Similarly redundant is the data returned from the sub-process pool to the calling activity, which you have modelled as *applicantId* and *loanApplicationId*. As the "correct" sub-process instance is targeted by the message, so also the "correct" parent process will advance, which already contains the *applicantId* and *loanApplicationId*.

Comment: Note also that you do not need to set up message correlation for messages being sent to instances of the loan application pool. This is because the message receiving events are start events and therefore create another instance of the process. In fact, Bonita Studio does not allow message correlations to be used with message start events. If you had modelled the loan application process in greater detail, with proper flow and activities, these message start events would be intermediate message events and would therefore require correlation between process instances.

Script Task Operations

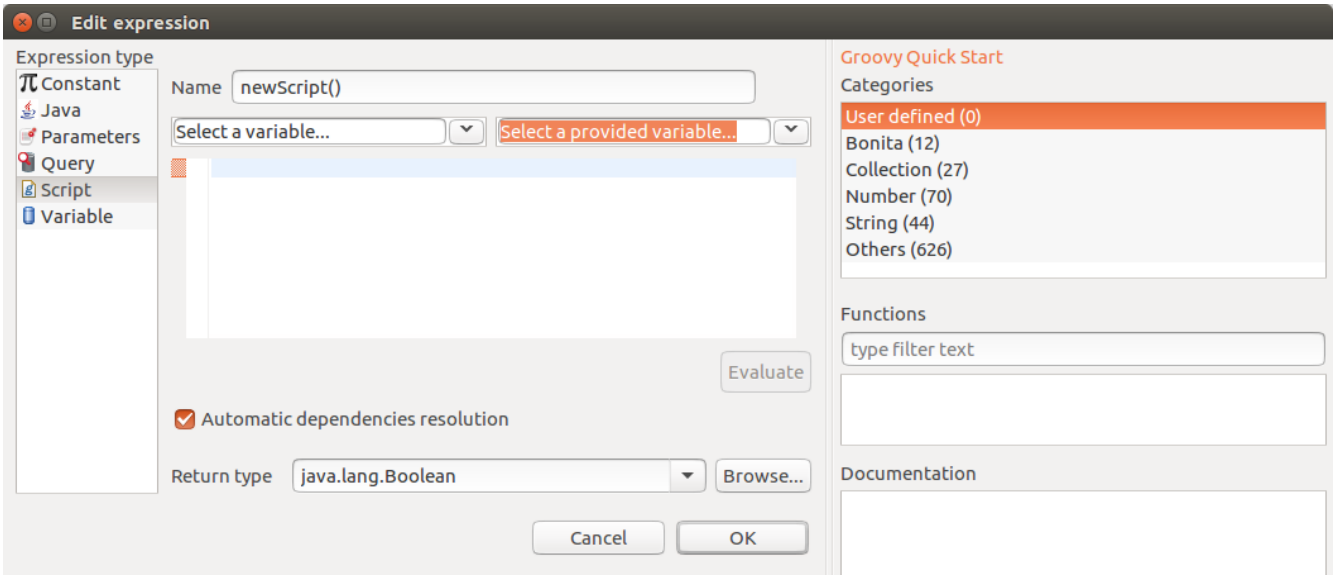
Your process model has two script tasks that need to provide information for subsequent flow decisions ("check application form completeness" and "check if insurance requested"). You have already modelled the flow conditions for the subsequent exclusive (XOR) gateways but so far have not specified where the variables for these conditions receive their values.

- Select the "check application form completeness" task, then select the "Operations" panel in the "Execution" tab of the properties pane.



An operation in Bonita is an assignment of values to variables after a task completes execution.

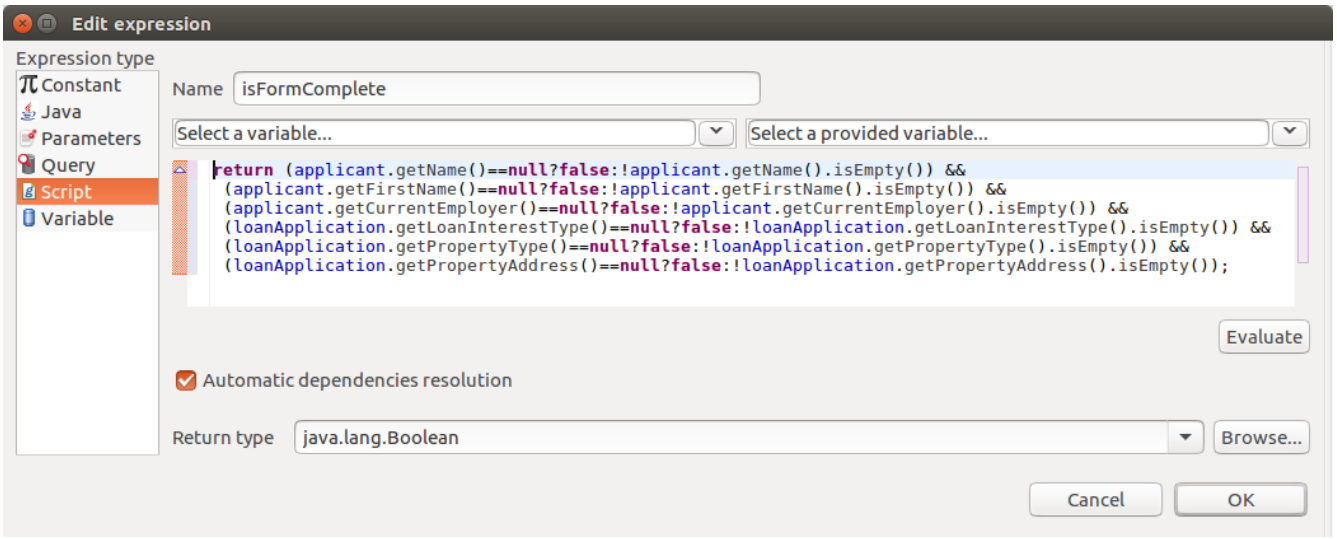
- Press the “Add” button to create a new operation.
- Set the target variable to be *isFormComplete.*
- For the expression on the right hand side, press the edit button (pencil symbol) to create the appropriate script that assigns a value to this variable.
- In the following dialog, select the script expression type in the list on the left.



- Name the script “isFormComplete”.

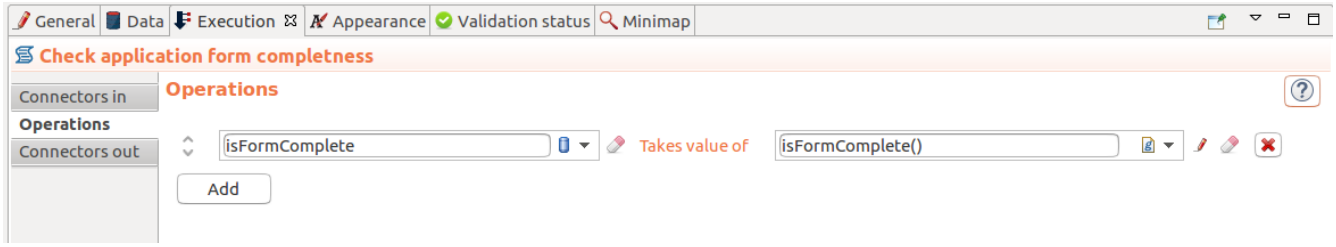
Next, specify the Groovy expression to use to evaluate whether the form is complete. You have access to all business and process variables in this script.

- Enter an expression like in the following image that checks whether the main attributes of applicant and loan application are present and not empty



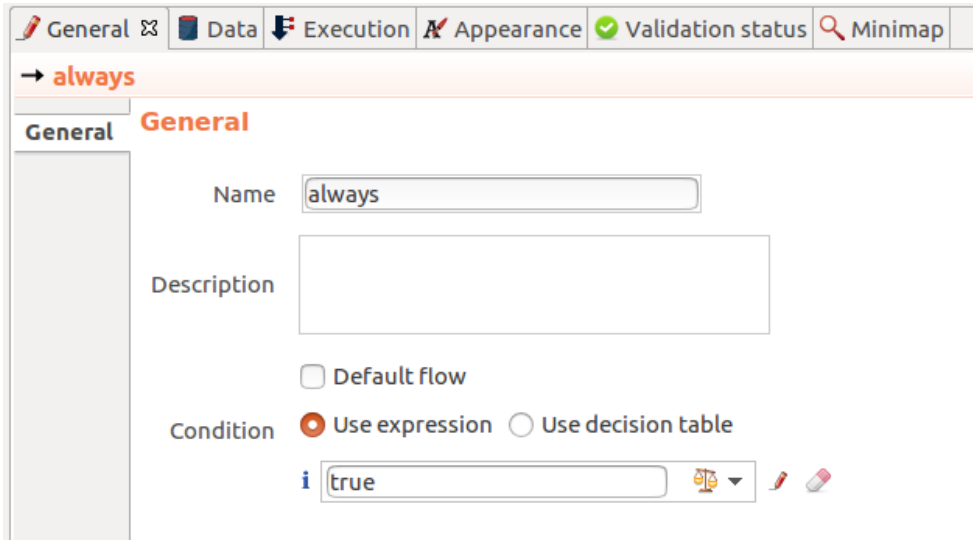
- Press the “OK” button to finish your expression.

Your expression should be reflected in the operation for the task:



Next, you will deal with the task “check if insurance is requested”. Note that you did not create a process variable to capture if insurance was requested, as this is directly contained in the loan application information. Rather than executing a script to fill a process variable, you will use the loan application information directly in the subsequent decision.

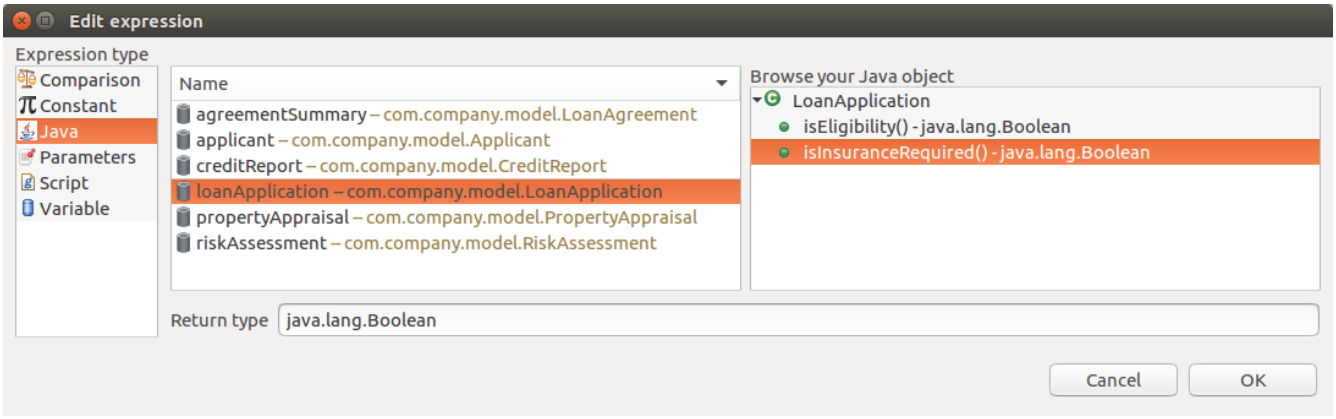
- Delete the script task “check if insurance is requested” and connect the task “prepare acceptance pack” directly to the following inclusive gateway.
- Select the flow from the inclusive gateway to the task “send acceptance pack”.
- In the “Properties” pane of the “General” tab, name this flow “always” and enter the condition “true”, as shown in the following figure.



- Select the flow from the inclusive gateway to the task “send home insurance quote”.
- Name this flow “isRequested” and select the edit button (pencil icon) to edit the condition.

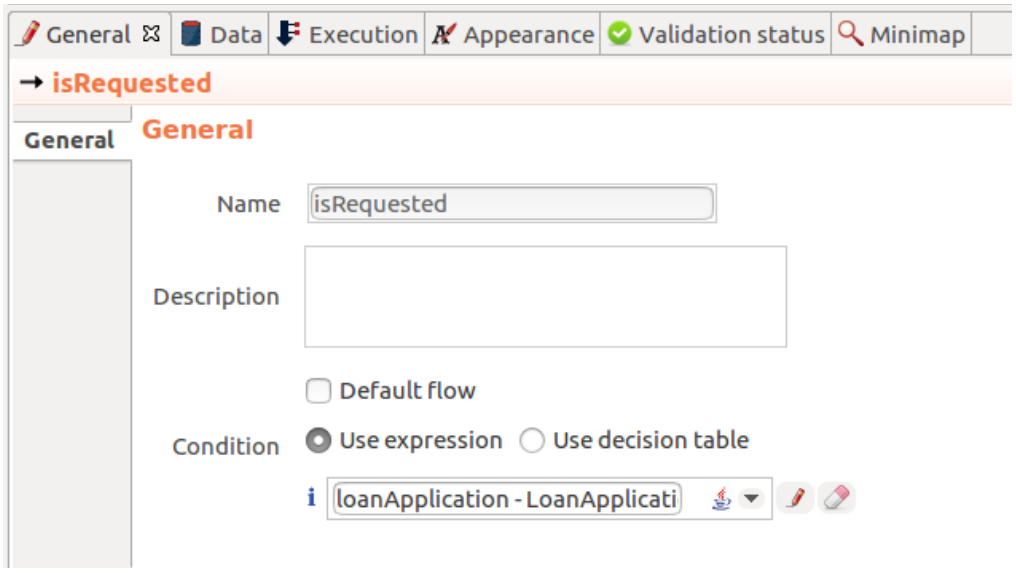
In the following dialog window:

- Select “Java” for the type of expression on the left.
- Select the “loanApplication” business variable in the center list.
- Select the method “isInsuranceRequired()” in the right list, as shown in the following figure.



- Press the "OK" button to finalize your expression.

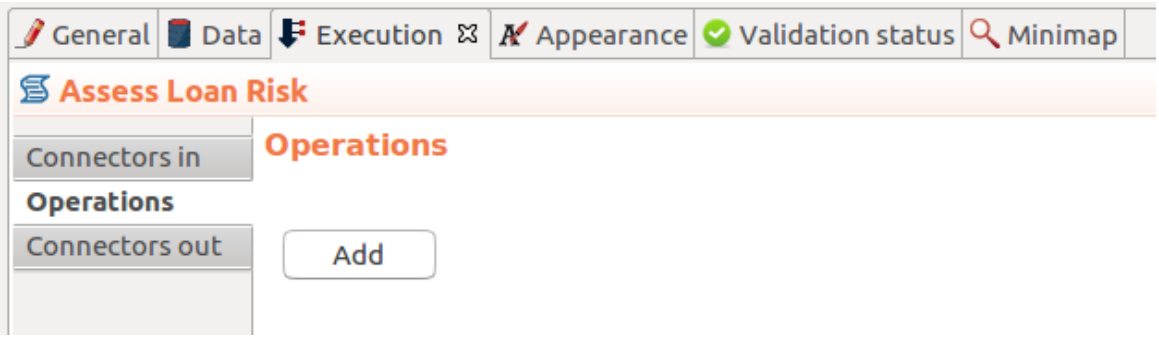
The expression should be reflected in the properties of the flow as follows:



There are two further exclusive (XOR) gateways with data-based decisions. You have already modelled the condition expressions for the outgoing flows. All are based on boolean process variables ("isApplicantEligible", "doesApplicantAgree"). However, the values for these variables cannot be automatically computed by script tasks but will be manually entered in the preceding activities when the process is executed.

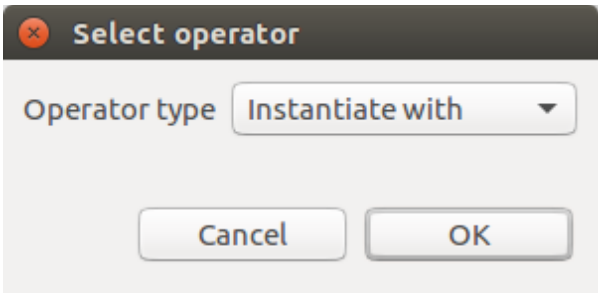
The task “Assess loan risk” is another script task. The purpose of this task is to use the credit history report to create a risk assessment object. Figure 10.12 in “Fundamentals” indicates that the task has access to a risk rules database, which you do not have for our simple tutorial. Instead, you will specify the risk rules directly in the form of a script.

- Select the task “Assess loan risk”.
- Then select the “Operations” panel in the “Execution” tab of the properties pane.



Operations in Bonita are executed after a task has completed execution to update values of business or process variables.

- Press the “Add” button to create a new operation.
- Select the risk assessment as the target of this operation.
- Click on the assignment operator to change it to “Instantiate with”.



Now create the script that instantiates a new risk assessment object and also provides the risk weight value for this object.

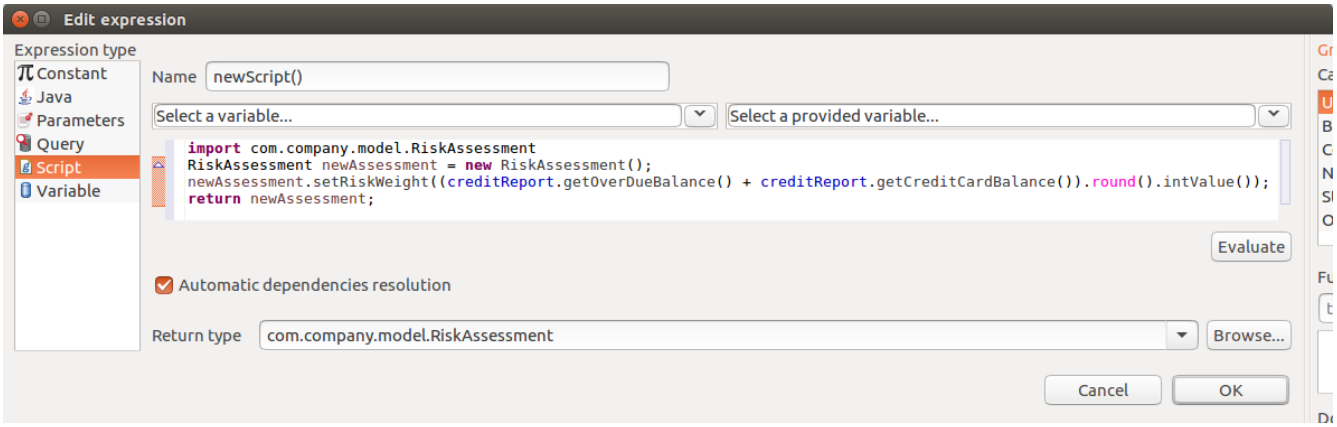
- Click on the edit button for the expression on the right (pencil icon).

In the following dialog box:

- Select “Script” as the expression type.
- Name the script “assessLoanRisk”.

The script has access to all business and process variables and you can define whatever credit risk evaluation rules you may want. As an example, the following expression adds the overdue balance and

the credit card balance from the credit history report, rounds the numbers as appropriate and converts them to an integer value.

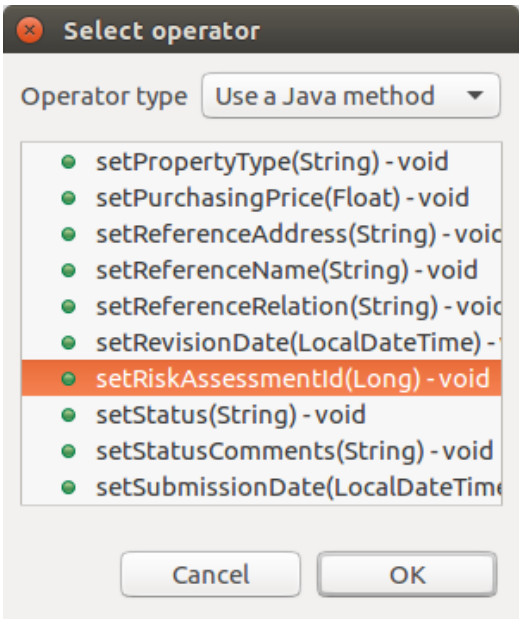


- Press the "OK" button to close the dialog window.

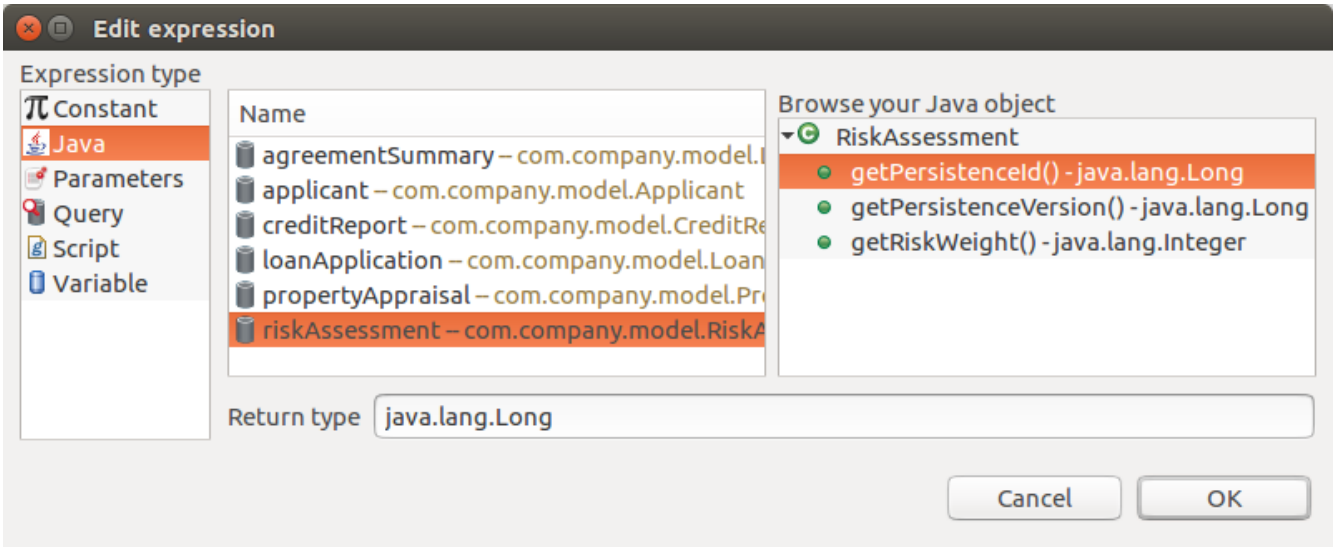
The script you have created should be reflected in the operation for the task "Assess loan risk".

Finally, create an operation that assigns the *persistenceId* of the newly created risk assessment object to the *riskAssessmentId* variable of the loan application object.

- Push the "Add" button to create a new operation.
- Select the loan application business variable as target.
- Select "Use a Java method" for the operator type.
- Select the "setRiskAssessmentId(Long)" as that Java method.

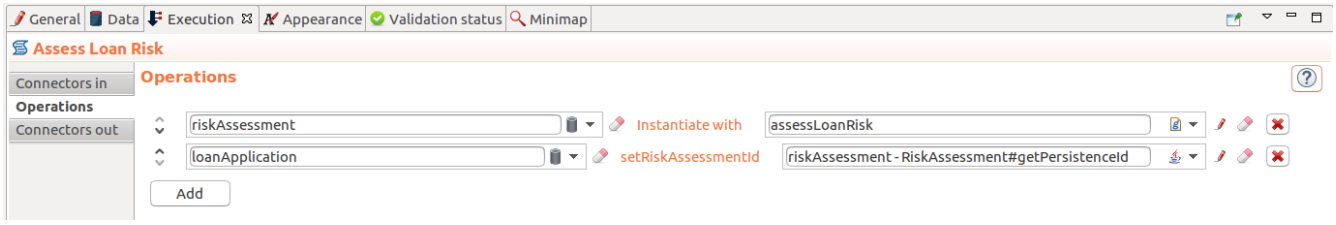


- Press the edit button (pencil icon) for the expression to determine the value to assign.
- As expression type, select “Java”,
- Select the riskAssessment business variable.
- Select its “getPersistenceId()” method.



- Press the “OK” button.

Your expression should be reflected in the operation you have just created:



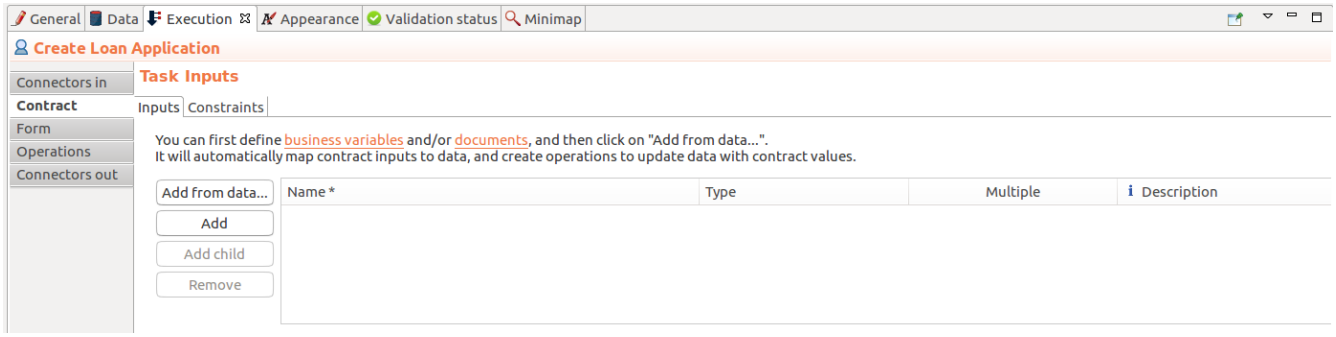
User Interface

With most of the process logic in place, it is time to the user tasks. User tasks (as well as pools) are specified using the concepts of contracts, forms, and operations. A contract is a set of business and process variables that serves as input to the task. A form is the visual user interface specification that presents these variables to the user and allows the user to make changes to the data. Finally, as noted earlier, operations are post-task assignments of values, including assignments from the user interface variables to business and process variables.

Bonita Studio provides a user interface designer that can automatically create basic forms from contract information. Bonita Studio can also create basic operations from contract information. For this tutorial, you will use the automatic generation of forms and operations.

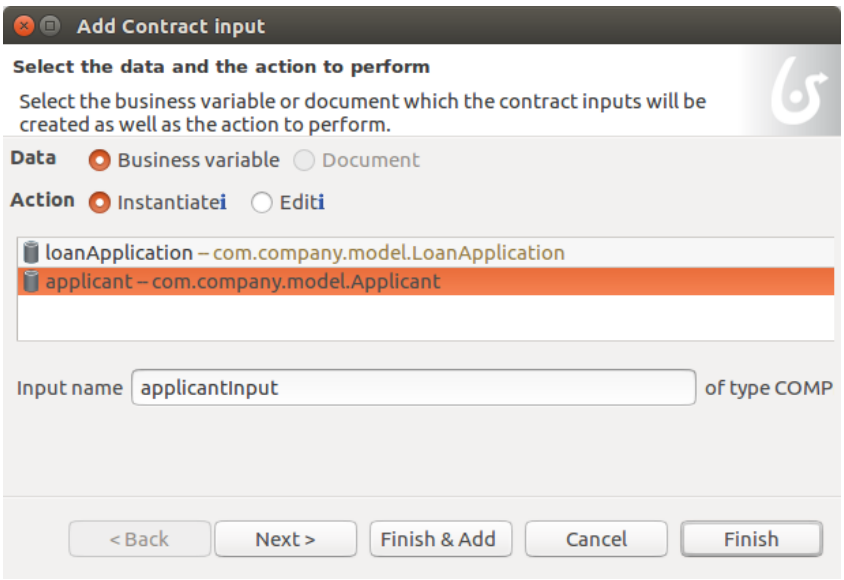
Begin with the user task “Create Loan Application” in the loan application pool.

- Select the task “Create Loan Application”, then select the “Contract” panel of the “Execution” tab of the properties pane below the diagram area

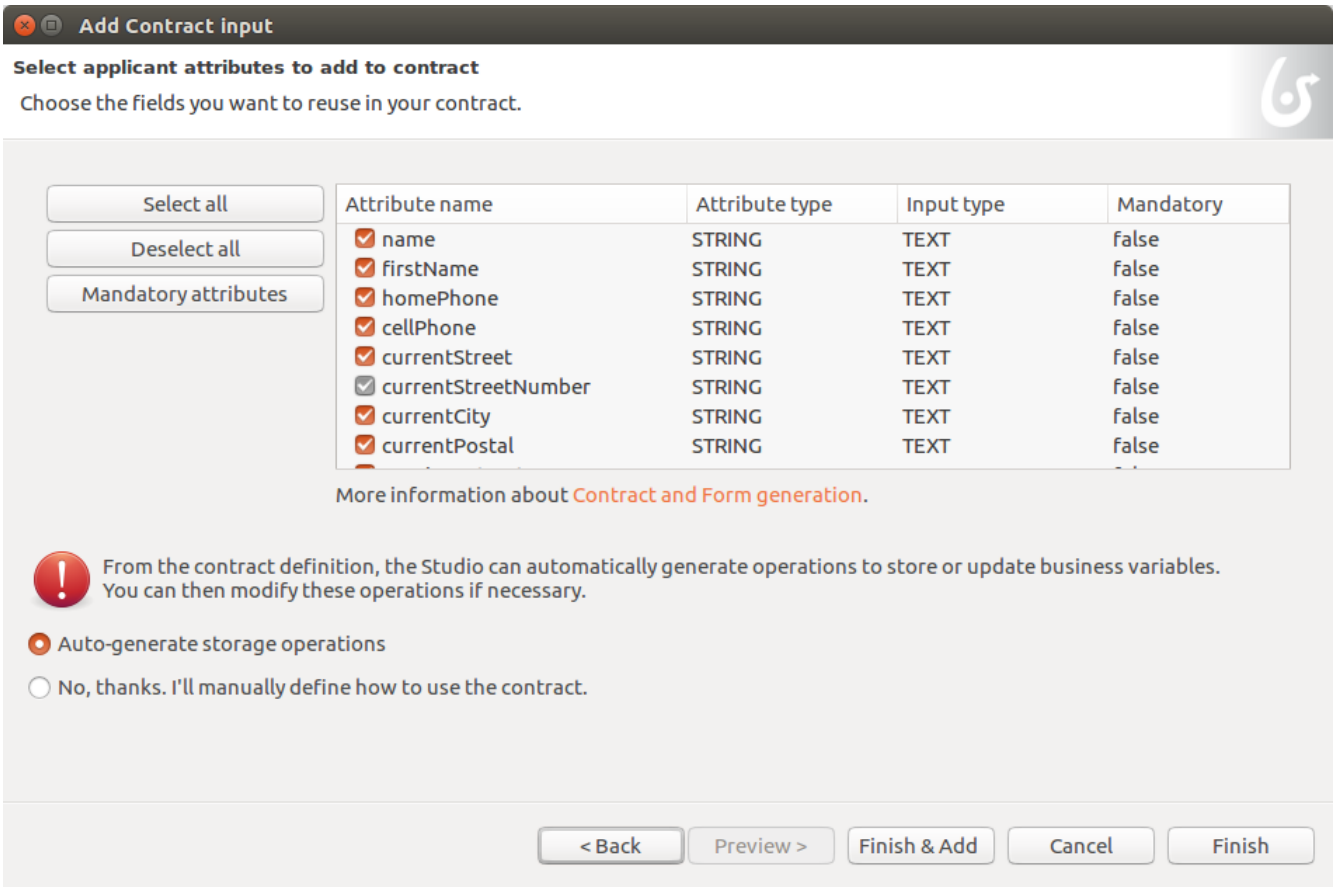


While you can manually specify the inputs to the task (using the “Add” button), you will instead use the existing data definitions to do this.

- Press the “Add from data ....” button



- Make sure that the "**Instantiate**" action is selected.
- Select the "applicant" business variable, then press the "Next >" button to select attributes of the applicant to be included in the contract



- Include all variables, and ask Bonita to automatically generate storage operations.

Storage operations are carried out once a task finishes and are used to write back information from the user interface form to the business and process variables.

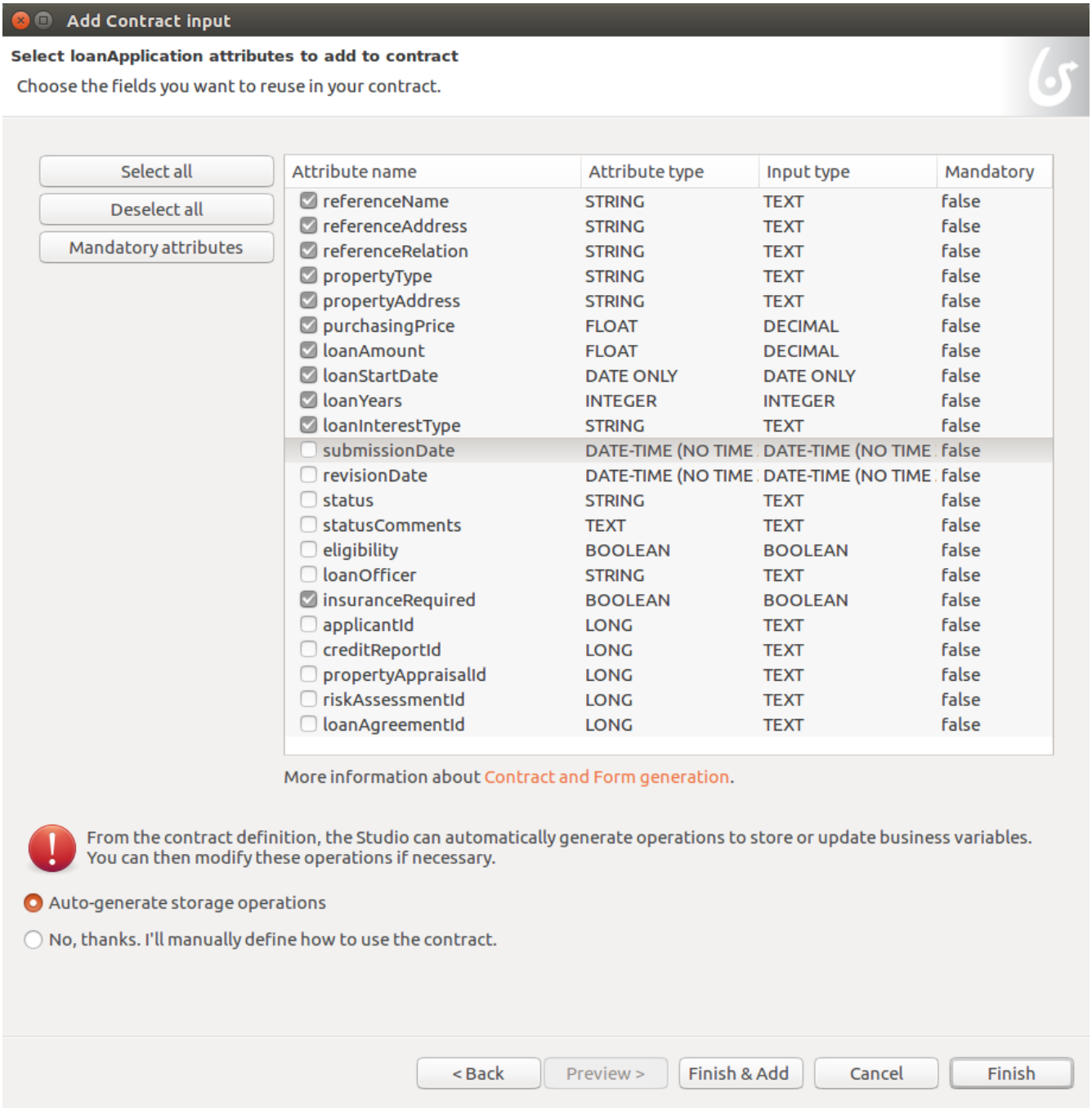
- Press "Finish & Add".

The same dialog window will appear.

- Select the "loanApplication" business variable
- Select "**Instantiate**" as the action
- Press "Next >".

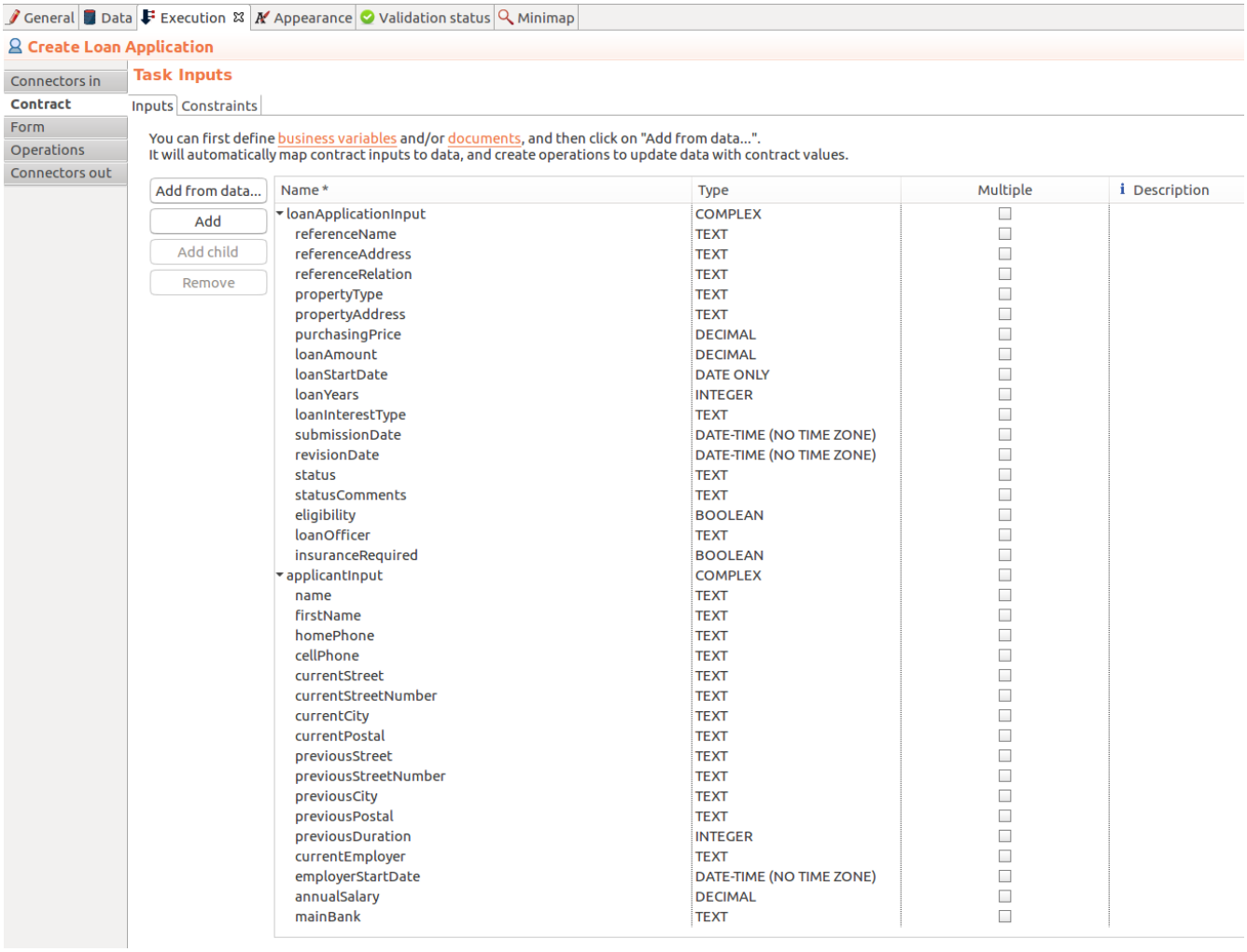
In contrast to the applicant information, there are some attributes of the loan application that you would not want the applicant to fill in. In particular you should un-select the following attributes:

submissionDate	eligibility	propertyAppraisalId
revisionDate	loanOfficer	riskAssessmentId
status	applicantId	loanAgreementId
statusComments	creditReportId	



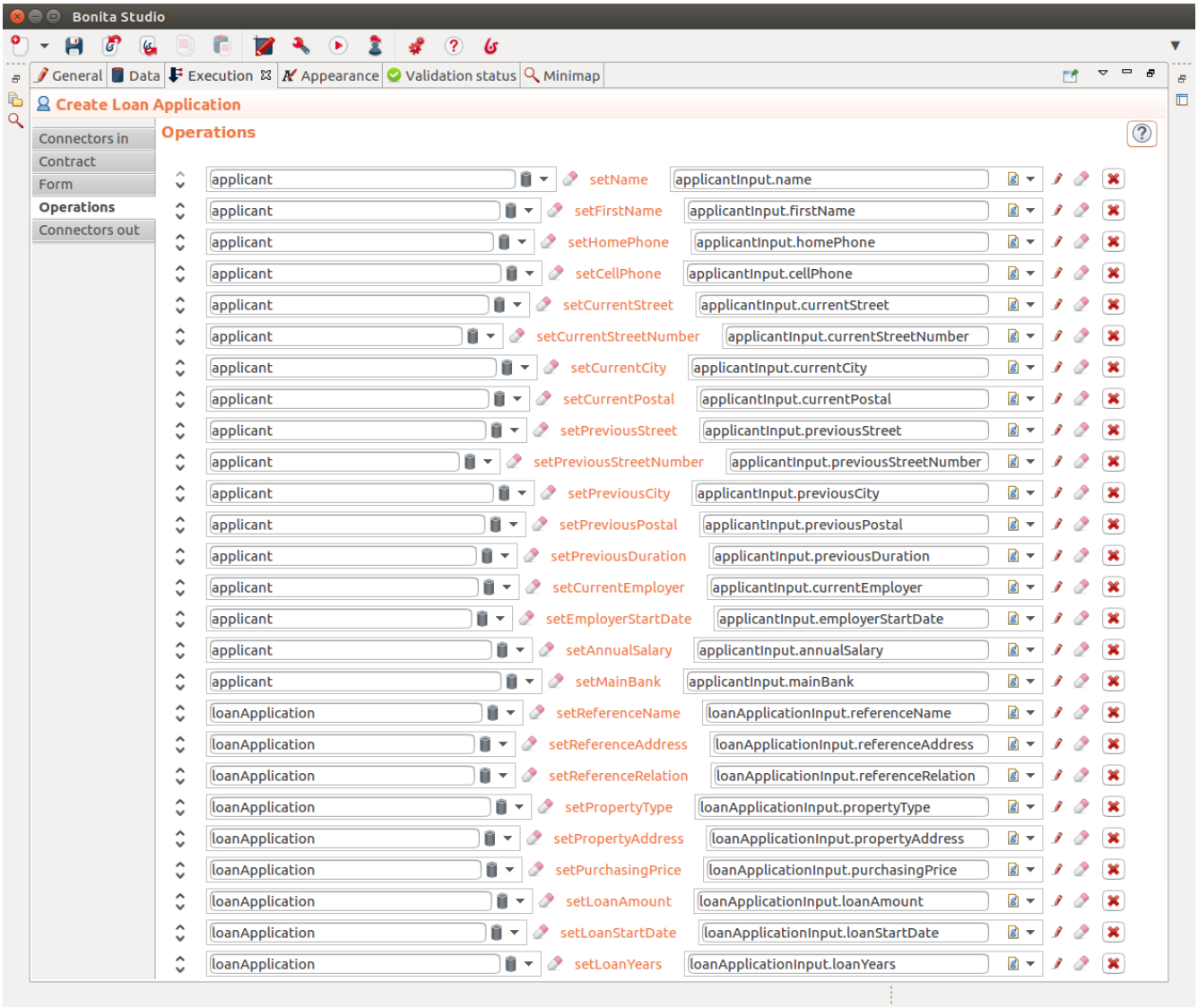
- Press the “Finish’ button to complete the addition of variables to this contract.

After the addition of each variable, Bonita informs you that it also has created operations and that no refactoring is provided. You can verify the input contract by expanding the two entries in the inputs list:

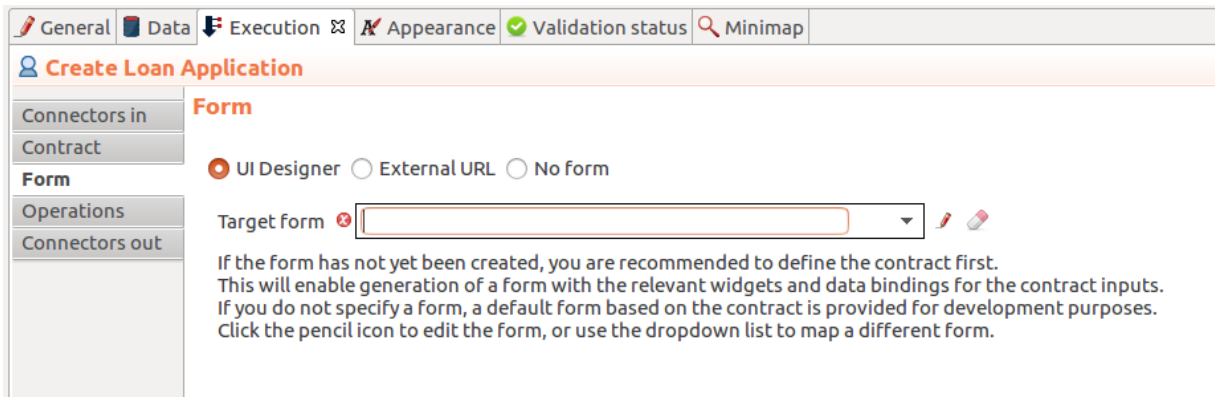


- Switch to the “Operations” panel to see the generated operations.

Bonita has generated one assignment operation for each attribute of each of the two business objects. Examining these operations in detail, the operations use the setter methods for each attribute to change the values based on the task inputs. You can remove operations for values that you do not want changed/edited and you can change the expressions on the right hand side if you want more complex value calculations and assignments.

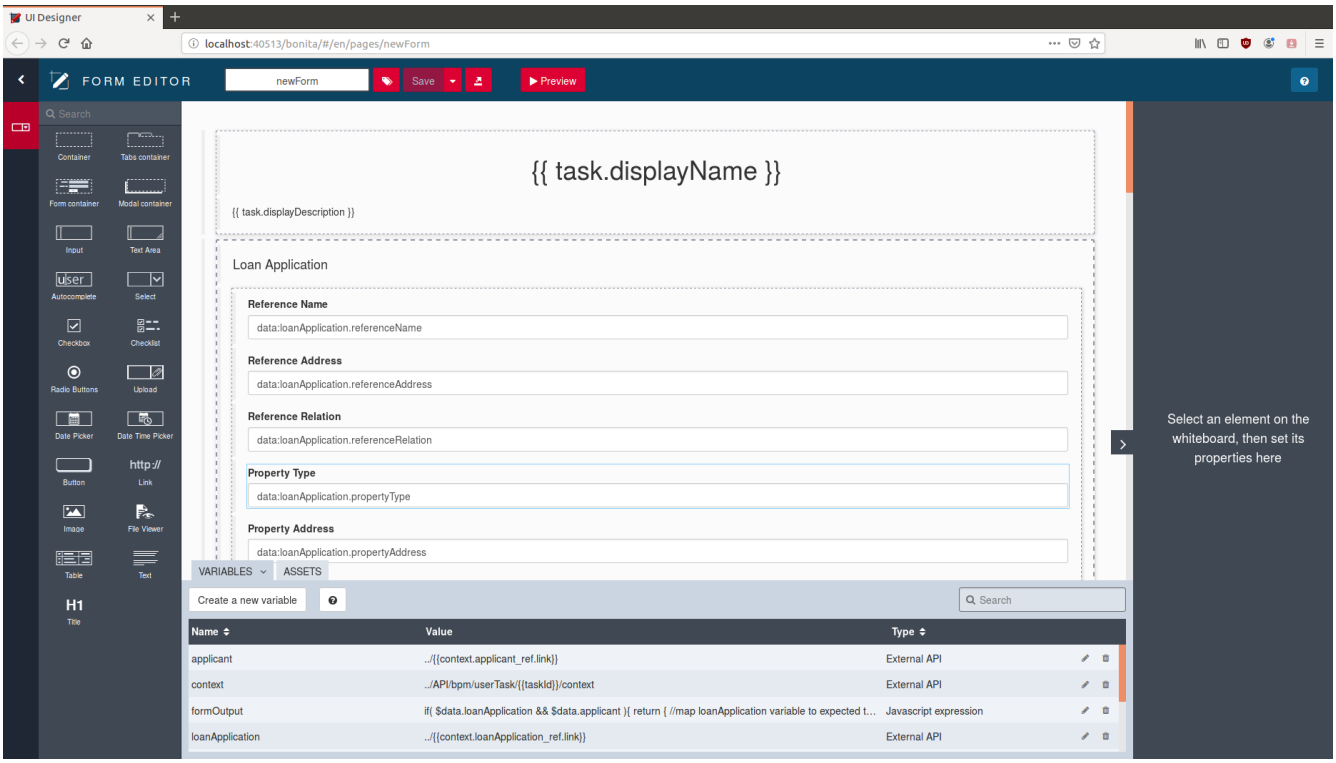


- Switch to the “Form” panel to create the user interface form for this contract.

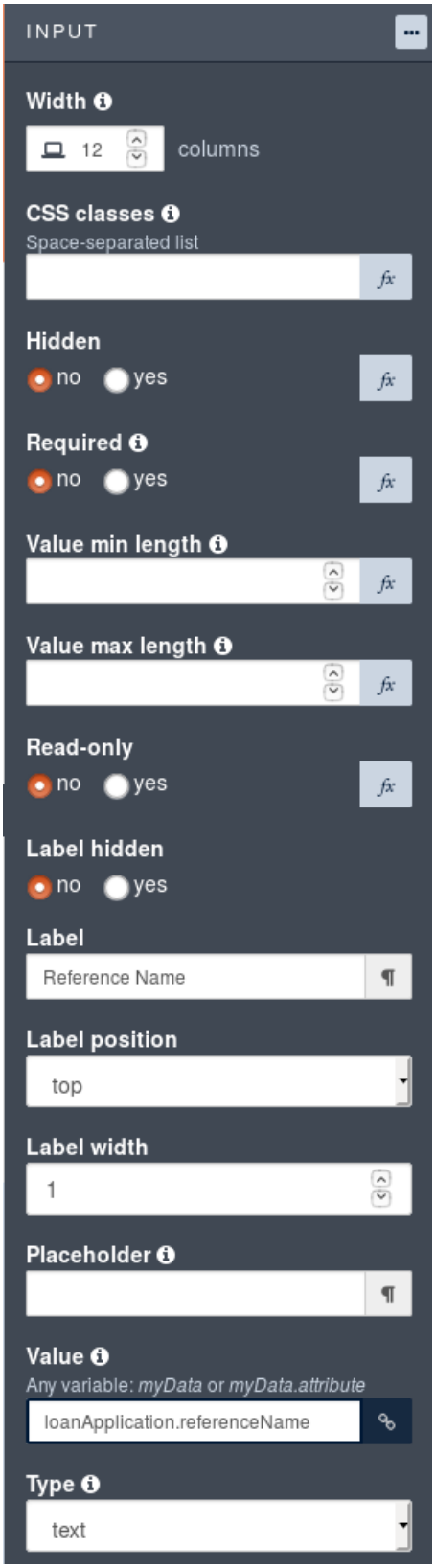


- Select the “UI Designer”
- Press the edit button (pencil icon).

This will open your web browser with the UI designer and a new form created from the task inputs.

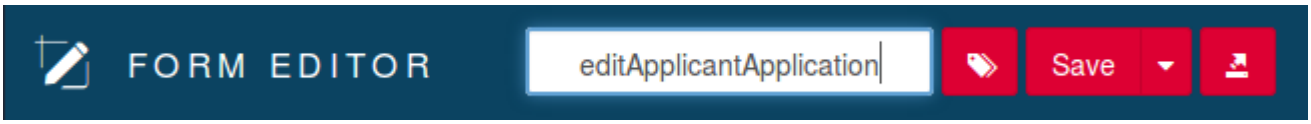


When you select one of the input elements on this form, the property pane on the right will present you with options for this input element, as shown in the figure on the next page. Note in particular the name of the JavaScript variable towards the bottom of the list. This is the same name that is used in specifying the contract and the operations in Bonita Studio. Most of the properties are self-explanatory and you can change them as required.

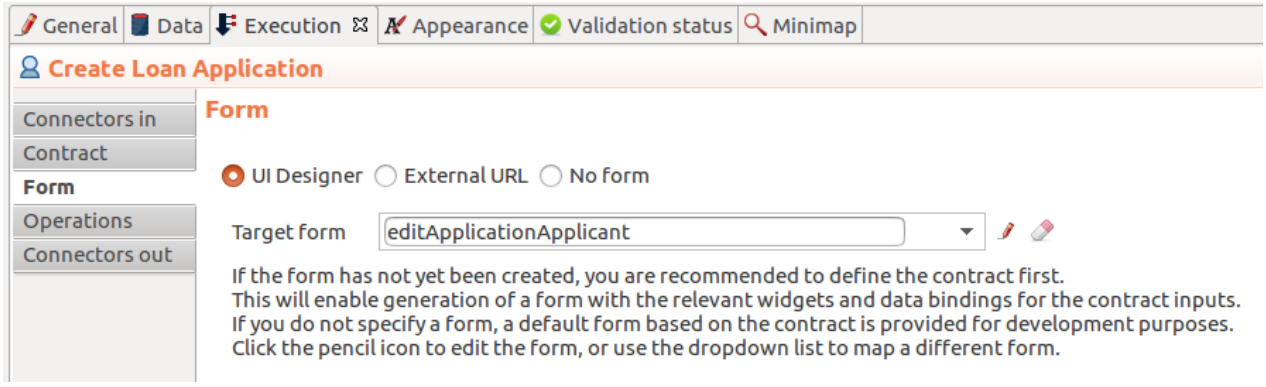

INPUT
Width
12
columns
CSS classes
Space-separated list
Hidden
no
yes
Required
no
yes
Value min length
Value max length
Read-only
no
yes
Label hidden
no
yes
Label
Reference Name
Label position
top
Label width
1
Placeholder
Value
Any variable: myData or myData.attribute
loanApplication.referenceName
Type
text

- At the top of the page, name the form “editApplicantApplication”.
- Save it using the “Save” button.

You will re-use this form for other tasks as well.

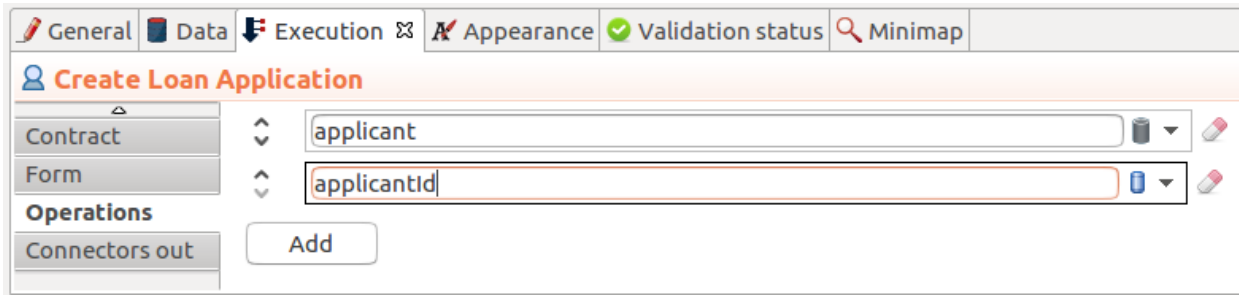


You can close the web browser window or tab. The form name should be reflected in the “Form” panel of the “Execution” tab of the properties pane, as shown below:



In addition to the automatically defined operations, you will need to define two further operations. Recall that the subsequent message end event sends a message with the applicant Id and the loan application Id as message content. Both are process variables, and you need to provide values for these in order for the message sending to be useful.

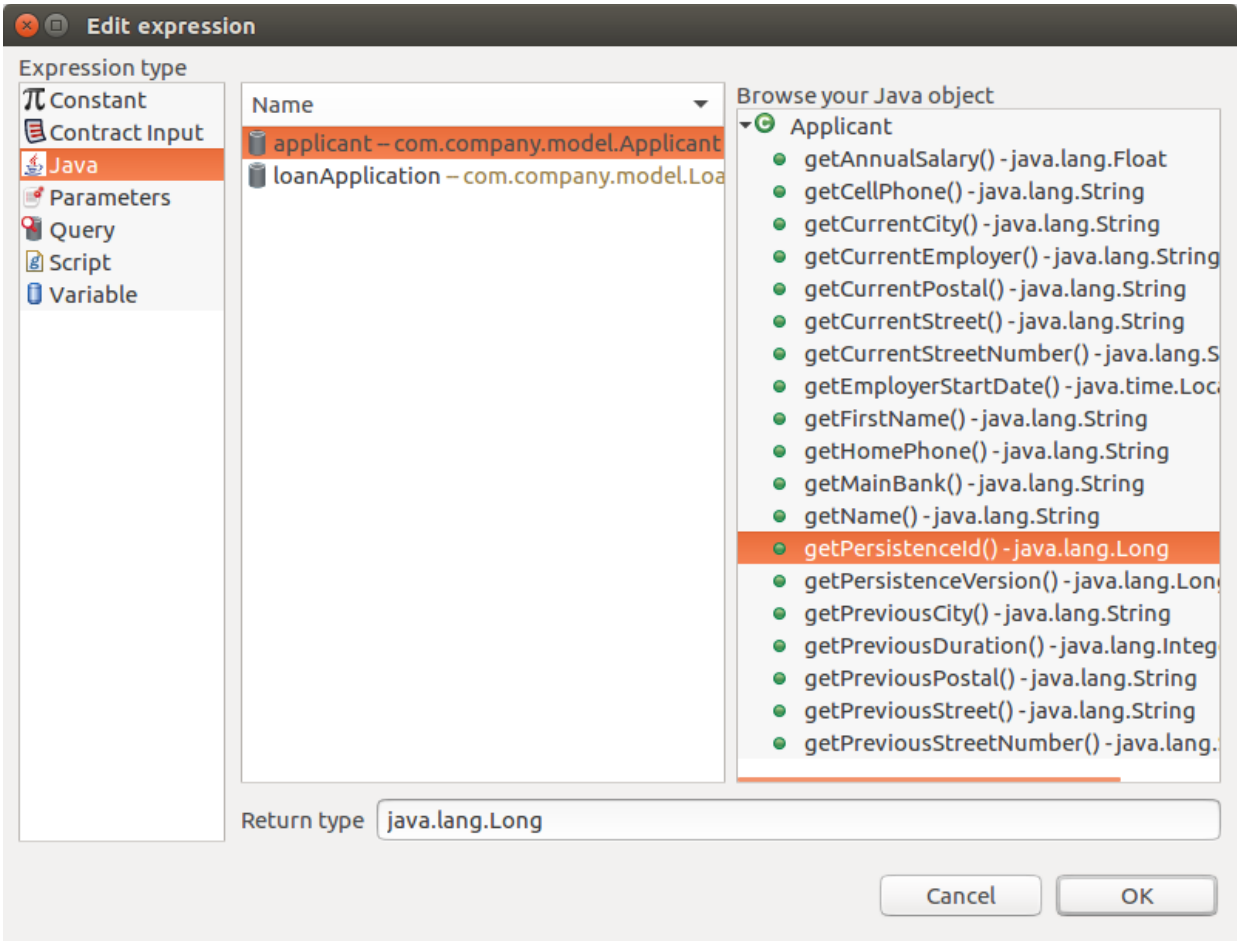
- Scroll to the end of the operations list, then use the “Add” button to add an additional operation.
- Select the “applicantId” process variable as target.



- Use the edit button on the right (pencil icon) to define the expression for this variable.

In the following dialog window:

- Select “Java” as the type of expression on the left
- Select the applicant business variable in the center column
- Select the “getPersistenceId()” method in the right column

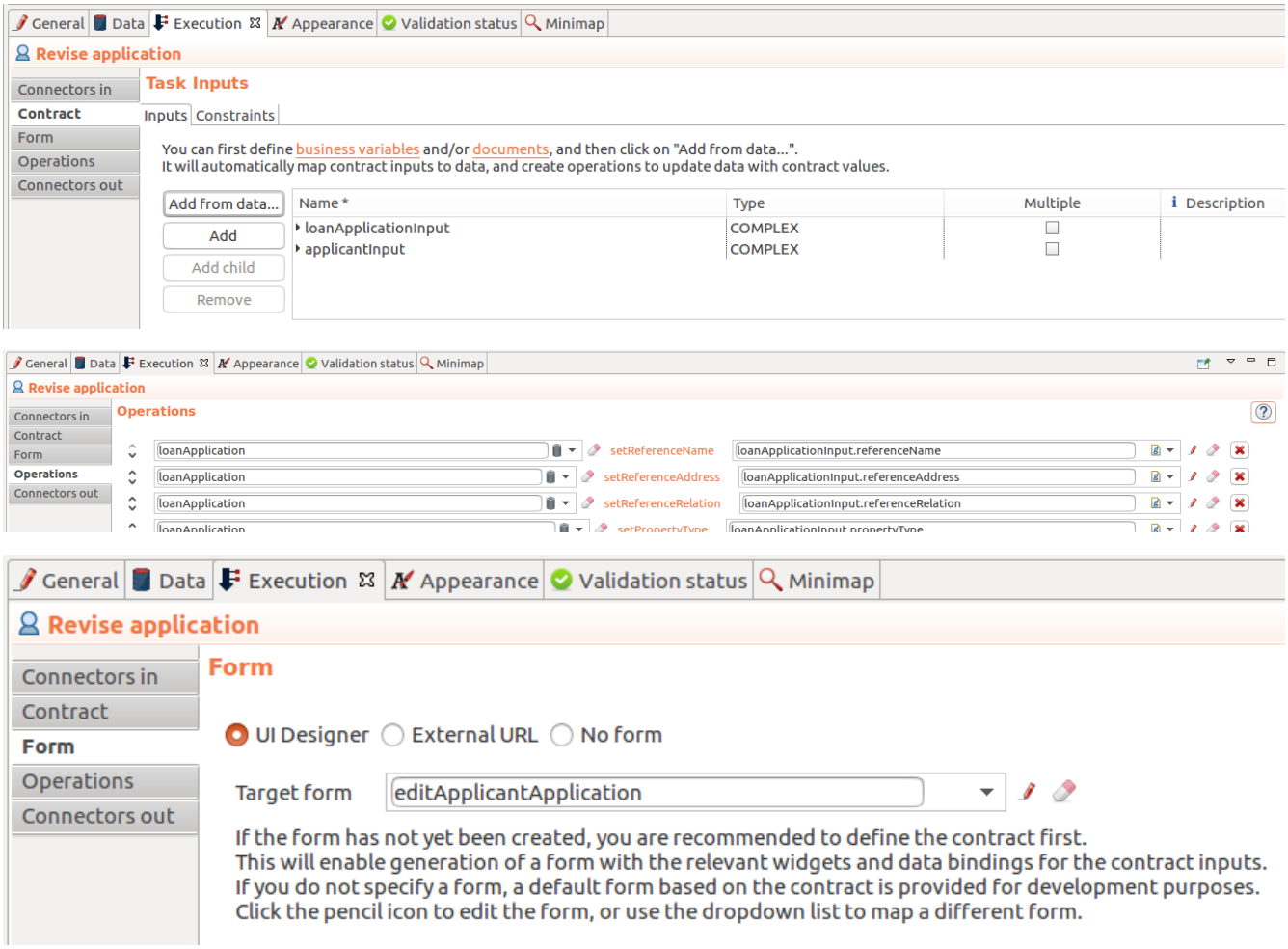


- Press the "OK" button to finalize the expression.
- Repeat the same process to create an operation for the loanApplicationId, using the "getPersistenceId()" method of the loanApplication business variable.

Your two additional operations should look as follows:

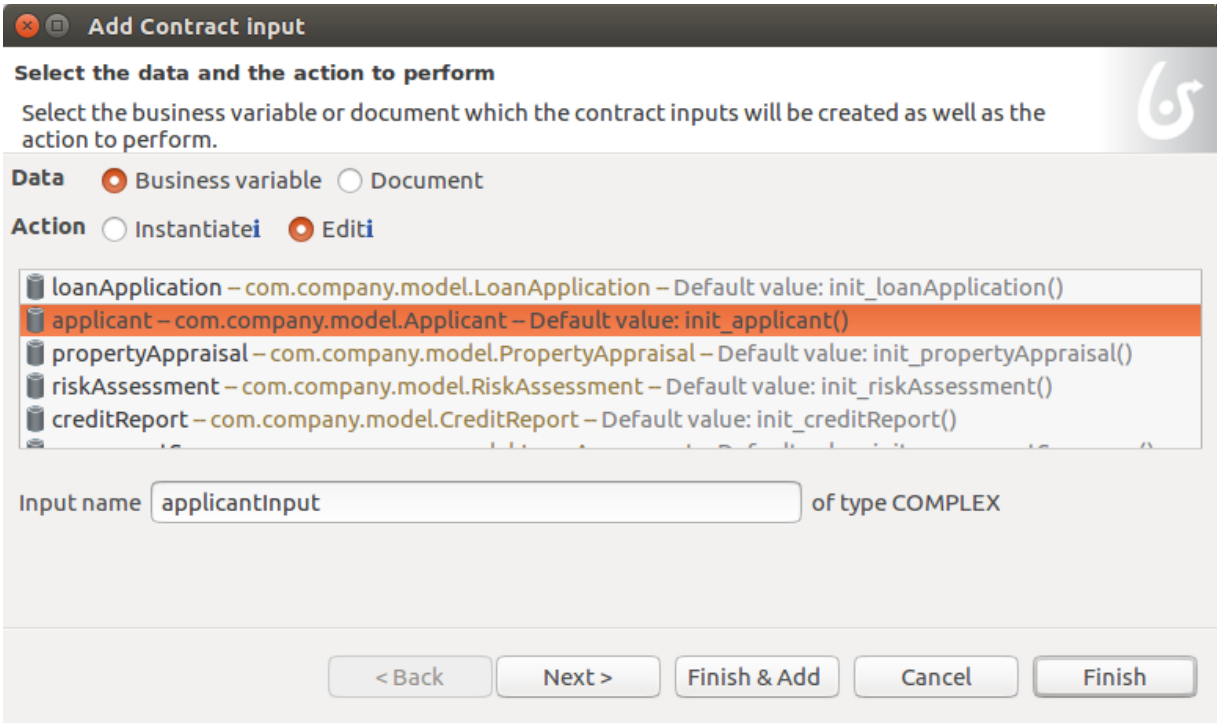
- Use the same procedure to define the execution properties (contract, form, operations) for the task "Revise Application" in the loan application pool, *but with two differences*:

1. Instead of selecting the "Instantiate" action when adding data to the input contract, make sure you select the "**Edit**" action.
2. Instead of creating a new form, select the existing form "editApplicantApplication" from the drop-down list. Your execution properties should look as follows:



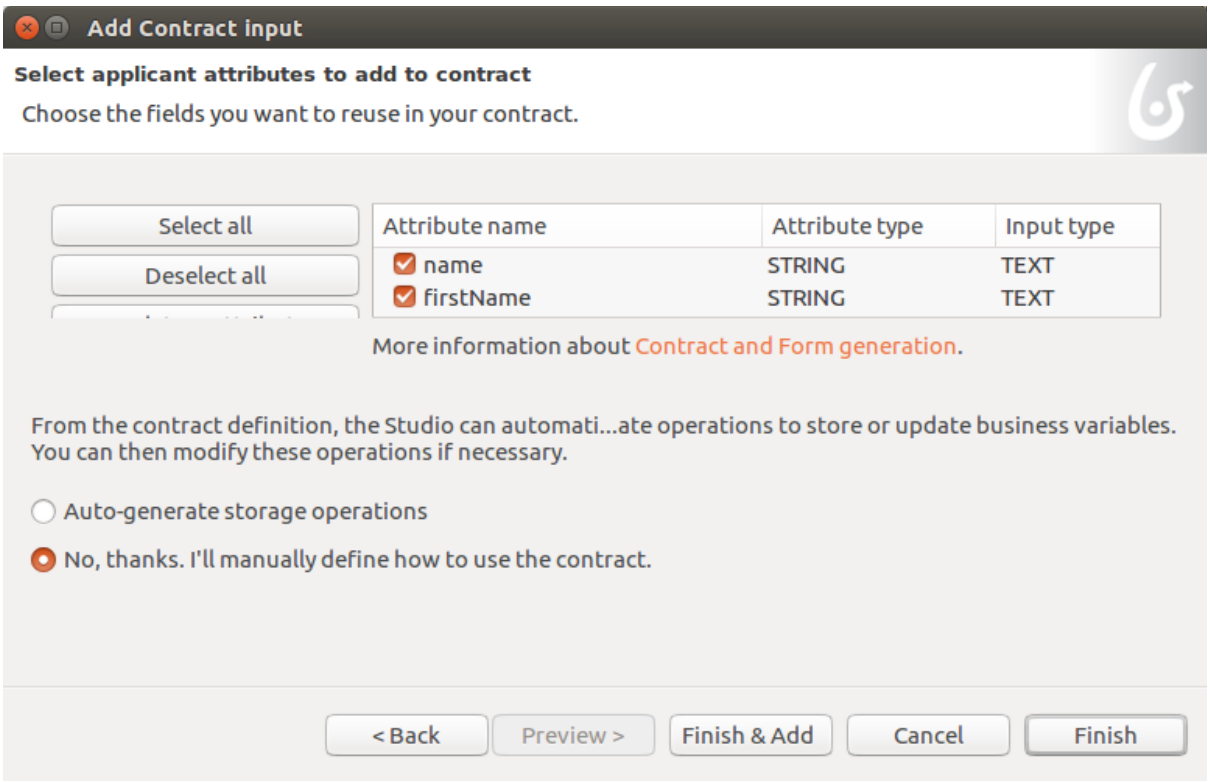
Next, consider the "Appraise Property" task. From Fig. 10.12 in "Fundamentals" you know that appraise property requires the loan application (which includes the applicant information) as input and creates a property appraisal as output.

- Select the "Appraise Property" task.
- Select the "Contract" panel in the "Execution" tab of the properties pane.
- Using the same method as above, create a contract in which the loan application, the applicant and the property appraisal are data inputs to this task:



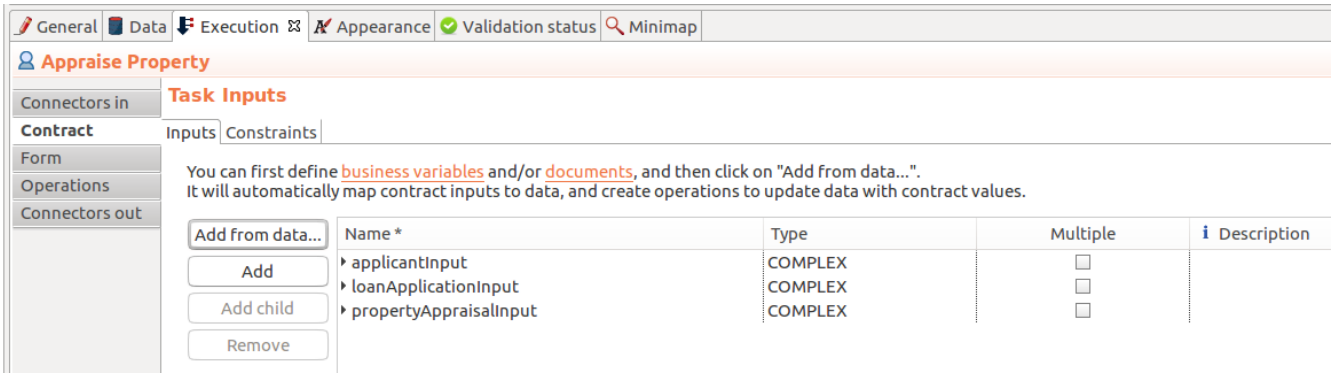
- Press the “Next >” button.
- On the following dialog, decline the creation of operations.

Applicant information is input only to this task and should not be updated after the task is complete.

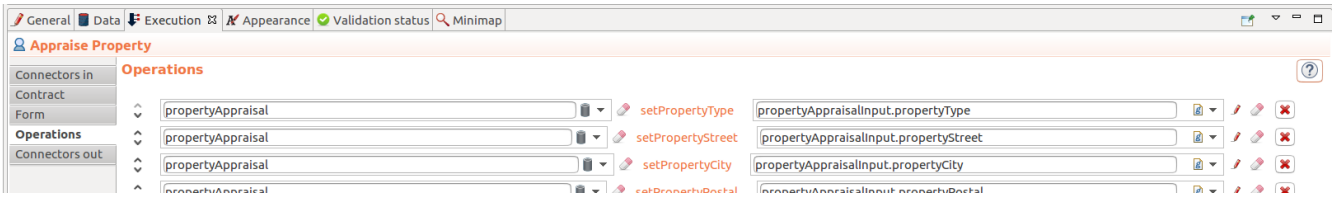


- Press the “Finish & Add” button.
- Do the same for the loan application, also declining the creation of operations. You should **de-select** the attributes for applicantId, propertyAppraisalId, loanAgreementId, creditReportId, and creditReportId. These should never be in any input contract nor should these be visible on any form.
- Press the “Finish & Add” button.
- Select the propertyAppraisal business variable as input.
- Ensure you select the “**Instantiate**” action in order to create a new propertyAppraisal object.
- Include all its attributes and ask Bonita to auto-generate storage operations for the property appraisal.
- Press the “Finish” button.

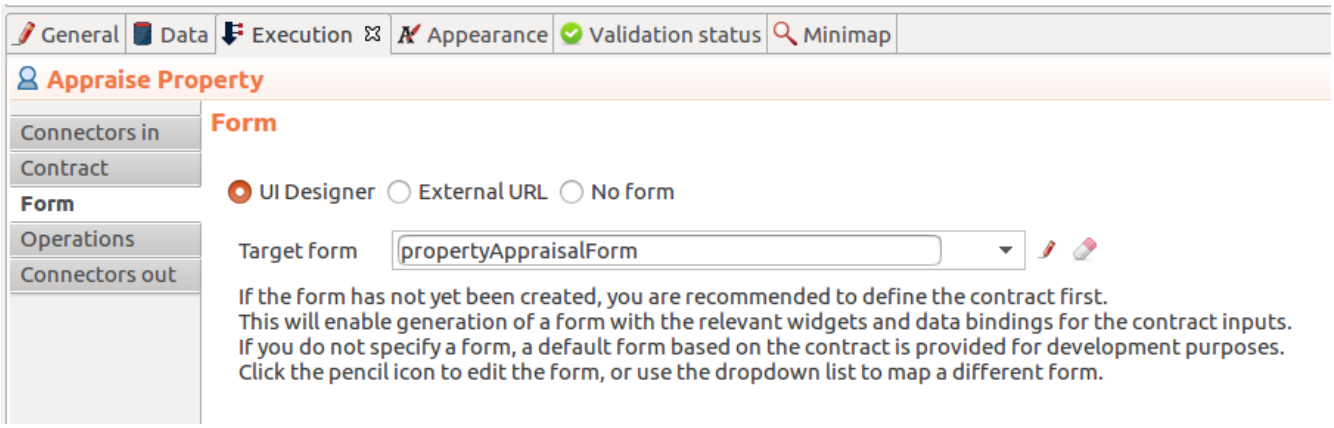
Your contract should like in the following diagram:



Your operations should include operations only for the property appraisal:

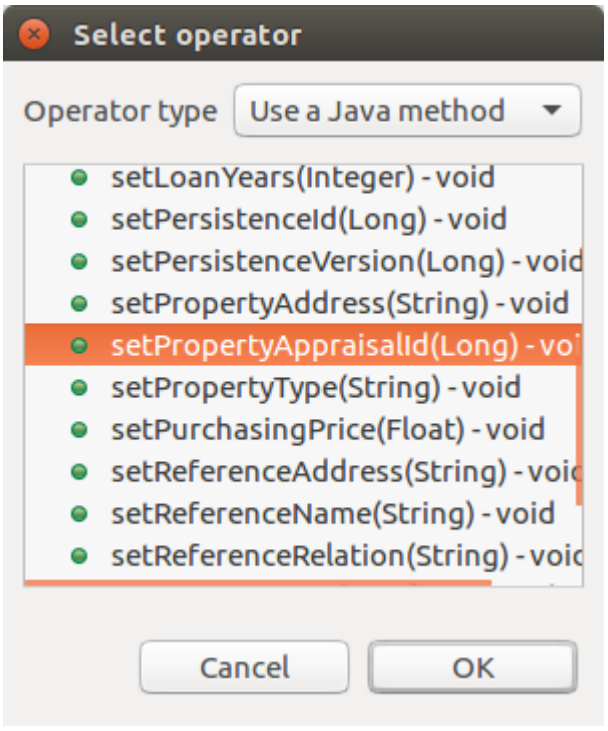


- Create a form for this task using the user interface designer.
- Name the form "propertyAppraisalForm" and set all fields for attributes of the applicant and loan application to be "read-only".

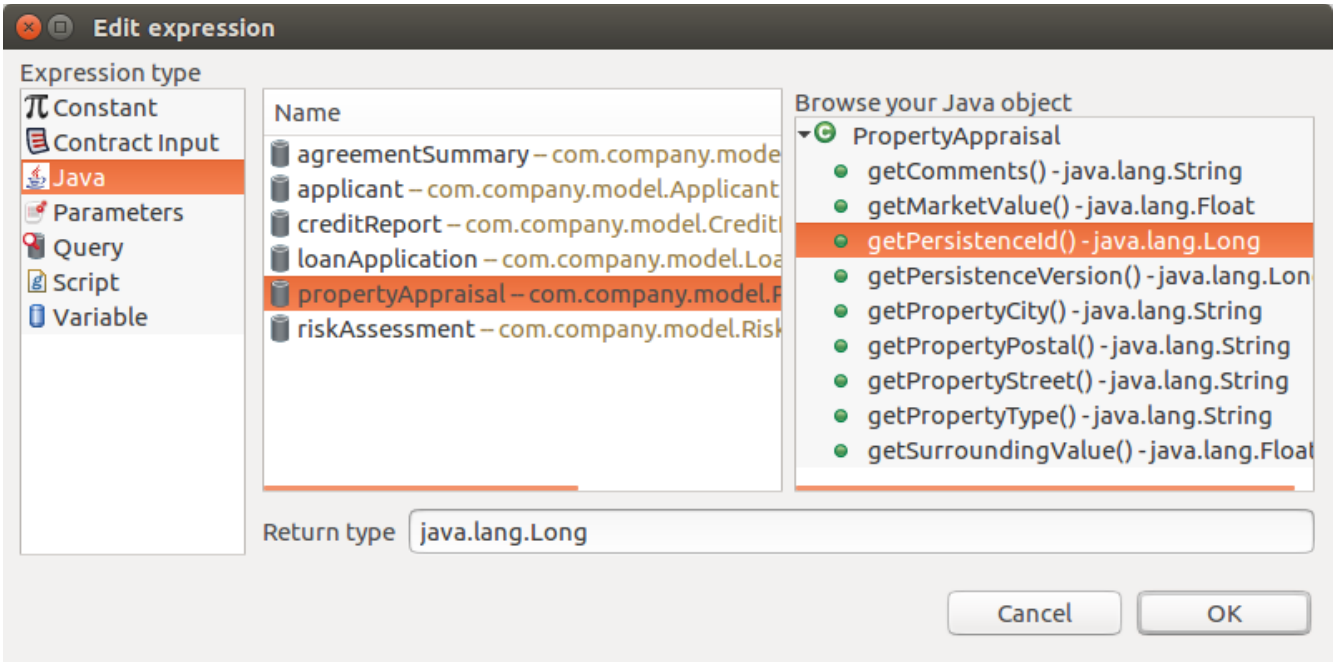


Finally, create an operation that assigns the persistenceId of the newly created property appraisal object to the propertyAppraisalId variable of the loan application object.

- Push the "Add" button to create a new operation.
- Select the loan application business variable as target.
- Select "Use a Java method" as operator type.
- Select the "setPropertyAppraisalId(Long)" as that Java method:

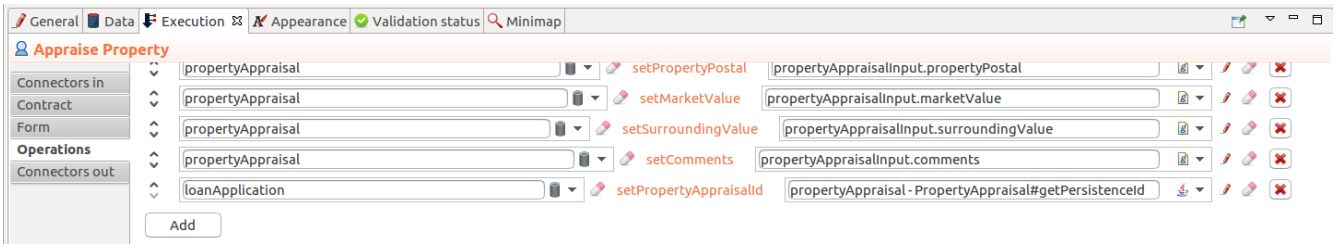


- Press the edit button (pencil icon) for the expression to determine the value to assign.
- Select "Java" as expression type.
- Select the propertyAppraisal business variable.
- Select its "getPersistenceId()" method.



- Press the "OK" button.

Your expression should be reflected in the operation you have just created:



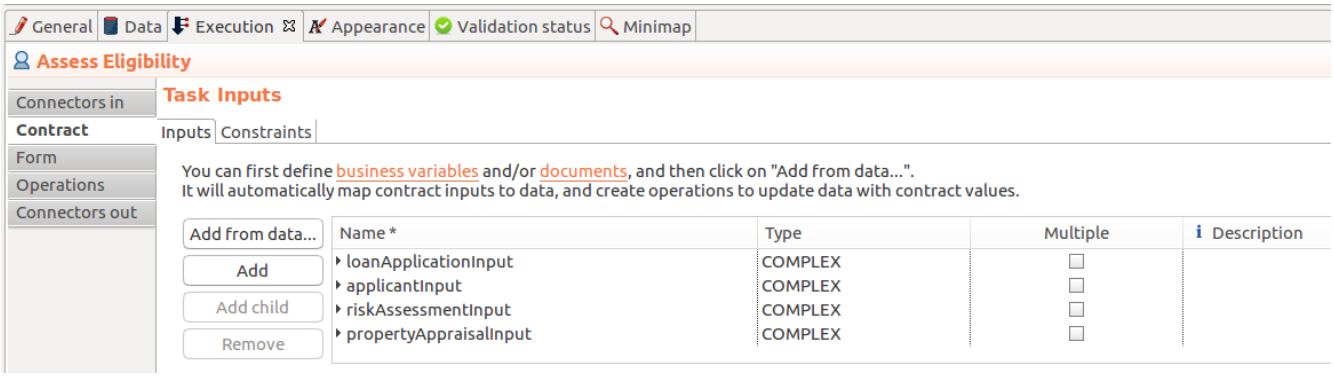
Next, consider the "check credit history" task. Figure 10.12 in "Fundamentals" tells you that the loan application (which includes the applicant) is input to this task, and the credit history report is output of this task.

- Create the contract, form, and operations for the task "check credit history".

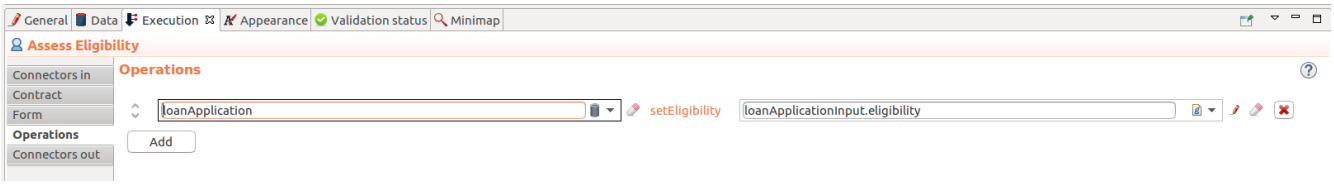
Make sure you select the "**Instantiate**" action for the credit history report and that you create operations only for the credit history attributes. Also ensure that attributes of the applicant and loan application are read-only on the user interface form. Finally, do not forget to set the *persistenceId* of the new credit history report object in the loan application attribute *creditReportId*.

Next, consider the task “Assess eligibility”. Figure 10.12 in “Fundamentals” tells you that the loan application (including applicant information), the risk assessment, and the property appraisal are input to this task, and a possibly changed loan application is output of this task.

- Create the contract, form, and operations for the task “Assess eligibility”.



Because this task assesses eligibility, assume that only the attribute *isEligible* of the loan application can be changed by this task. You need one operation:

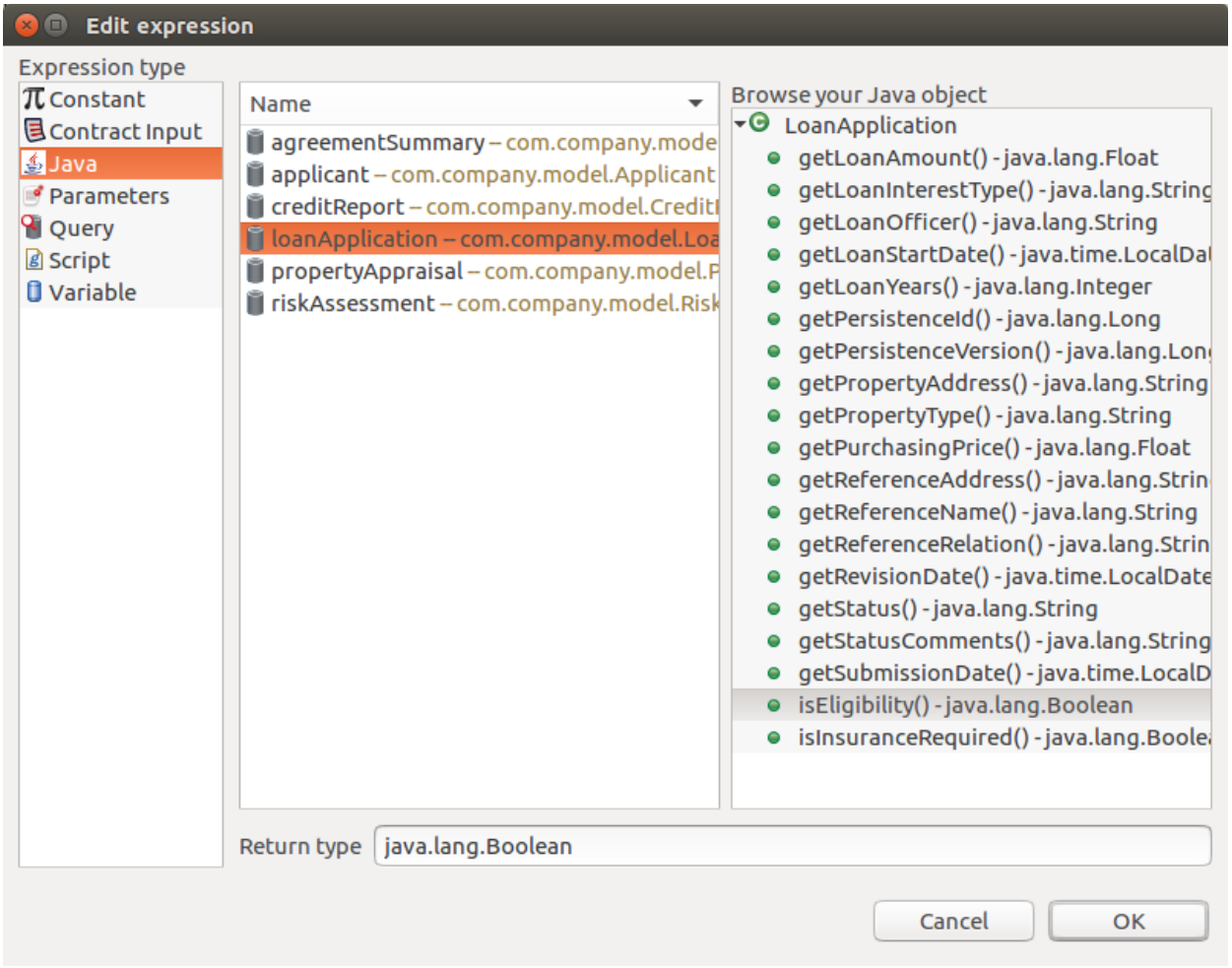


However, the following exclusive (XOR) gateway has flows with conditions that depend on the process variable *isApplicantEligible*. Hence, you need to create an operation for the task “Assess eligibility” that sets the value of this process variable from the form input.

- Press the “Add...” button to create a new operation.
- Select the process variable *isApplicantEligible* as target.
- Select the edit button (pencil icon) to define the expression for this variable.

In the following dialog window:

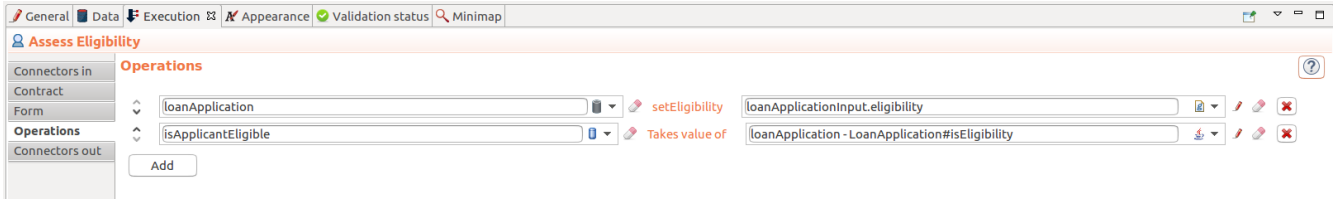
- Select “Java” for the type of expression.
- Select the loan application variable
- Select the “isEligibility()” access method.



- Press the "OK" button to finalize your expression.

Explanation: You can get the value from the loan application business variable because operations are executed in the order in which they are specified. This operation is carried out after the loan application is updated with the form value.

Your operations for the task "Assess eligibility" should look as in the following figure:



According to Fig. 10.12 in "Fundamentals", tasks "Prepare acceptance pack," "Send acceptance pack," and "Send home insurance quote" require only the loan application (with applicant information) as input, and have no outputs. Therefore, you do not need to create any operations, and can make all form elements read-only. As all three tasks use the same inputs (and outputs), you can reuse the form that you create for the first of these.

- Specify input contract and form(s) for these three tasks

The task “Verify repayment agreement” requires the loan application (with applicant information) as input, and produces the agreement summary as output.

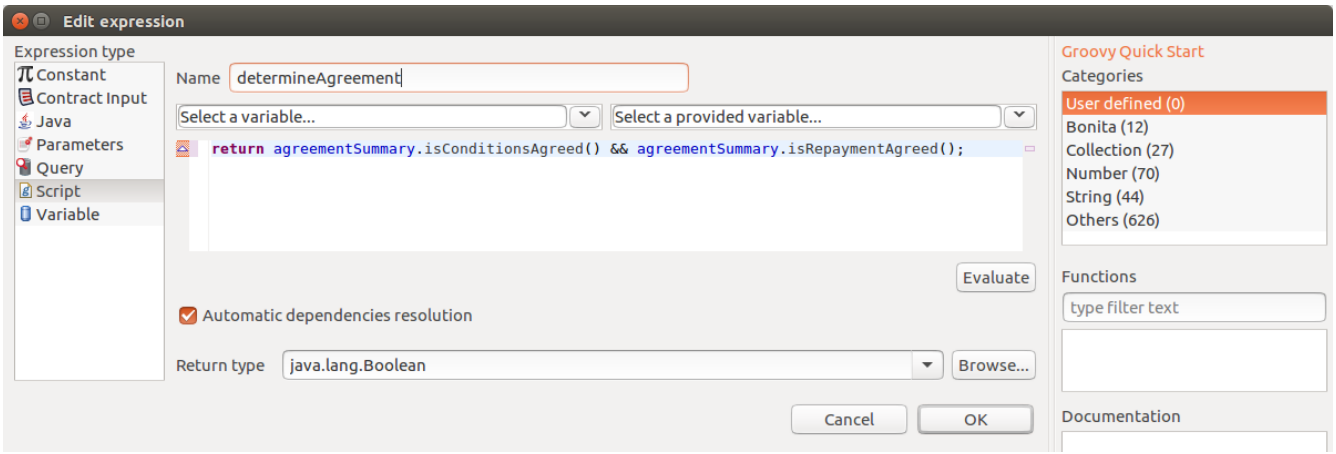
- Create contract input, form and operations for task “Verify repayment agreement”.
  - Select the “**Instantiate**” action when adding the loan agreement summary to the input contract
  - Define operations only for the agreement summary attributes, not for the loan application attributes.
  - Create a customized form for this task. Set the loan application and applicant fields to be read-only.
  - Aadd an operation that records the *persistenceId* of the new loan agreement summary object in the *loanAgreementSummaryId* attribute of the loan agreement object.

Next, you need to create an additional operation to set the value of the process variable *doesApplicantAgree* which is used in the flow conditions of the subsequent exclusive (XOR) gateway. The value of this variable should be the conjunction (AND) of the two boolean values of the repayment summary business variable.

- Use the “Add” button to create an additional operation.
- Select the process variable “doesApplicantAgree” as target.
- Select the edit button (pencil icon) to create an expression for the value to assign to this variable.

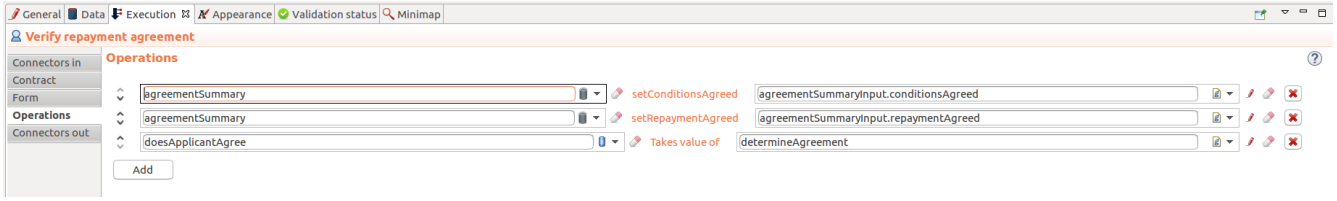
In the following dialog window,

- Select “Script” as the type of expression
- Name the expression and enter the script as follows:



- Push the “OK” button to complete the expression.

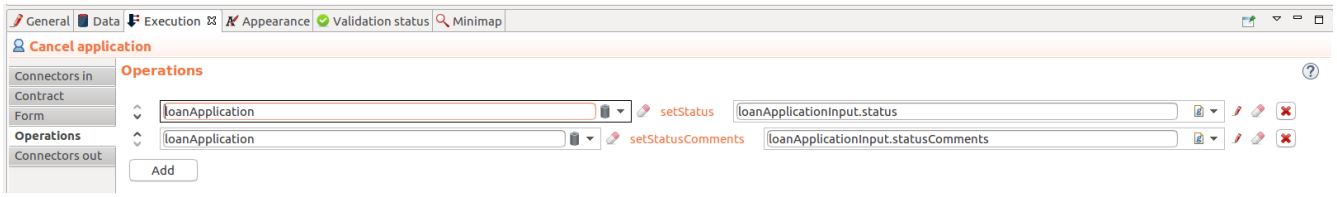
The “determineAgreement” script should be reflected in the list of operations for the task, shown in the following figure:



The two final tasks “Cancel application” and “Approve application” both have the loan application (with applicant information) as input and output and also require the agreement summary as input. Here too, the form you create for one task is reusable for the other task. You can assume that only the status and comments on the status of the loan application can be updated by these tasks.

- Create input contracts, forms, and operations for tasks “cancel application” and “approve application”

Your list of operations will look as follows and your form should have all other fields as read-only.



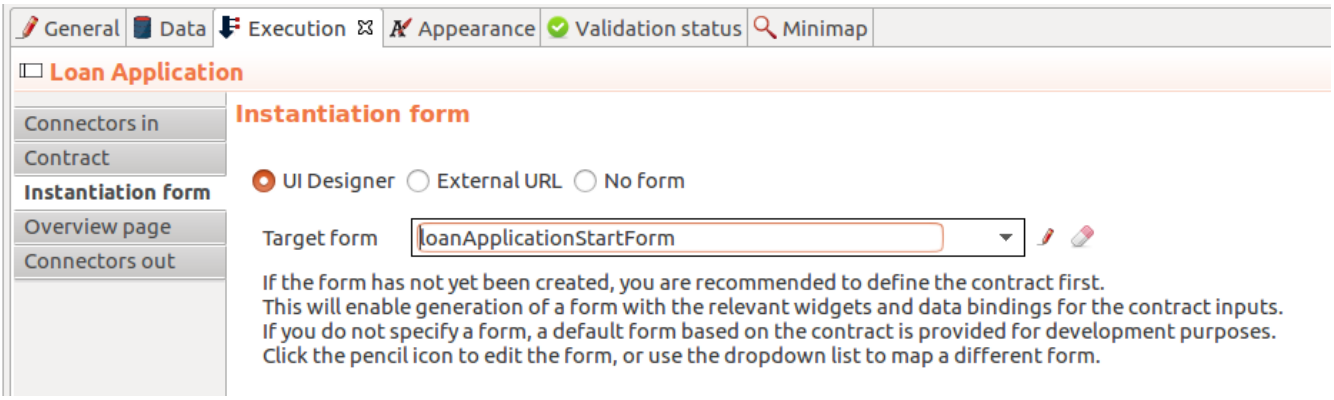
Pool User Interface

Pools in Bonita can have user interfaces just as user tasks do. In fact, pools have an input contract as well. However, in your process, variables are instantiated by task operations and you not require the user to provide any values when the process in a pool is instantiated. In fact, only the loan application pool process should be directly instantiated by the user. The loan provisioning pool is instantiated by a message and the receive updated application (sub-process) pool is instantiated by the call activity.

- Select the loan application pool.
- Select the “Instantiation form” panel of the “Execution” tab in the properties pane.
- Use the UI designer to create a new instantiation form.

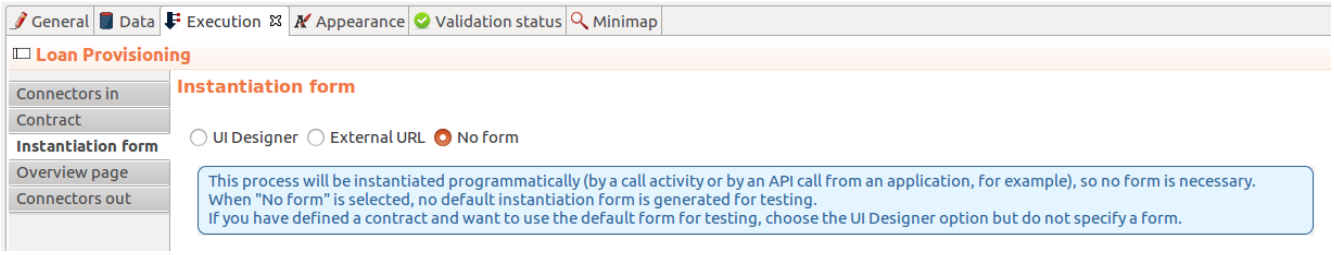
This is the same procedure as forms for user tasks. Because the input contract is empty (there are no inputs specified for instantiating this pool), the default form simply provides a “submit” button. You do not need to make any changes for this tutorial.

- Name the form “loanApplicationStartForm” and save it.



Because the loan provisioning pool's process is not instantiated by a user but by a message, no form is required.

- Select the loan provisioning pool.
- Select the "Instantiation form" pane in the "Execution" tab of the properties pane.
- Set it to "No form"



- Do the same for the receive revised application (sub-process) pool.

Runtime Variables

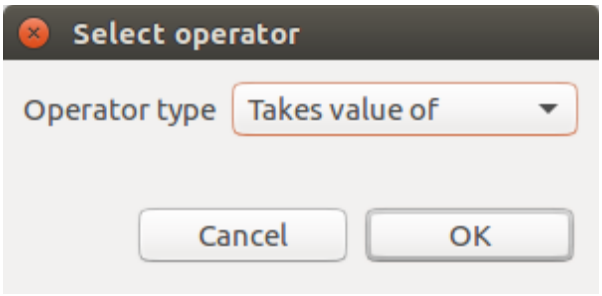
There are various situations where data should be updated automatically based on runtime variables. For example, the loan application contains an attribute loan officer which should be updated with the name of the loan officer who dealt with this application first (or last). Similarly, the credit history report has an attribute for the name of the financial officer. The values for these variables can be retrieved from runtime variables that Bonita maintains during process instance execution. The following is a useful Bonita runtime function using runtime variables:

```
BonitaUsers.getUser(apiAccessor,taskAssigneeId).userName;
```

The function returns the user name of the resource that is assigned to the task. Other attributes of a user are the *firstName*, *lastName*, *title*, *jobTitle*, *managerUserName*, and *managerUserId*.

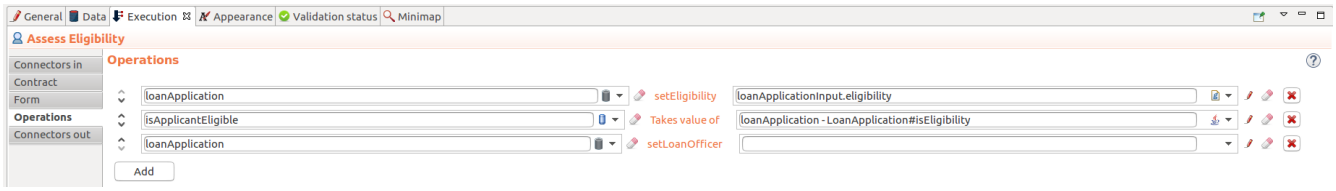
- Select the task "Assess eligibility"
- Select the "Operations" panel in the "Execution" tab in the properties pane.
- Press the "Add" button to add an operation.

- As target, select the loan application business variable.
- Change the operation type by clicking on “Takes value of”:



- In this dialog box, select “Use a Java method”.
- In the following dialog box, select “setLoanOfficer(String)” as the Java method to use.

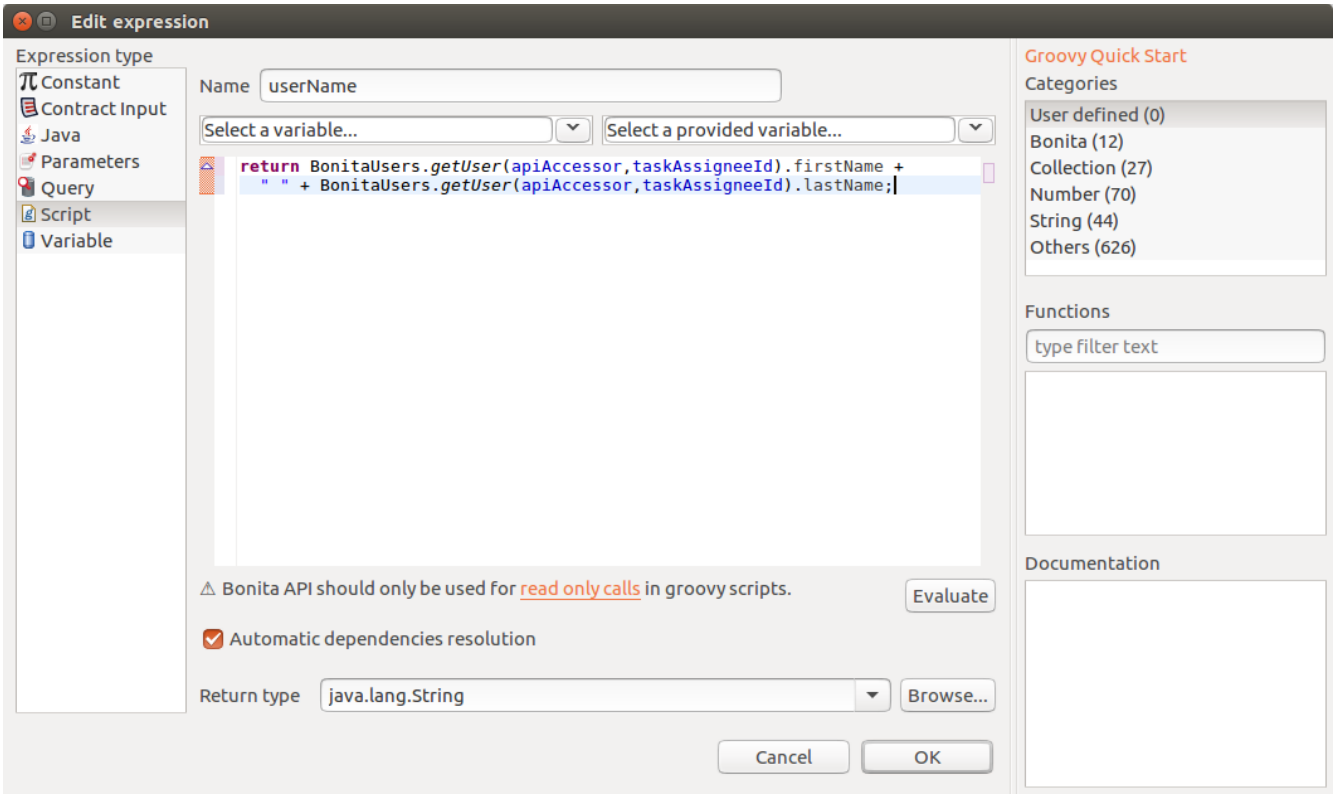
This should be reflected in the new operation:



- Use the edit button on the right (pencil icon) to edit the expression for creating the value for the parameter for that method.

In the dialog window that follows:

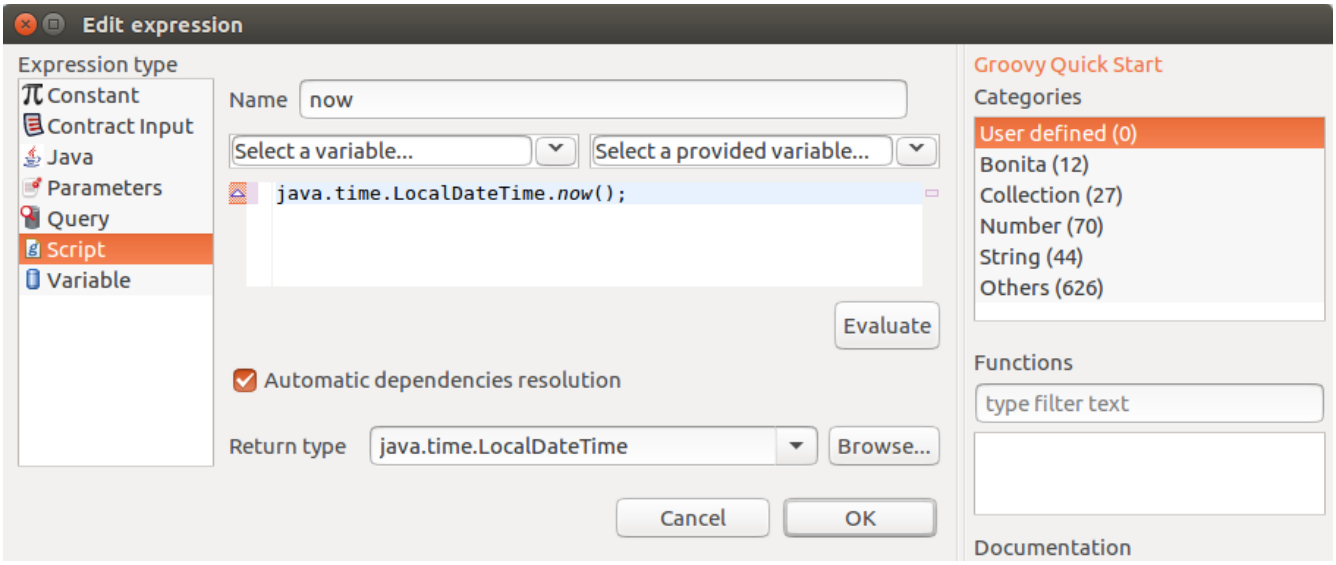
- Select “Script” for the type of expression.
- Name the expression “userName”.
- Enter the script as in the following figure (next page).



- Press the "OK" button to finish the expression.
- Do the same for the task "Check credit history" to set the name of the financial officer in the credit history report in that task's operations.

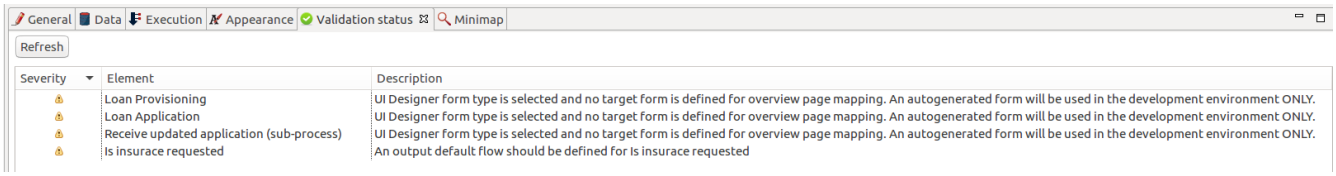
The loan application also contains the submission and the revision date. Both of these should be automatically set after the tasks "Create loan application" and "Revise application" complete.

- Select the task "Create loan application".
- In the "Operations" panel of the "Execution" tab of the properties pane, find the operation that sets the value for the attribute "submissionDate".
- Click on the edit button (pencil icon) to adapt the script.
- Change the script and its name as follows (next page)



- Do the same for the task “Revise application” and the operation to set the revision date.

At this point, the process specification is complete. Select the “Validation status” tab in the properties pane and press the “Refresh” button. You will see a number of warnings:



One warning is for the lack of a default flow from the inclusive (OR) gateway. The desired process flow requires conditions on both flows (one of which is the constant ‘true’) but no condition can be a assigned to a default gateway, our process demands. In fact, Bonita activates the default flow from an inclusive gateway only when no other flow is activated; it behaves the same as an exclusive gateway, which is not what we require for this process.

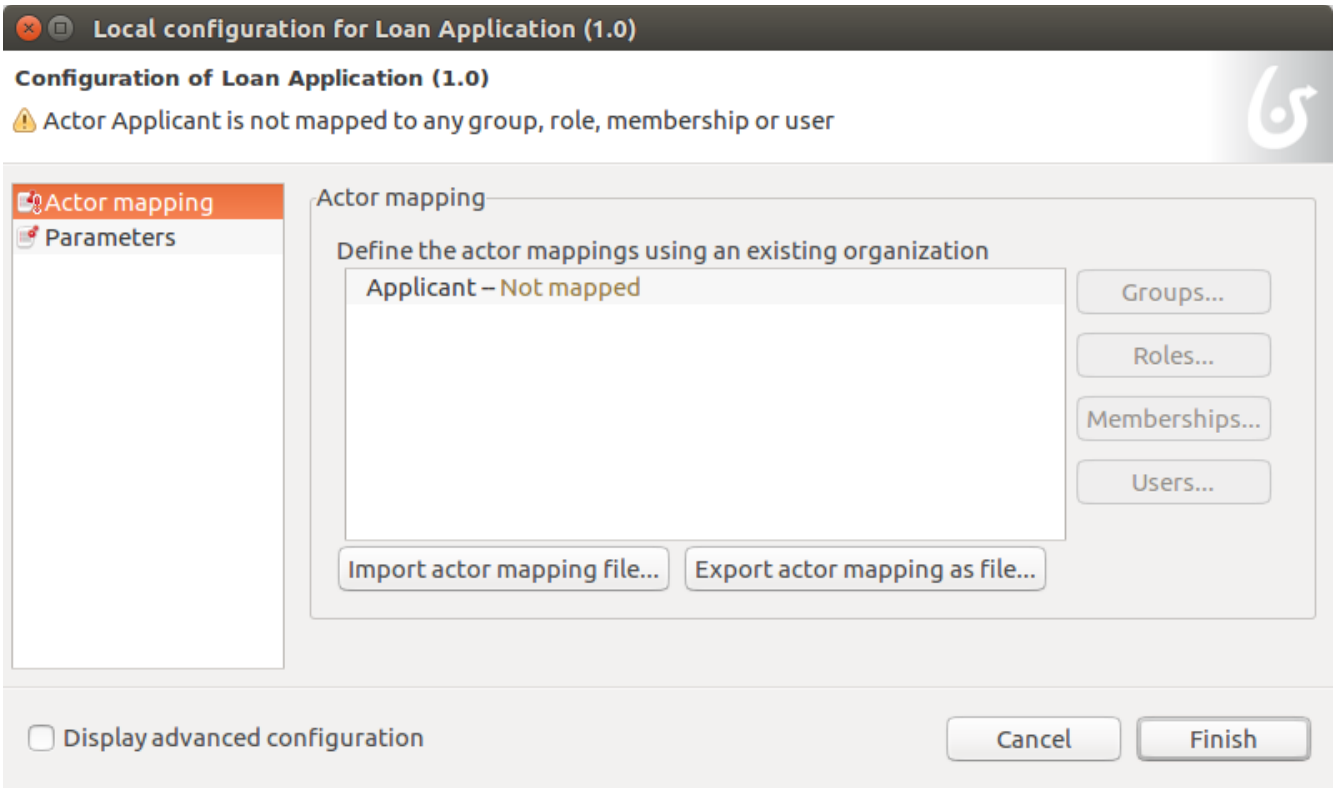
The remaining three warnings concern the lack of an overview page. Overview pages provide information about running and archived cases in the Bonita Portal. They can show current values for business and process variables, and information about completed and activated tasks. For this tutorial, you can rely on the default forms that Bonita provides for testing the process.

Process Deployment

Once the process model is specified, it needs to be deployed to the Bonita workflow management server. The primary work involved in this is the mapping of actors specified for the BPMN pools to resources declared on the workflow management server. This mapping is part of process "configuration".

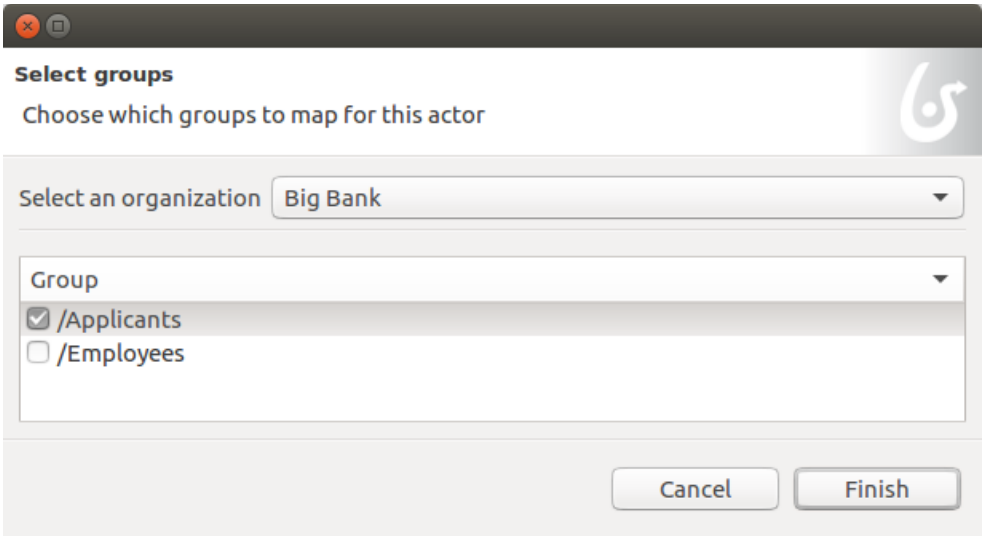
In Bonita Studio,

- Select the Loan Application pool.
- In the menu, select "Server" → "Configure".
- In the following dialog window, select "Actor mapping" in the list on left side.

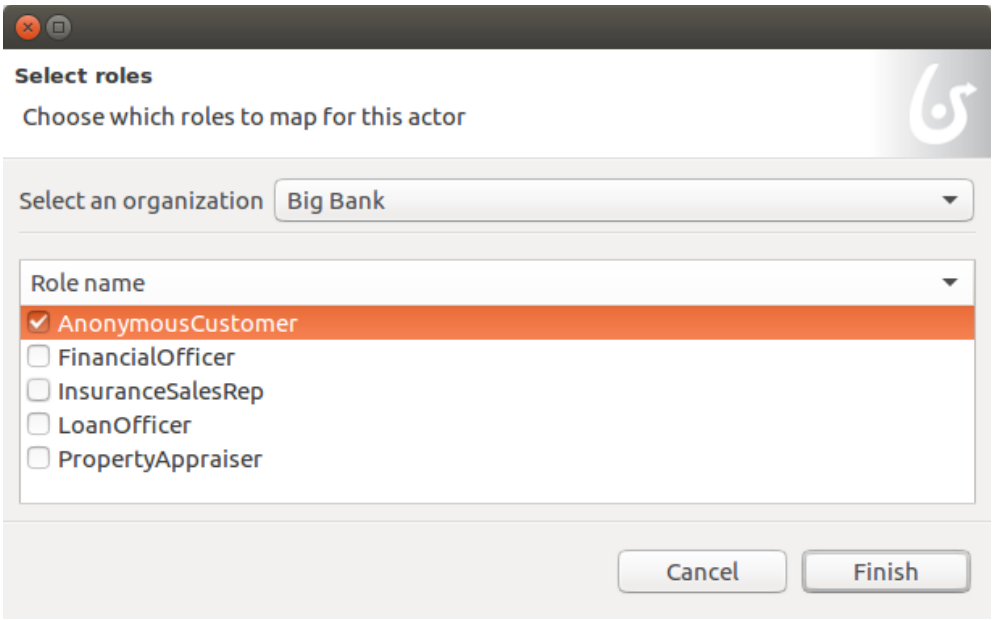


There is only one actor defined for this pool, the applicant, and Bonita tells you it is not yet mapped to any resource in the Big Bank organization that you defined and deployed on the server earlier.

- Select the "Applicant" actor, then press the "Groups ..." button.
- In the following dialog window, select the Applicants group in the Big Bank organization.

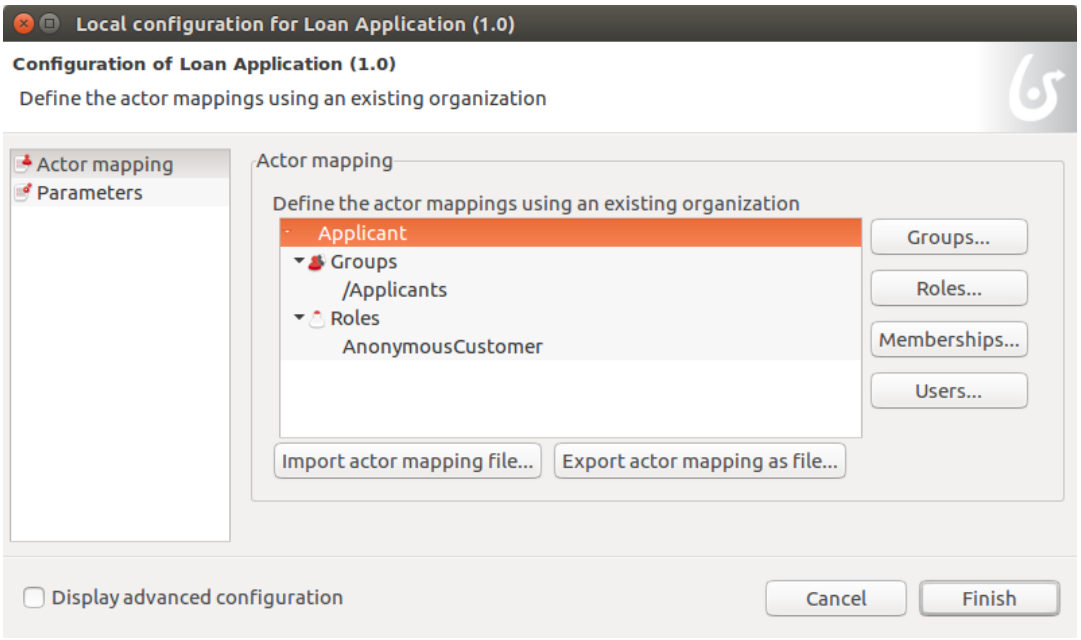


- Press the “Finish” button to complete this mapping.
- Press the “Roles ...” button to further select a role.
- In the following dialog window, select the “Anonymous Custonmer” role.



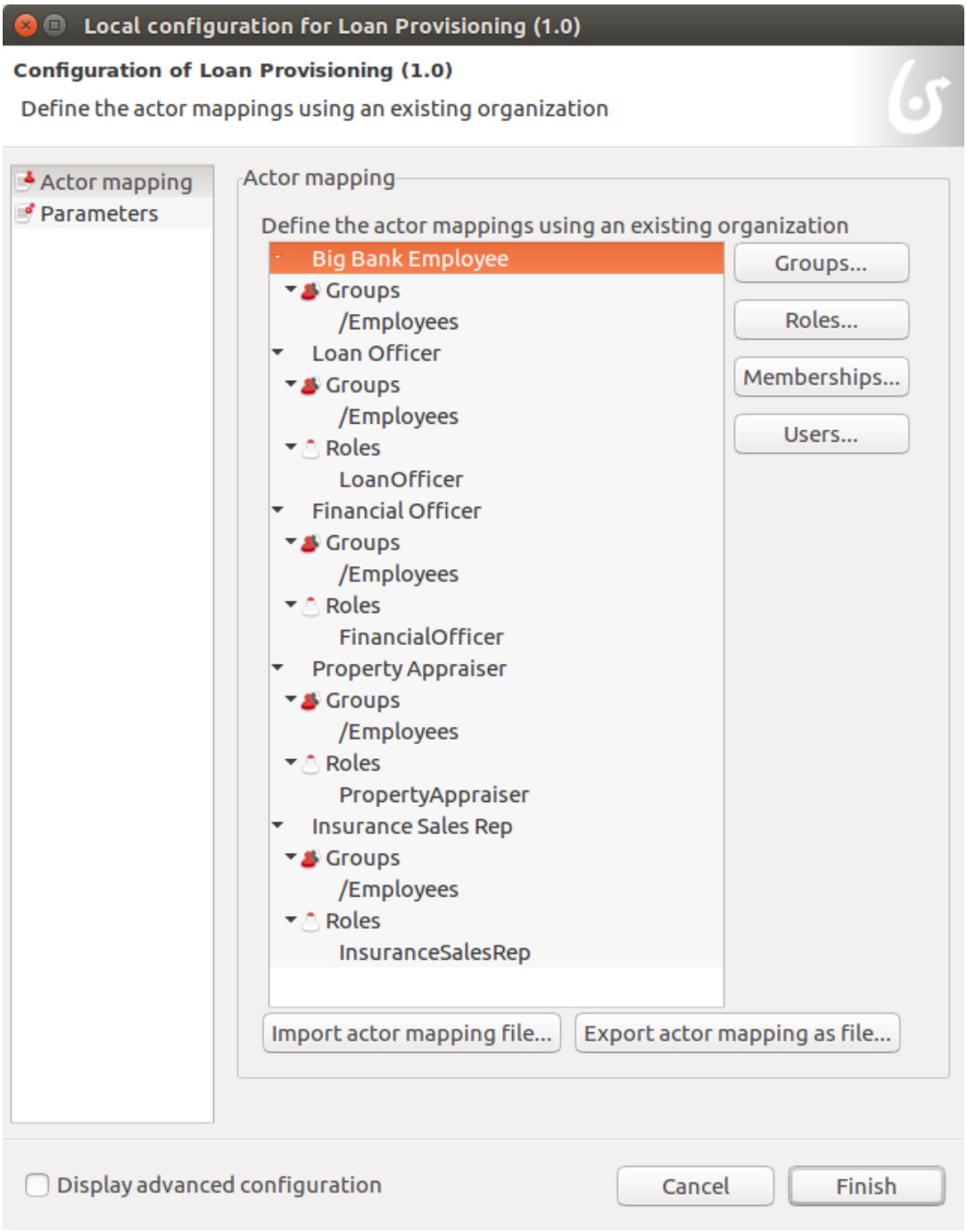
- Press the “Finish” button to complete this mapping.

The mappings you have created for the Applicant actor to the group and role should be reflected in the dialog window:

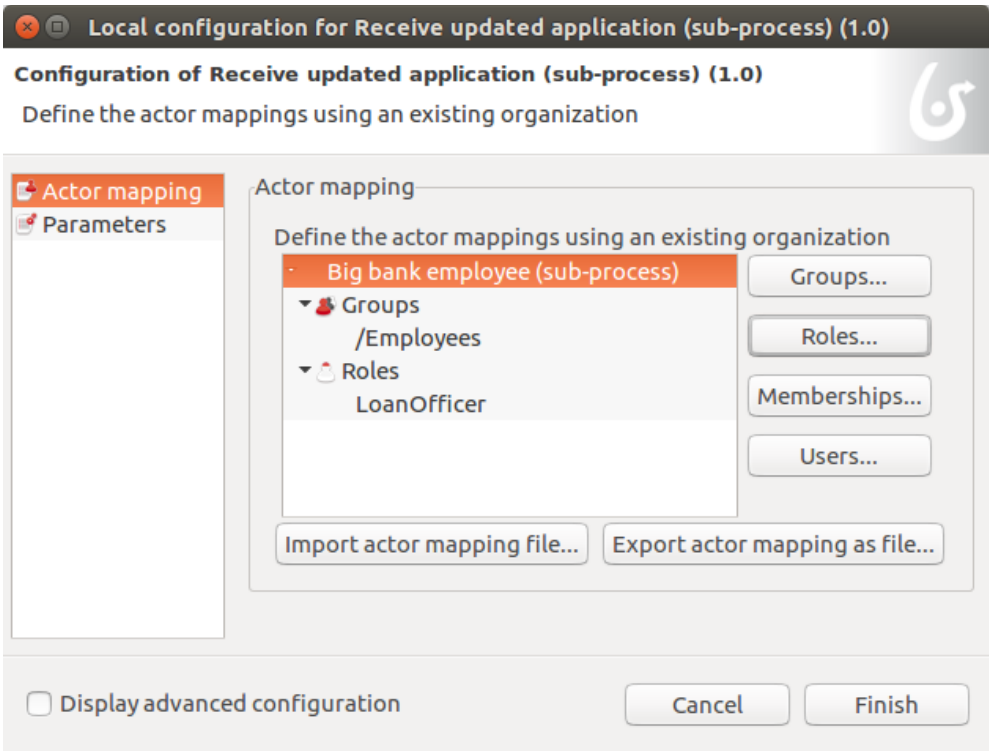


- Press “Finish” to complete the mapping for this process.
- Use the same procedure to map all actors for the remaining two pools.

The actor mapping for the loan provisioning pool process should look as follows:



The actor mapping for the receive revised loan application (sub-process) pool should look as in the following image:



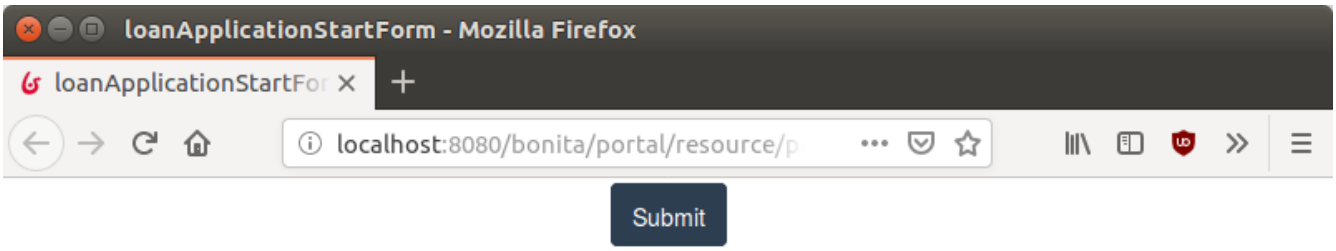
Process Execution

To launch the process on the Bonita workflow management server and to log into the Bonita Portal to work with the process, its cases, and tasks:

- Select the loan application pool in the BPMN diagram.
- In the menu, select “Server” →- “Run”.

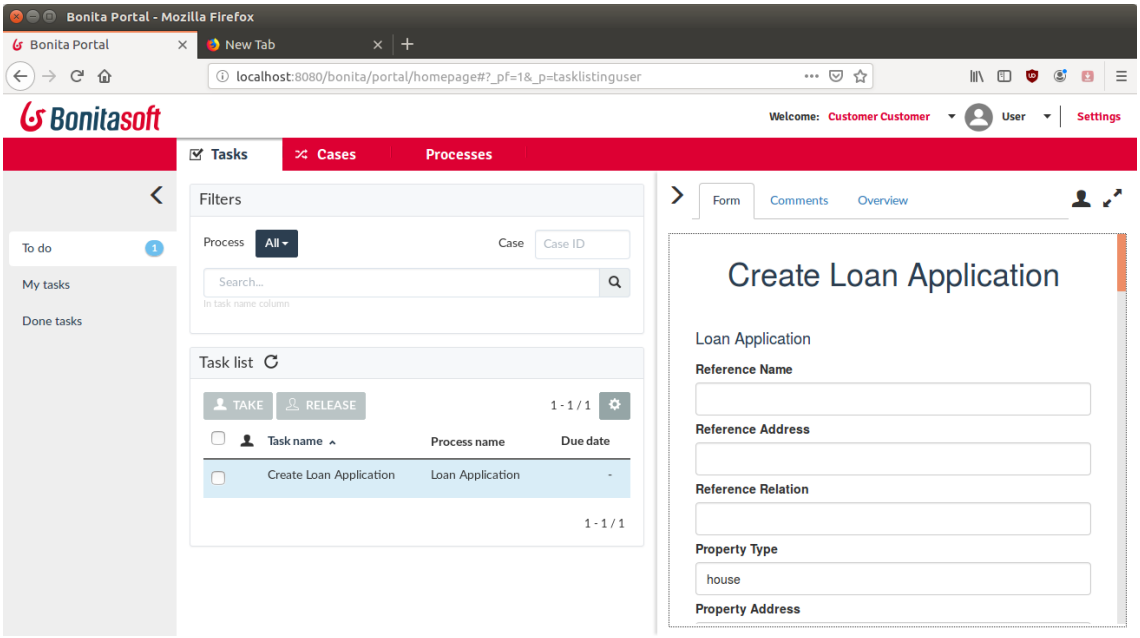
Bonita Studio will deploy the latest versions of all processes to the workflow management server, and will open a browser window where the default user for that organization is logged in to Bonita Portal. The portal page will show the instantiation form that you have defined for the pool.

In your case, you may have defined a form with only a “Submit” button as the pool instantiation form.



- Press the “Submit” button to create a new instance of the loan application process.

You will be taken to the task list where the first task is already selected and its user interface form is shown on the right side of the page.



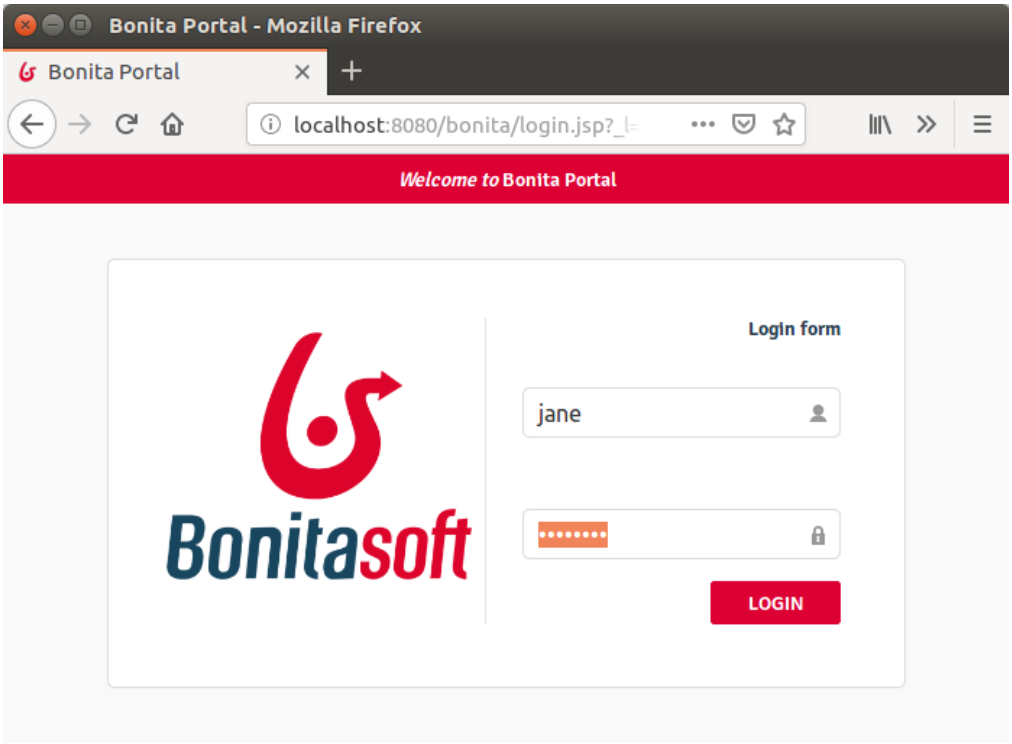
You cannot interact with the task (that is, its form) unless you accept ("take") the task. When you mouse over the form, Bonita Portal will offer you a button to take the task. Alternatively, select the task in the task list on the left, and select the "Take" action.

After you have taken the task, you can interact with the form and provide required values. Notice that some values are required (indicated by a little star next to the form label), and some values are prepopulated based on the initialization script for the loan application and applicant business variables. Fill in all required values and press the "Submit" button.

The customer's loan application process is finished after this task.

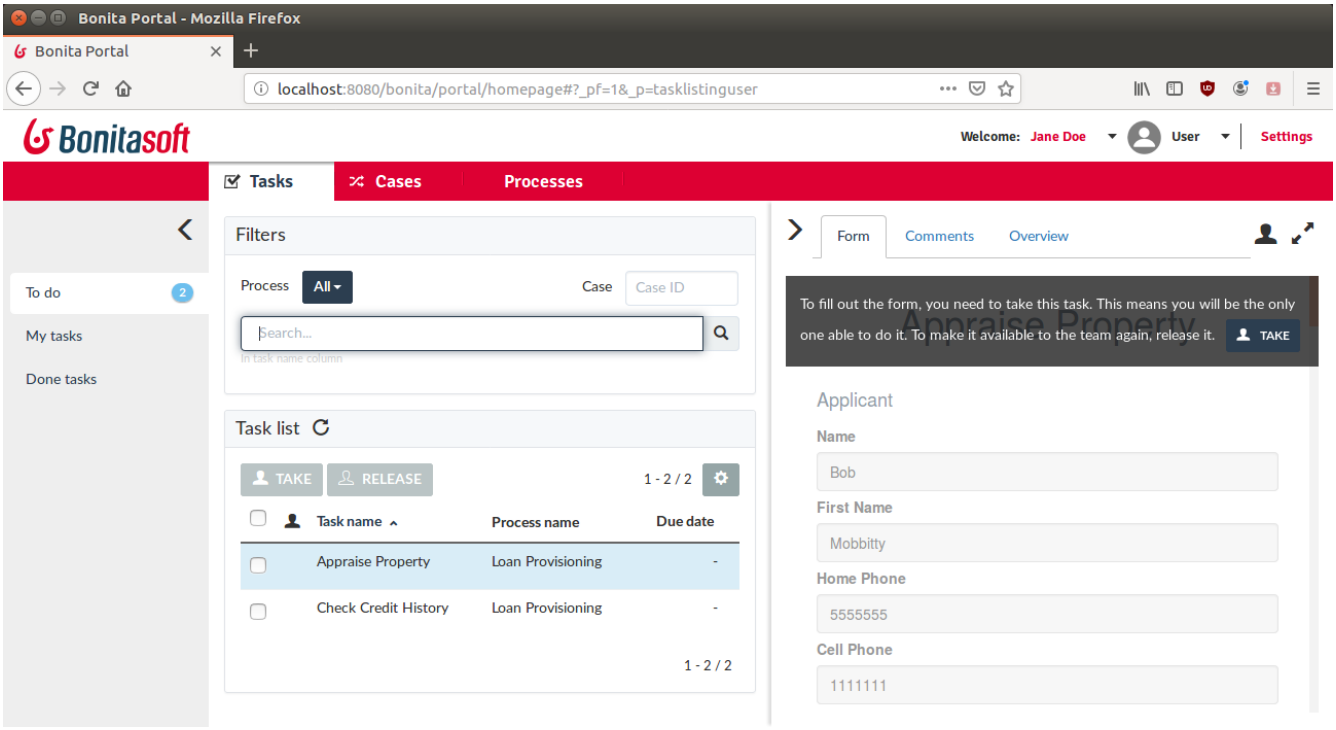
- Log out of the portal on the top right.

You will be taken to the Bonita Portal log in page.



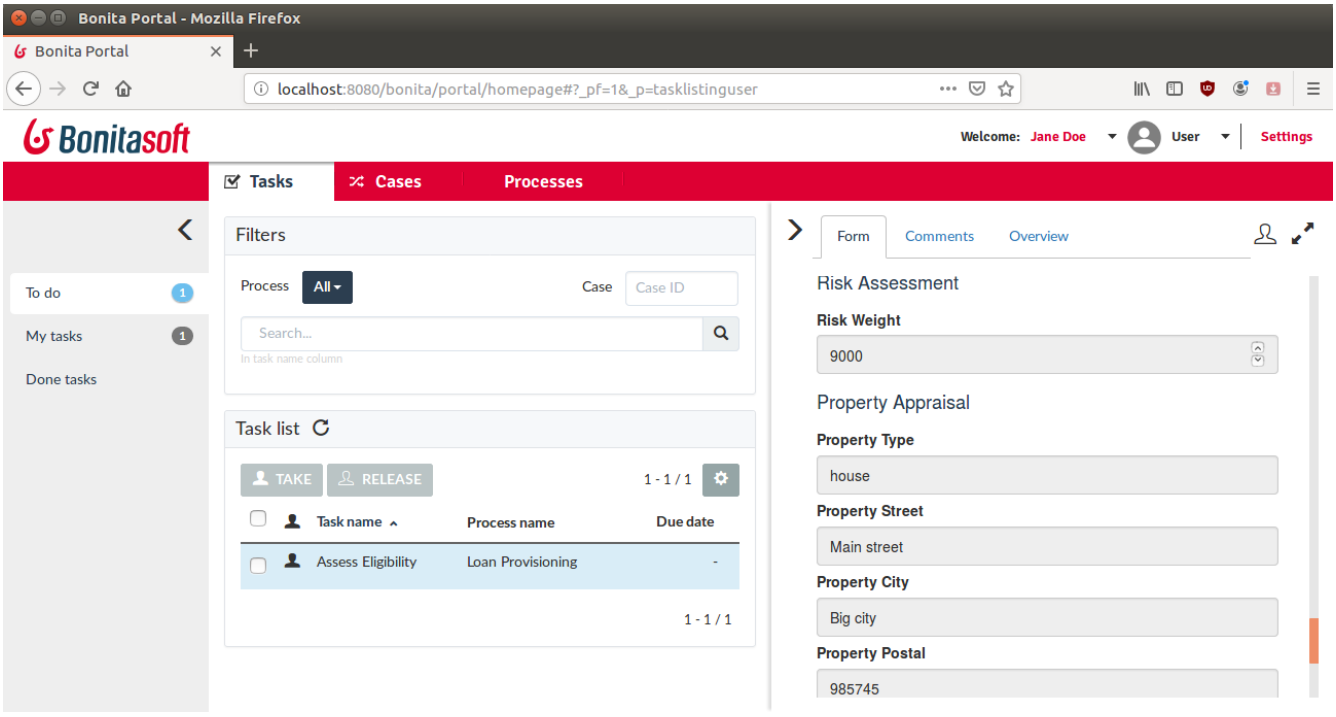
- Log in with the user name "Jane" and the password you have specified for this user when you defined the organization.

You will be presented with Jane's worklist, which contains two tasks for same case, "Appraise Property" and "Check Credit History".



You defined Jane as the only employee of Big Bank and had assigned all four roles within Big Bank to Jane. Consequently, she is able to perform tasks in any of the four lanes of the loan provisioning pool and she is the only one able to perform that work. In a realistic environment, you would define multiple users with specialized roles.

Both tasks contain the relevant information that was entered by the customer/applicant in the earlier task. Upon taking and completing both tasks, Jane will be offered the task "Assess Eligibility". After taking this task, note that the risk assess attribute value is the sum of the credit card balance and the overdue balance. Recall that this is what you specified for the script task "Assess loan risk", demonstrating that this task has been executed.



Accept this task and check the box “Eligibility” to indicate that the applicant is eligible to receive a loan for this loan application. The next task on Jane’s work list is “Prepare acceptance pack”. After you take and complete this task, “Send acceptance pack” will be in the work list, and, depending on whether you ticked the box “Insurance required”, the task “Send home insurance quote” may also be in the work list. After completing these tasks, “Verify repayment agreement” will be in the work list. Depending on whether box conditions in the agreement summary are checked (true), the final user task in the process is either “Notify Cancellation” or “Notify Approval”.

You may also want to play through the process with an incomplete loan application to ensure that the process properly handles revision of incomplete applications.

You may also wish to play through ineligible applications or applicants not agreeing with the repayment terms to ensure the process handles these cases properly.

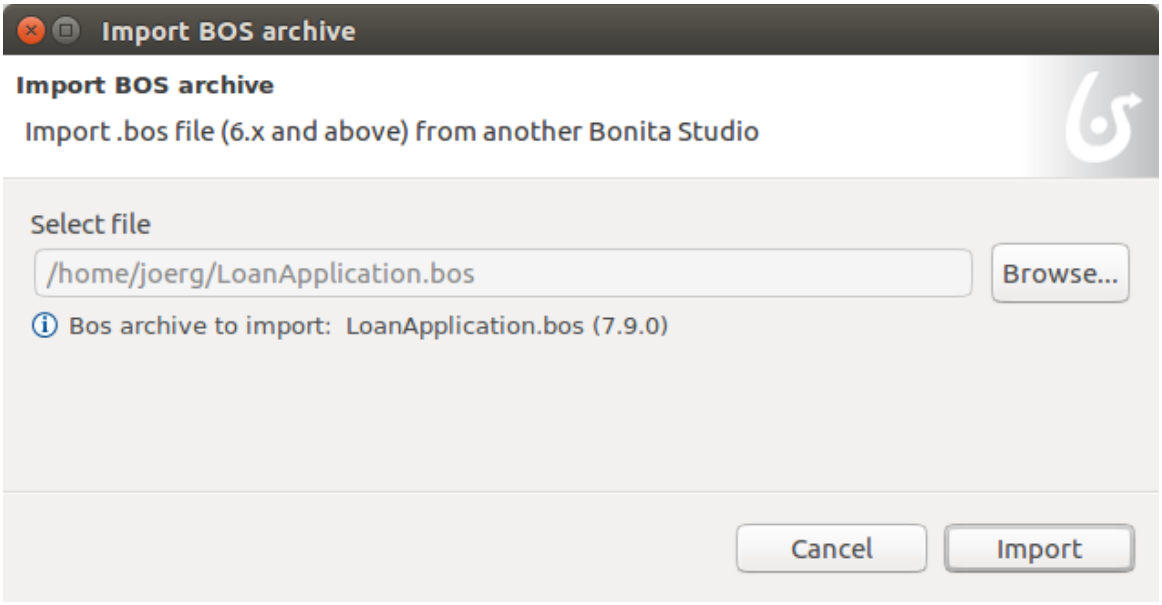
Finally, to test the timers, you should set them to a shorter duration, on the order of minutes, so that testing can be done quicker.

Appendix: Importing and Deploying the BOS Archive File

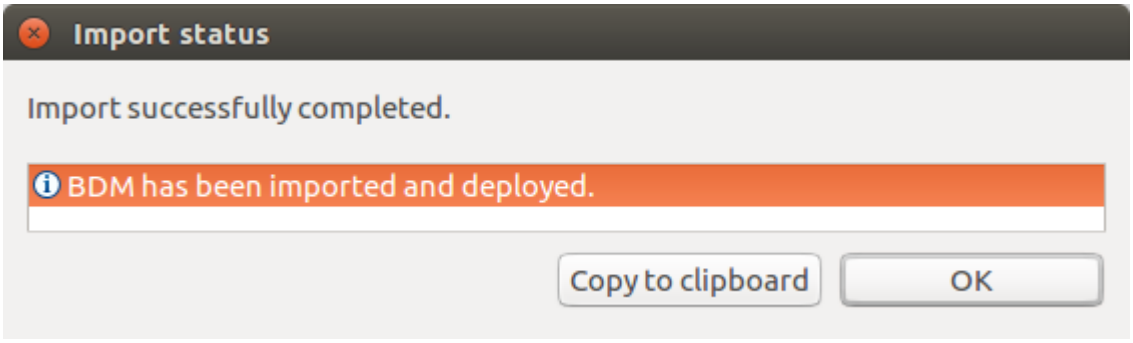
The completed process model, data definitions, and organizational definitions are also provided as a BOS archive file, the project export format of Bonita Studio. BOS archives can be easily imported into a new installation of Bonita.

After you have installed Bonita Community edition,

- Select “File” → “Import” → “BOS archive ...” from the menu
- Select the “LoanApplication.bos” archive file
- Press the “OK” button to begin the import



After a little while and a number of import operations, Bonita Studio will report successful import:



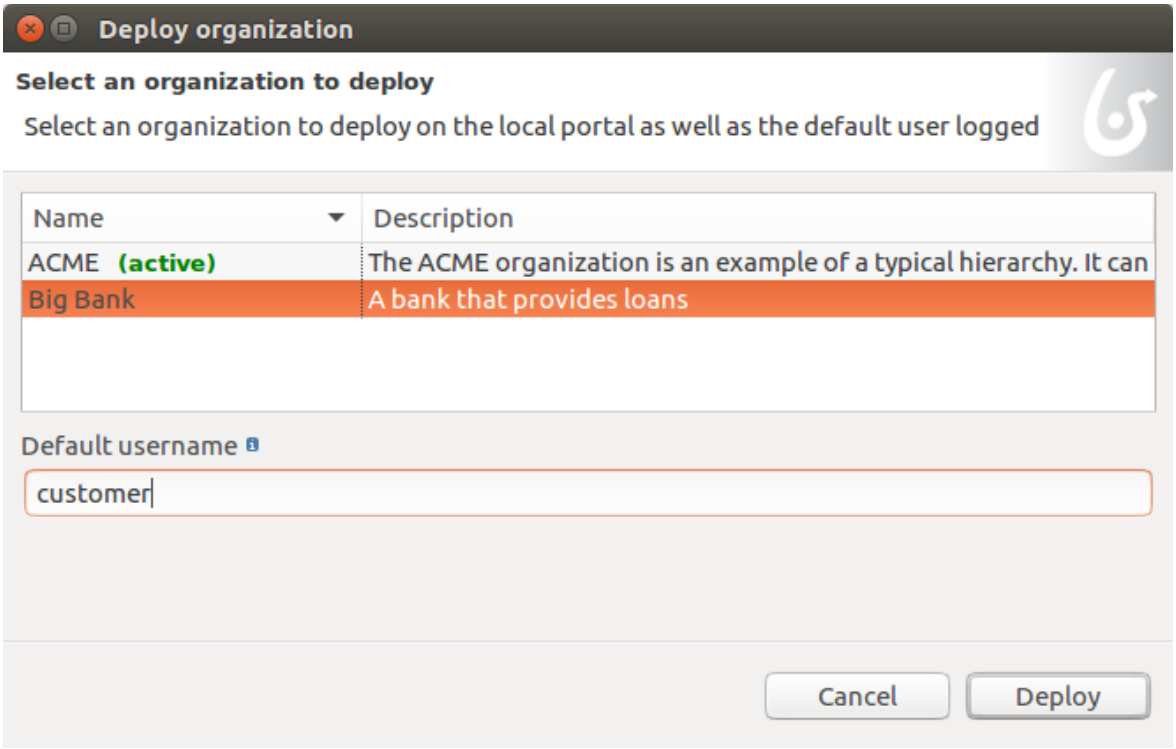
Note: Should Bonita warn of any conflicts during import, select to overwrite any existing files.

Next, you must deploy the Big Bank organization defined in this project to the workflow management server and provide a default user name for the Bonita Portal.

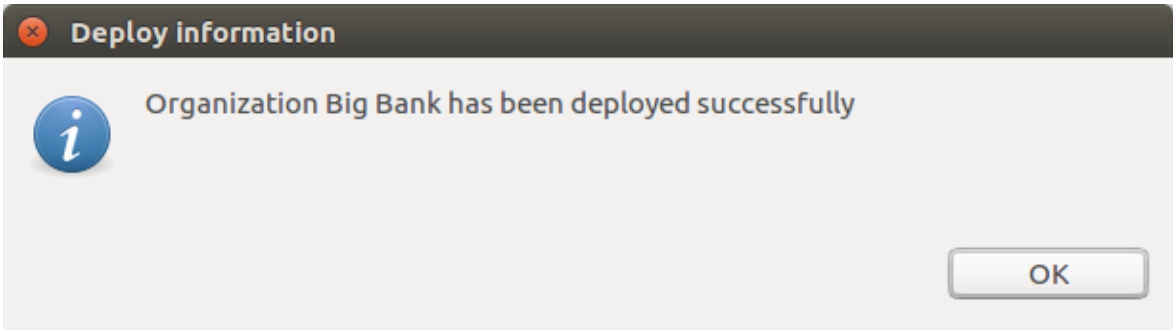
- Select “Organization” → “Deploy ...” from the menu

In the following dialog window:

- Select the “Big Bank” organization
- Enter “customer” for the default user name
- Press the “Deploy” button



Bonita will acknowledge successful deployment of the organization:



You are now ready to run the process (see the above section on executing the process)