TECHNICAL MANUAL

SUBMITTED BY: AAKASH VISHWAS JADHAV(N10726888)
AKASH VANVARIA (N10739530)
HARSHIT TYAGI (N10644571)
URUSA ZACK(N10667814)

Contents

INTRODUCTION

This guide is designed for developers and analysts. It includes a complete overview of the configuration, operation, and execution of the loan automation process that has been automated using "codelets" and "services." In addition, users must become acquainted with the installation of Yawl, the IntelliJ IDE, and the Java environment setup. There are various advantages in automating a business process, such as standardizing operations, improving efficiency, streamlining business processes, reducing errors, and providing a better customer experience. For example, the former loan application process for a home loan involved several repetitive processes and dependencies among various bank stakeholders, resulting in delays, human errors, and low consumer satisfaction. Business process automation is used to automate repetitive mundane tasks, which can assist organizations in streamlining their processes, improving efficiency, and providing a better customer experience.

This document provides a brief description of modeling the loan application and assessment process using YAWl editor. The process models fully describe the control flow of the home loan application and assessment process. This documents also mention the data variables which are formed and defined during the entire process. In addition, specific tasks have been automated by developing the relevant java codelets and set of services. Further, the document also provides a resource perspective that mentions each stakeholder performing their roles involved in this process.

The details on Yawl and its installation, inbuilt methods, and services can be found in the Yawl User manual, which is available for download from https://yawlfoundation.github.io/The development, dependencies, executions, and deployment of numerous artifacts for automating the home loan application process have been discussed in depth throughout this manual as needed.

PROCESS MODEL

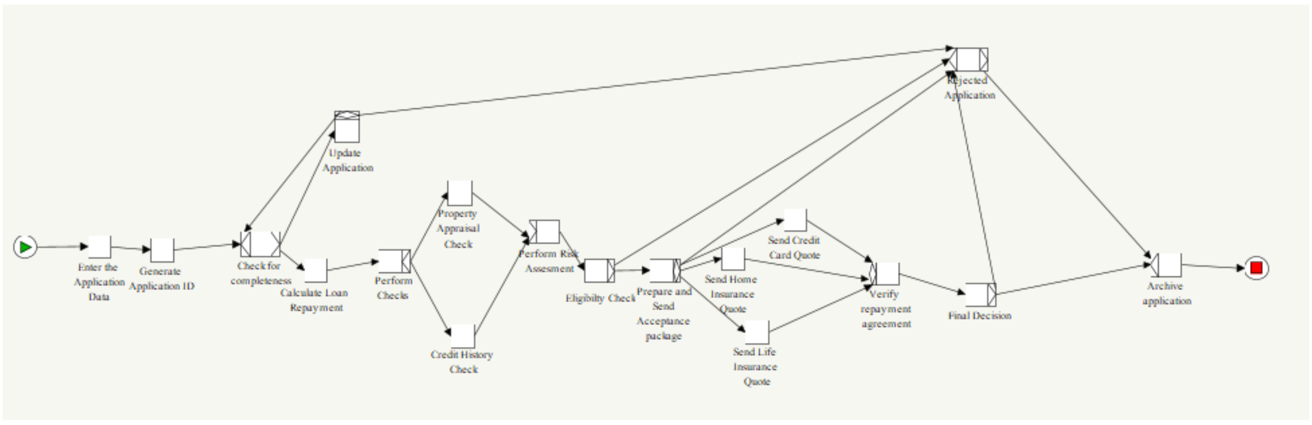


Figure 1: Home loan application & assessment process

The process model presented above is the graphical representation of the home loan application and assessment process control flow. It is created in YAWL editor 4.5.7 and thoroughly details the control flow of the home loan and assessment process from start to end.

TASK VARIABLES

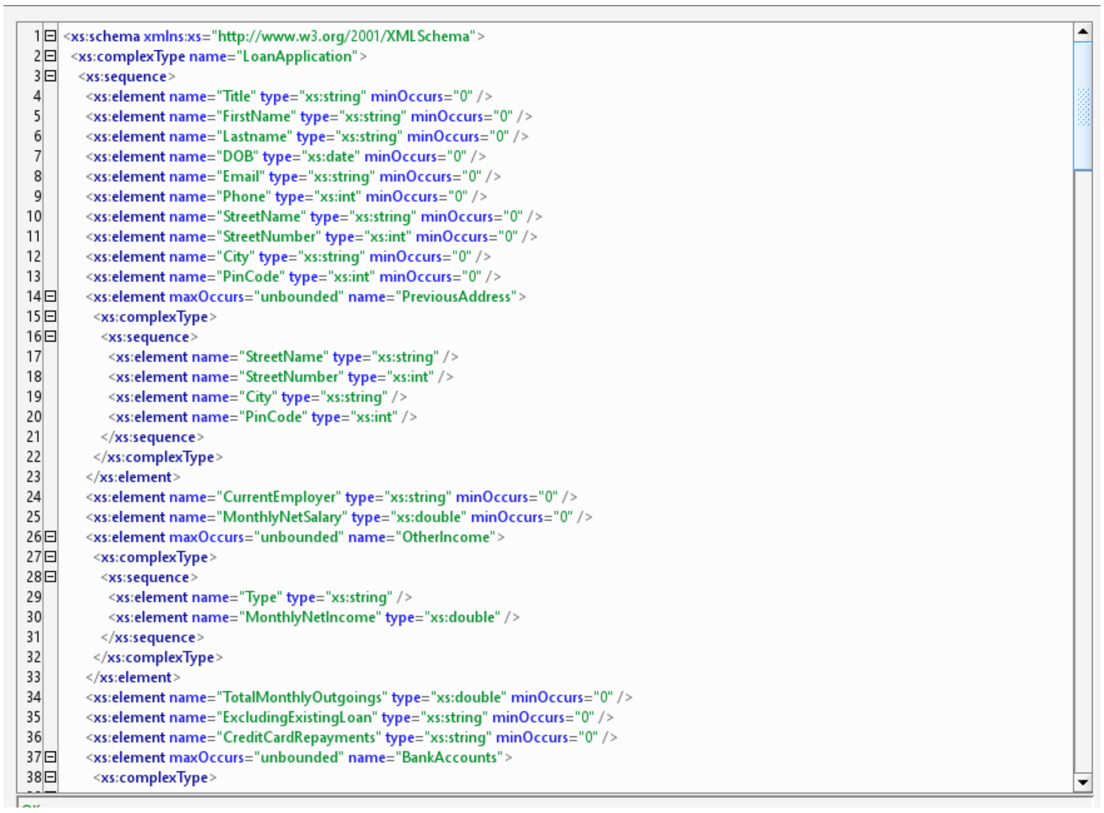
The task variables of the above process models are described in detail as follows.

Enter the Application data	Once the applicant submits a loan application, the process begins. First, a loan officer enters the applicant's information into the system in the "Enter the Application data" task.
Generate Application ID	Once the application is received, a unique application number is generated based on the applicant's provided information, and the application is registered into the system.
Check for completeness	The loan officer reviews the application for completeness. If the application is incomplete, the applicant is notified and asked to provide the missing details.
Update Application	The loop continues until the applicant provides the requested information to the bank and the loan officer confirms that the application is complete. Then, the application is eligible for further processing.
Cancel Application	If the applicant does not respond to the request for the requisite missing information within five working days, the loan application is canceled or it can also be used by the bank employees, to reject the loan application.
Perform Checks	The “perform checks” task begins the loan evaluation process with two tasks: a credit history checks and a property appraisal check.
Property Appraisal Check & Credit History Check	The next step involves a property appraisal and a credit history check, which results in the Perform Risk Assessment task.
Perform Risk Assessment	The risk weightage is automatically calculated using the previous task's outcomes: property appraisal checks, and credit history checks by using a “Risk Assessment” external service.

Eligibility Check	In the task Eligibility check, the loan officer formally evaluates the applicant's loan eligibility based on various factors such as the application, credit report, property appraisal, and risk assessment.
Cancel Application	Following the Eligibility check, if the application fails eligibility, the application is rejected, and the application is archived, and the process is ended.
Prepare and Send Acceptance package	An acceptance package, including a repayment agreement, is prepared, printed, and sent to the applicant if the application is accepted after the eligibility check.
Verify repayment agreement	The repayment agreement specifies a monthly payment amount as well as a repayment schedule. The applicant must agree to both conditions. In this event, the applicant must sign and return the agreement to the bank within 14 days.
Send Home Insurance Quote	Suppose an applicant expresses an interest in a house insurance plan at the time of application. In that case, a discounted plan for 12 months is provided separately to the applicant by the sales representative.
Send Life Insurance Quote	Suppose an applicant expresses an interest in a life insurance plan at the time of application. In that case, a discounted plan for 12 months is provided separately to the applicant by the sales representative.
Send Credit Card Quote	Suppose an applicant expresses an interest in a credit card at the time of application. In that case, a credit card quote discounted plan for 12 months is provided separately to the applicant by the financial officer.
Cancel Application	Suppose the applicant does not respond with a signed agreement to the bank within 14 days of receiving the repayment agreement, and the two conditions the application is canceled. In that case, the application is archived, and the process is ended.
Final Decision	If the applicant submits a signed agreement accepting the two conditions to the bank within 14 days, a different loan officer validates it and provides a final decision of approval. Thus, the application is successful.
Archive application	Once the application is successful, it is archived into the database before the process instance completes.

DATA SCHEMA AND VARIABLES

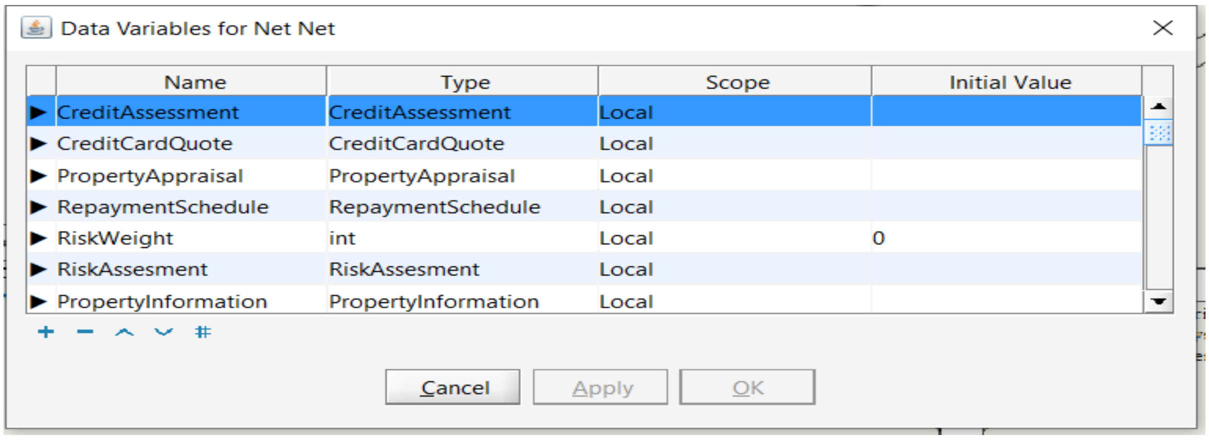
To create user defined data-schema, select the data definitions property in the specification part of the properties pane. In the data definitions box, enter the XML schema data type (For Example- In figure 3, Loan application is defined as a complex type, which further consists of the elements like Title, FirstName, DOB etc.); if any errors are identified, the bottom pane will display the error details. After that, click the Done button. After you've defined the data type, you can use it to create net and task variables by clicking on the data variables button on “properties” tab, as we can see the complex type variables can now be used as a user-defined data types shown in figure 4.



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="LoanApplication">
  <xs:sequence>
   <xs:element name="Title" type="xs:string" minOccurs="0" />
   <xs:element name="FirstName" type="xs:string" minOccurs="0" />
   <xs:element name="Lastname" type="xs:string" minOccurs="0" />
   <xs:element name="DOB" type="xs:date" minOccurs="0" />
   <xs:element name="Email" type="xs:string" minOccurs="0" />
   <xs:element name="Phone" type="xs:int" minOccurs="0" />
   <xs:element name="StreetName" type="xs:string" minOccurs="0" />
   <xs:element name="StreetNumber" type="xs:int" minOccurs="0" />
   <xs:element name="City" type="xs:string" minOccurs="0" />
   <xs:element name="PinCode" type="xs:int" minOccurs="0" />
   <xs:element maxOccurs="unbounded" name="PreviousAddress">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="StreetName" type="xs:string" />
      <xs:element name="StreetNumber" type="xs:int" />
      <xs:element name="City" type="xs:string" />
      <xs:element name="PinCode" type="xs:int" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="CurrentEmployer" type="xs:string" minOccurs="0" />
   <xs:element name="MonthlyNetSalary" type="xs:double" minOccurs="0" />
   <xs:element maxOccurs="unbounded" name="OtherIncome">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Type" type="xs:string" />
      <xs:element name="MonthlyNetIncome" type="xs:double" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="TotalMonthlyOutgoings" type="xs:double" minOccurs="0" />
   <xs:element name="ExcludingExistingLoan" type="xs:string" minOccurs="0" />
   <xs:element name="CreditCardRepayments" type="xs:string" minOccurs="0" />
   <xs:element maxOccurs="unbounded" name="BankAccounts">
    <xs:complexType>
```

Fig 1: Data Schema

The data variables can be defined as input-output, input or output depending on their use in a task. When a variable is defined as an input, it can only receive a value from a previous task and is “read-only”, In case of output scope, the value of a variable can be altered externally, and input-output allows the task to change the value of variable and forward it to the next task.



Data Variables for Net Net

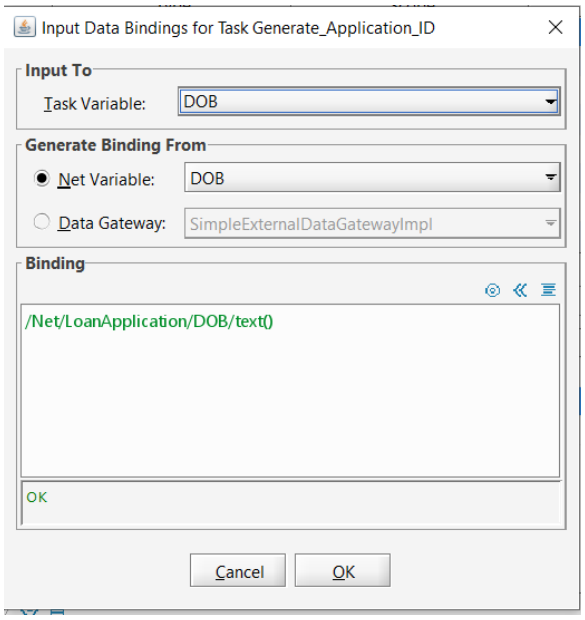
Name	Type	Scope	Initial Value
CreditAssessment	CreditAssessment	Local	
CreditCardQuote	CreditCardQuote	Local	
PropertyAppraisal	PropertyAppraisal	Local	
RepaymentSchedule	RepaymentSchedule	Local	
RiskWeight	int	Local	0
RiskAssesment	RiskAssesment	Local	
PropertyInformation	PropertyInformation	Local	

Fig 2: Data Variables for Net

DATA BINDING AND SPLIT PREDICATES

In Data binding process, each variable is bounded to other variable, depending on the scope and their use. The data binding is performed in two ways, first one being the input binding and output binding. The input data bindings dialogue box appears when you click the input bindings button ( ) on the lower toolbar. In the input binding window, all the net variables are listed in the net variable drop down box, and in the generate binding from section, all the task variable(s) are listed in the task variable drop down box. As illustrated in figure 4, make sure both the net and task variables are chosen, and then add the XQuery in one of two ways: 1.) click the generate and insert binding button, or 2.) type the XQuery straight into the XQuery field, then click the OK button. In the data variable dialogue, click the fast view bindings button to examine the binding you just created.

Creation of output binding is followed in a similar way by clicking on output binding icon.



Input Data Bindings for Task Generate_Application_ID

Input To

Task Variable: DOB

Generate Binding From

● Net Variable: DOB

○ Data Gateway: SimpleExternalDataGatewayImpl

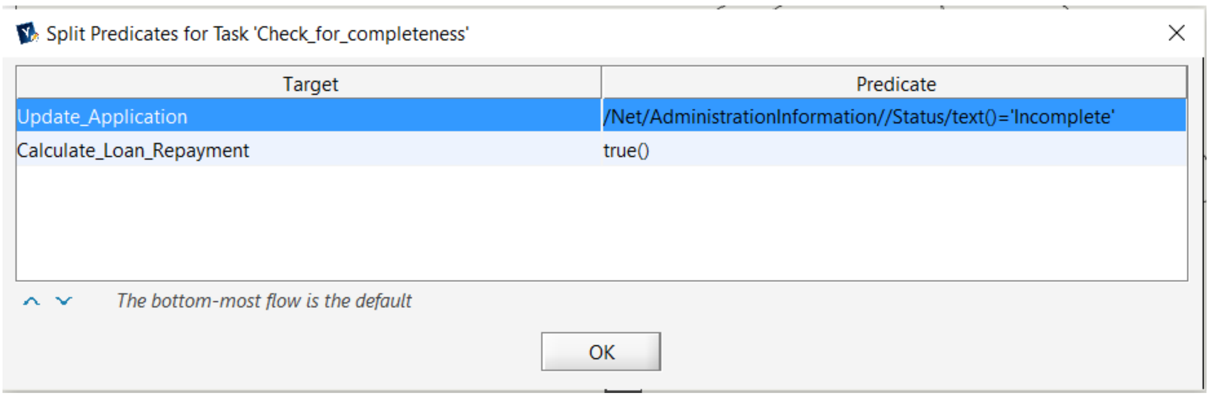
Binding

/Net/LoanApplication/DOB/text()

OK

Fig 3: Data Binding Window

At every decision point, such as OR splits, XOR splits. We have specified, which flow should be engaged at runtime, to add predicates to the decision nodes, Select the split predicates property in the task part of the properties pane, then click the action button (to the right of the property) to open the split predicates dialogue box by clicking on the check application form completion task. In the split predicates dialogue box that displays, all the split's outgoing flows, as well as the other accompanying flow expressions (predicates), will be listed. For example, In figure 4, We have defined predicates for "Check_for_completeness", the control flow to update_application is set to trigger by defined X:Path, so that it will only be activated, when the administrator sets the status as "incomplete", the "Calculate_loan_Repayment" task is set as true, which means it is a default path and will have control flow for rest of the cases.


Split Predicates for Task 'Check_for_completeness'

Target	Predicate
Update_Application	/Net/AdministrationInformation//Status/text()='Incomplete'
Calculate_Loan_Repayment	true()

The bottom-most flow is the default

Fig 4: Split Predicates

PROCESS DEVELOPMENT & AUTOMATION

From the process model shown in Fig 1.1, the following are the four tasks that have been automated.

Sr.	Automated Task	Automation Type	Artifacts
1.	Generate Application ID	YAWL Codelet	Application Identifier Generation
2.	Perform Risk Assessment	YAWL Service	Risk Assessment Service
3.	Calculate Repayment Amount	YAWL Codelet	Repayment Calculation
4.	Archive application	YAWL Service	Archiving Service

The four processes automated in the home loan application and assessment process are achieved by creating yawl codelets and yawl services. The steps in the development of yawl codelets and yawl services are as follows.

CODELET

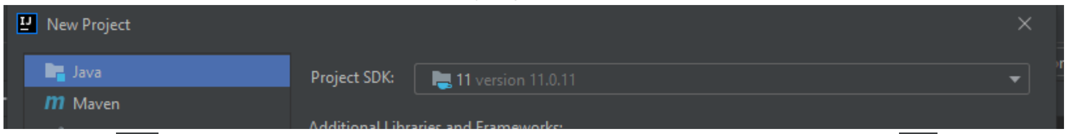
A Yawl codelet is a Java class developed to perform a particular action that represents the work of a work item record. It is a framework intended to carry out fully contained, short-term tasks. A Yawl codelet is a sub-component of a resource service with its thread. By default, the Resource service handles the running codelets instances.

DEVELOPING A YAWL CODELET

The following is the process for developing a YAWL codelet. We will design a Yawl codelet to automate the task of "repayment amount calculation" in our home loan application and walk through the codelet development process. We will be building the Yawl codelet with IntelliJ Idea (IDE), Yawl Editor, and Java 11 environment.

Steps:

1) Open the IntelliJ IDE and create a new Java project.



Project SDK: 11 version 11.0.11

2) Go to setting > Project Structure >Modules >Dependencies > Click on Add and select the yawlib folder.

   yawlib can be found in [yawl installation]\engine\apache-tomcat-7.0.65\yawlib.
3) Create a new package and create a new Java Class in it named "Repay code" under the package structure
4) Once the class is created, extend the' AbstractCodelet,' which is necessary for a codelet creation.

```
5) public class Repay_code extends AbstractCodelet {
```

   AbstractCodelet is a method called when an external entity such as a Yawl editor requests Override to populate the list. Each parameter includes a minimum of name, data type, and description

6) We can set a description for our class by creating a constructor, and with the helper method, we inherit "setDesription" and set a description for our class.
7) The next step is to implement an execute method that provides a framework of a method to build.

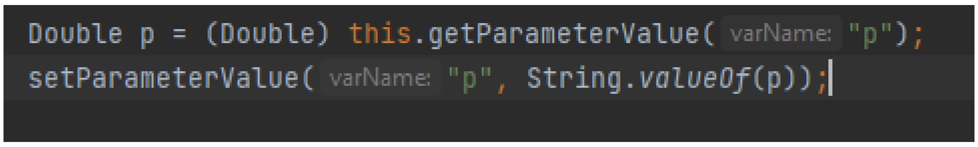
```
public Element execute(Element inData, List<YParameter> inParams, List<YParameter> outParams) throws CodeletExecutionException {
    this.setInputs(inData, inParams, outParams);
```

8) "inData" gives us the "input data," "inPutparams" provides us with the input variables names and their types, the "Ouparams" helps us to set the output parameters needed.
9) We can extract the values in "inData" with the help of a helper method, i.e., the "setInputs." It will provide the other methods to access the inputs.

```
this.setInputs(inData, inParams, outParams);
```

10) Next, we need to fetch our input values by calling the method "getParametrValue."
    The names of the variables are the actual variables that exist in our task data with their respective data type. Get parameter value will extract the parameter value from the XML element.

So, in our codelet, this is "p," "r," and "n" that we are fetching to calculate the repayment amount.



```
Double p = (Double) this.getParameterValue( varName: "p");
setParameterValue( varName: "p", String.valueOf(p));
```

11) We do the calculation for our Loan repayment amount here for this codelet.

```
12)     double a = p * (r * Math.pow((1 + r), n) / Math.pow((1 + r), n) -
   1);
```

13) Again, we create a variable for our output that is " a" that will be the output data for the task the result into the "getOutputData()."
14) After doing this, we need to create a method "getRequiredParams()" that allows us to define input and outputs variables. When the codelet is used in the Yawl editor, it automatically creates the variables for the task. It will give a list of parameters for the inputs and outputs
15) Next, we need to create decompositions for the variables and set the type, i.e., input or output.

```
16)     public List<YParameter> getRequiredParams() {
        List<YParameter>parameters= new ArrayList<>();
        YParameter parameter = new YParameter((YDecomposition)null, 0);

   parameter.setDataTypeAndName("double","p","http://www.w3.org/2001/XMLSch
   ema");
        parameter.setDocumentation("Enter the amount of loan requested:");
        parameters.add(parameter);
```

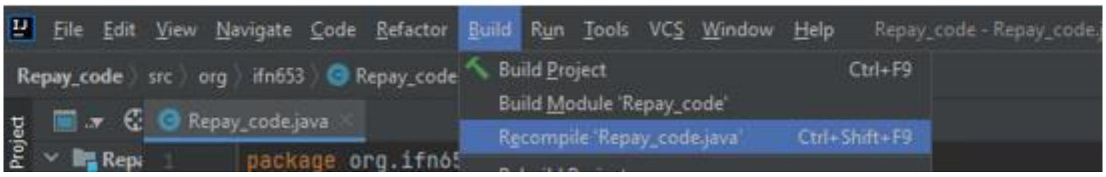
17) Then we need to set up the data type and name for the variables. 0 refers to INPUT, and 1 refers to OUTPUT for the variable as follows.

```
18)     YParameter parameter = new YParameter((YDecomposition)null, 0);
   parameter.setDataTypeAndName("double","p","http://www.w3.org/2001/XMLSch
   ema");
```

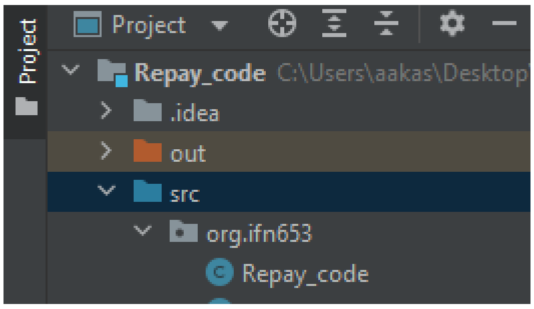
19) Now we finally return the list of parameters to YAWL engine.

```
Return parameters;
```

20) Now the codelet is ready, and we need to compile the codelet.


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Repay_code - Repay_code
Repay_code src org ifn653 Repay_code
Build Project Ctrl+F9
Build Module 'Repay_code'
Recompile 'Repay_code.java' Ctrl+Shift+F9
Repay_code.java
Project
Repay
package org.ifn65

21) After compiling the codelet, we can see the "repay_code" class under our package structure.

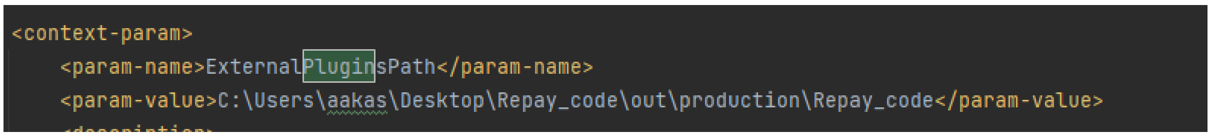


22) Now we have finally developed and compiled the codelet.

23) The next step is to define a path to the class file generated, by editing the “WEB.XML” file present in “Resource service” folder located inside yawl installation folder.

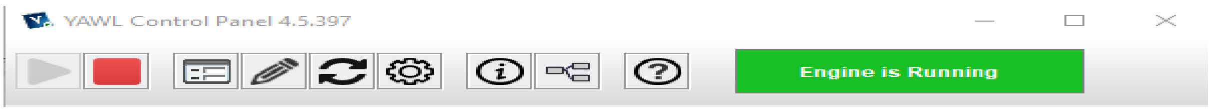
DEPLOYING A YAWL CODELET

1) For deploying the Yawl codelet, we need to follow the below steps.
2) We need to configure The Deployment Descriptor File (web.xml)
3) Go to the Yawl distribution i.e. [yawl installation]\engine\apache-tomcat-7.0.65\webapps\resourceService\WEB-INF\web.xml
4) Next step is to open the XML file and find look for the tag <param-name> ExternalPluginsPath </param-name>.Here we need to set up the path for our codelet as follows.

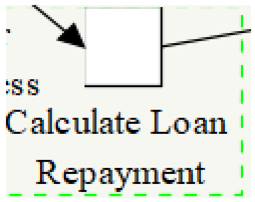


```
<context-param>
    <param-name>ExternalPluginsPath</param-name>
    <param-value>C:\Users\aakas\Desktop\Repay_code\out\production\Repay_code</param-value>
```

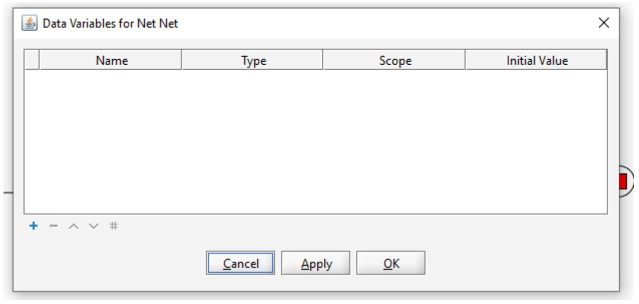
5) This path needs to match the codelet project structure. This path will tell the yawl installation where to find our codelet.
6) Then we start the yawl engine and open the Yawl editor.



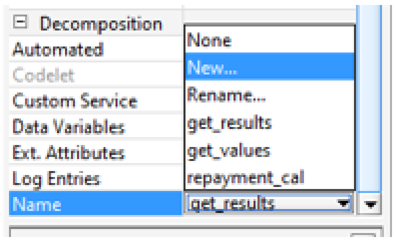
7) Next, we go to the Yawl editor and open our process model. In this case, we have developed the codelet for the task "Calculate Loan Repayment."



8) Now we need to create the data variables for the codelet. We can do this by selecting our codelet in the process model. Then go to data variables and generate the data variables.

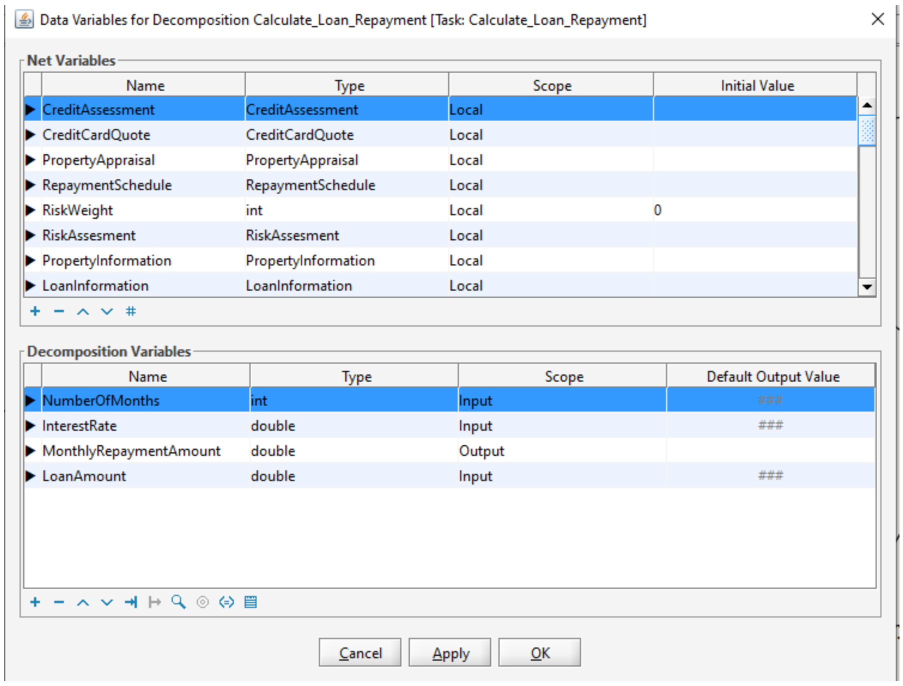


9) Now need to declare the variables for the net and add the variables
10) After declaring the variables, we need to create the decomposition for the variables and create the mapping.
11) Next, we need to go to decomposition and create a new decomposition for the tasks.



⊟ Decomposition	
Automated	
Codelet	
Custom Service	
Data Variables	
Ext. Attributes	
Log Entries	
Name	get_results

12) In our case, the decomposition is completed as follows:



Data Variables for Decomposition Calculate_Loan_Repayment [Task: Calculate_Loan_Repayment]

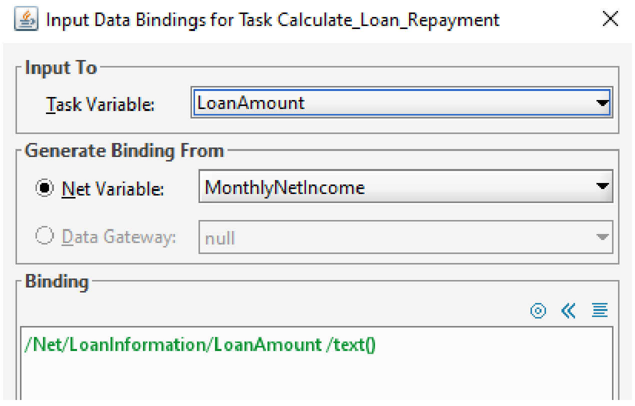
Net Variables

Name	Type	Scope	Initial Value
CreditAssessment	CreditAssessment	Local	
CreditCardQuote	CreditCardQuote	Local	
PropertyAppraisal	PropertyAppraisal	Local	
RepaymentSchedule	RepaymentSchedule	Local	
RiskWeight	int	Local	0
RiskAssesment	RiskAssesment	Local	
PropertyInformation	PropertyInformation	Local	
LoanInformation	LoanInformation	Local	

Decomposition Variables

Name	Type	Scope	Default Output Value
NumberOfMonths	int	Input	###
InterestRate	double	Input	###
MonthlyRepaymentAmount	double	Output	
LoanAmount	double	Input	###

13) The values are fetched from the loan application form that the applicant submits to the bank.
14) The binding for inputs and outputs needs to be completed for all the variables, as shown below.



Input Data Bindings for Task Calculate_Loan_Repayment

Input To

Task Variable: LoanAmount

Generate Binding From

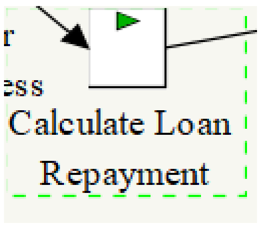
[✓ Net Variable: MonthlyNetIncome]

Data Gateway: null

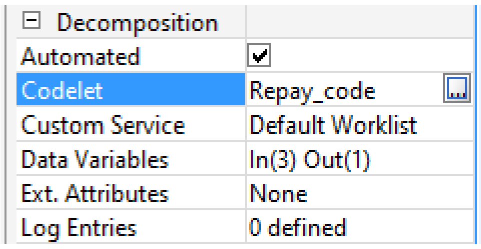
Binding

/Net/LoanInformation/LoanAmount /text()

15) Once the mapping is completed for the variables, we need to automate the task. This is done by selecting the particular task and click on the check box automated.
16) When we check the box for automated, we get a play button on the "Calculate Loan Repayment."



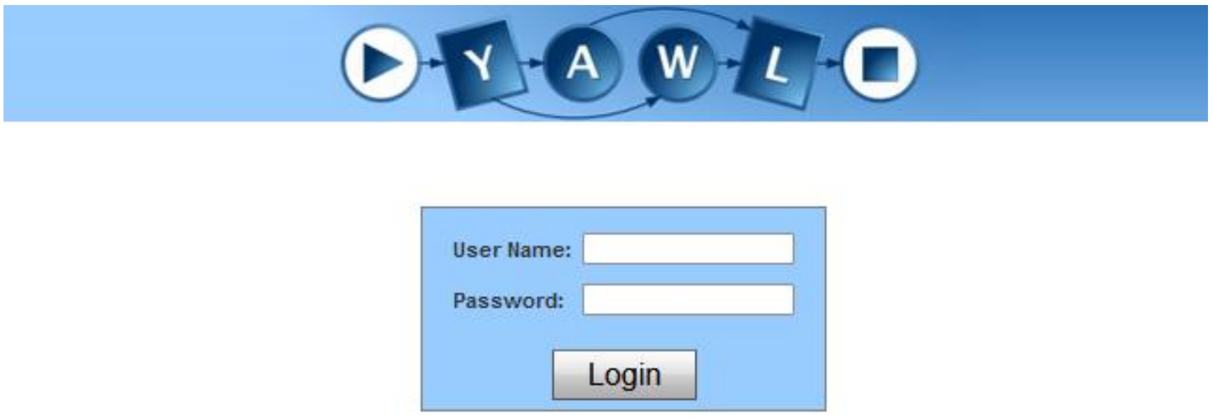
17) The next step is to select the codelet we created



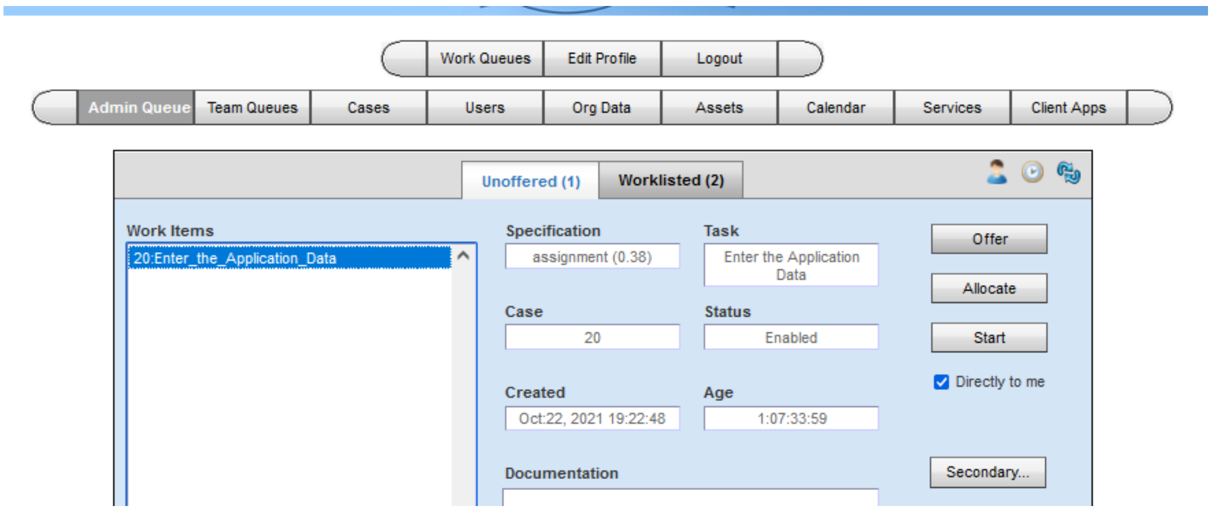
Decomposition	
Automated	✓
Codelet	Repay_code
Custom Service	Default Worklist
Data Variables	In(3) Out(1)
Ext. Attributes	None
Log Entries	0 defined

18) After selecting the codelet, the play button will turn green in color, and we are ready to push the codelet to the yawl engine.

19) The next step is to upload the model to the Yawl engine by clicking on the upload button.
20) After clicking on the upload button, we can launch the yawl logon page.



21) Login to the Yawl login page with the credentials and log in.
22) Once we login into the admin queue, we can see our workitem in the unoffered list.



Work Items: 20:Enter_the_Application_Data

Specification: assignment (0.38)

Task: Enter the Application Data

Case: 20

Status: Enabled

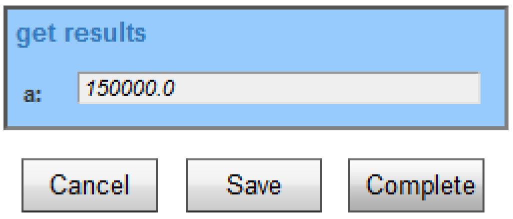
Created: Oct:22, 2021 19:22:48

Age: 1:07:33:59

☑ Directly to me

Documentation

23) We can select the checkbox "directly assign to me" or we can offer it to an employee, who is responsible for this task and click on start.
24) Then if we go to work Queues,> started> we can see view our workitem and select "view."
25) The required inputs for the calculation of loan repayment amount will be fetched, and a final calculated repayment amount will be calculated using the codelet as follows.



a: 150000.0

The complete java code for this codelet will be included as an appendix. The same development and deployment steps should be followed to develop the other codelet, "Generate Application ID." for the automating loan application process using codelet.

SERVICE

A web service is a software program accessible via the internet and uses a standardized XML messaging system. All communications to a web service are encoded in XML. A Yawl custom service is a web-based service that receives delegation notifications from the engine for enabled work items and notifies the engine that work item execution has commenced. A Yawl custom service executes the workitem's defined activities and notifies the Yawl engine that the work item execution is complete.

A Yawl custom service is in charge of ensuring that each work item allocated to it gets checked out of the engine and later returned to the engine, along with any updated data, when it is deemed complete.

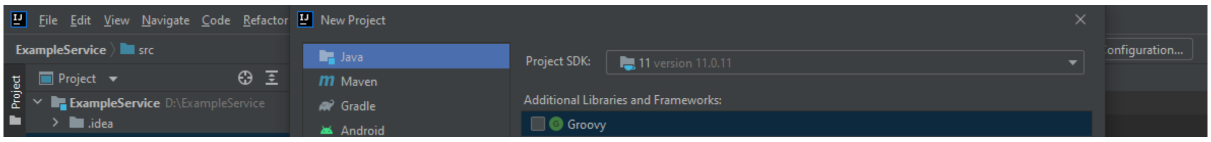
DEVELOPING A YAWL SERVICE

This section walks you through the steps necessary to create a Yawl custom service. As a running example, the Yawl Archiving service will be used. The entire service is made up of a single class written in roughly 150 lines of Java code. We will be building the Yawl codelet with IntelliJ Idea integrated development environment (IDE), Yawl Editor, and Java 11 environment.

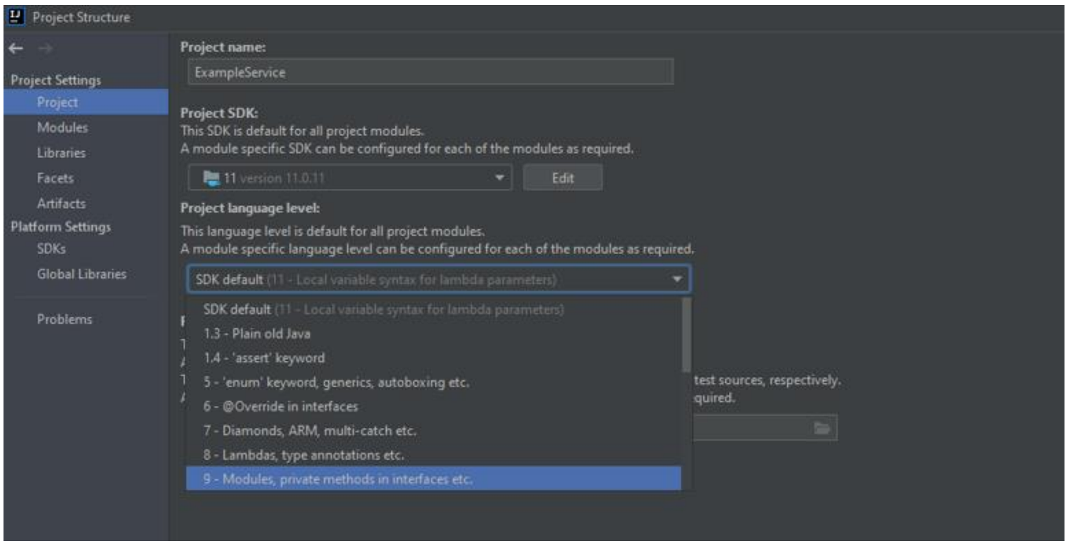
We will create a yawl service to automate the "Archive application" process in the model shown in Fig 1.1. We intend to archive each application after a final decision has been made before the process instance is completed by developing this archive service.

The steps are as follows:

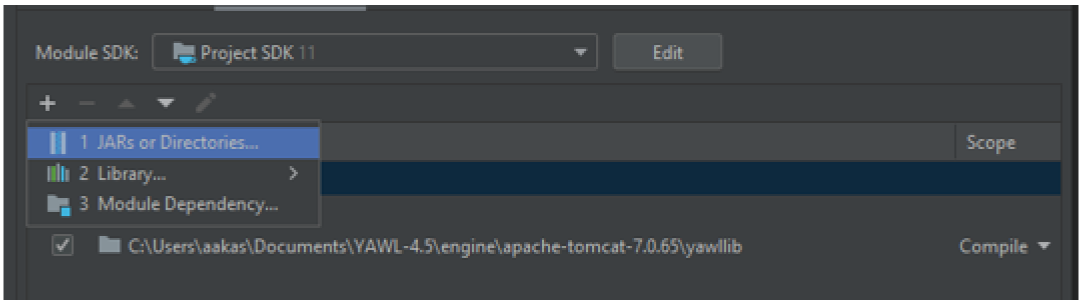
1. Go to file> New project and create a new project
   Here we are creating a new project with the name "Archiving service."



2. Now we need to set up our libraries; for doing this, we need to go to settings >Project structure> Project language level and select "9 Module's methods,private methods." This is the lang in which Yawl is compiled. This service should match the version in which Yawl is compiled.

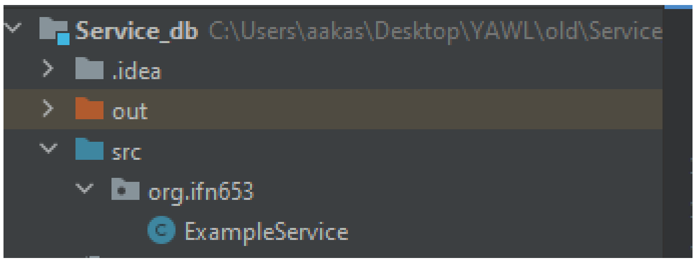

Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
Project name:
ExampleService
Project SDK:
This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.
11 version 11.0.11
Edit
Project language level:
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.
SDK default (11 - Local variable syntax for lambda parameters)
SDK default (11 - Local variable syntax for lambda parameters)
1.3 - Plain old Java
1.4 - 'assert' keyword
5 - 'enum' keyword, generics, autoboxing etc.
6 - @Override in interfaces
7 - Diamonds, ARM, multi-catch etc.
8 - Lambdas, type annotations etc.
9 - Modules, private methods in interfaces etc.
test sources, respectively.
quired.

3. For doing this, we need to go to Modules> Dependencies> Click on Add > JARS & dependencies and select the yawlib folder.



C:\Users\aakas\Documents\YAWL-4.5\engine\apache-tomcat-7.0.65\yawllib

4. yawlib can be found in [yawl installation]\engine\apache-tomcat-7.0.65\yawlib.
5. We have added all the required dependencies and are ready to create a java class in our package.
6. Similar to the codelet, we need to add a package structure for our service under the source folder. Add a package > Then create a Java class. Here for this example, we are creating a class "Example service."



7. After creating our Java class, we need to extend the class "InterfaceBWebsideController" for the custom service.

```
8. public class ArchivingService extends InterfaceBWebsideController {
```

9. Every custom service in Yawl extends from class "InterfaceBWebsideController" to the main class.

And the "InterfaceBWebsideController" must implement the two abstract methods as below for the custom services.

(a) public void handleEnabledWorkItemEvent(WorkItemRecord workItemRecord) :
    1. This method receives notification when a new work item is enabled.
    2. This method is the endpoint of a Yawl Engine-enabled work item event notification. This means the method is called when the engine creates a new work item and places it in the enabled state, and the Custom Service containing the method is the one associated at design time with the work item.

(b) public void handleCancelledWorkItemEvent (WorkItemRecord workItemRecord):

1. This method receives notification when a new work item is canceled.
2. This method is invoked when an enabled or executing work item is canceled by the engine. In principle, after this method has been invoked, a Custom Service should not

attempt to check-out or check-in the work item, and if the work item is in progress, its processing should be stopped.

Only the custom service specified for a work item receives these events

10. After implementing the two previous methods for our archive service, we must create the credentials used to register the service with the Yawl engine.

```
private final String USER = "akash";              //Service username
  private final String PASSWORD = "akash";          //Service password
```

Note: We will use the same password in the yawl admin screen to register this service.

11. We need to construct a session handle before we can do anything with the service. The following step is to generate a session handle with a string variable. Next, we need to use this session handle to authenticate our service with the engine.

```
   private String handle;
```

```
if (!connected()) handle = connect(USER, PASSWORD);
WorkItemRecord wir = checkOut(workItemRecord.getID(), handle);
String result = getInputs(wir);
```

12. We must specify the database URL where we intend to store our data and the credentials. We will need to store our data through this service. This is executed as follows in our class.

```
static final String USERDB = "";                //DB username
static final String PASS = "";                    //DB password
static final String JDBC_DRIVER = "org.h2.Driver";   //JDBC Driver
static final String DB_URL =
"jdbc:h2:\\\\C:\\Users\\aakas\\Documents\\YAWL-4.5\\engine\\apache-
tomcat-7.0.65\\da
```

13. The connected method will check if the connection is established with the engine by using the credentials provided and if it is valid.

```
static final String USERDB = "";                //DB username
static final String PASS = "";                    //DB password
static final String JDBC_DRIVER = "org.h2.Driver";   //JDBC Driver
static final String DB_URL =
"jdbc:h2:\\\\C:\\Users\\aakas\\Documents\\YAWL-4.5\\engine\\apache-
tomcat-7.0.65\\da
```

14. We need to create the methods to fetch the inputs to the service i.e., "String getStatus", "String getComments"," String getLoanID"," getSubmissionDate."

```
  private String getStatus(Element datalist,String Status){     //
status declaration
    if(datalist==null){
        return "error: no currency defined";
    }
    return datalist.getChildText(Status);

}
```

```
private String getComments(Element datalist,String Comments){
// comments declaration
    if(datalist==null){
        return "error: no currency defined";
    }
    return datalist.getChildText(Comments);

}
private String getLoanID(Element datalist,String LoanApplicationID){
// Loan id declaration
    if(datalist==null){
        return "error: no currency defined";
    }
    return datalist.getChildText(LoanApplicationID);

}
```

15. Next, we create a method to hold the output data

```
private Element getOutputData(String taskID, String result) {
    Element root = new Element(taskID);
    Element inner = new Element("result");
    inner.setText(String.valueOf(result));
    root.setContent(inner);
    return root;
}
```

16. Next, we need to create decompositions for the variables and set the type, i.e., input or output, as this service does not require any output to be sent to YAWL engine, we would only be using input variables and storing them into the database.

```
 public YParameter[] describeRequiredParams() {
    YParameter[] parameters = new YParameter[3];

    YParameter parameter = new YParameter(null,
YParameter._INPUT_PARAM_TYPE);
    parameter.setDataTypeAndName("string", "Status", XSD_NAMESPACE);
    parameters[1] = parameter;

    parameter = new YParameter(null, YParameter._INPUT_PARAM_TYPE);
    parameter.setDataTypeAndName("string", "Comments", XSD_NAMESPACE);
    parameters[2] = parameter;

    parameter = new YParameter(null, YParameter._INPUT_PARAM_TYPE);
    parameter.setDataTypeAndName("string", "LoanApplicationID",
XSD_NAMESPACE);
    parameters[3] = parameter;
```

We must use the service to insert the loan application into the database. This will be achieved by inputting the necessary inputs and storing them in the database by using a SQL query in our code.

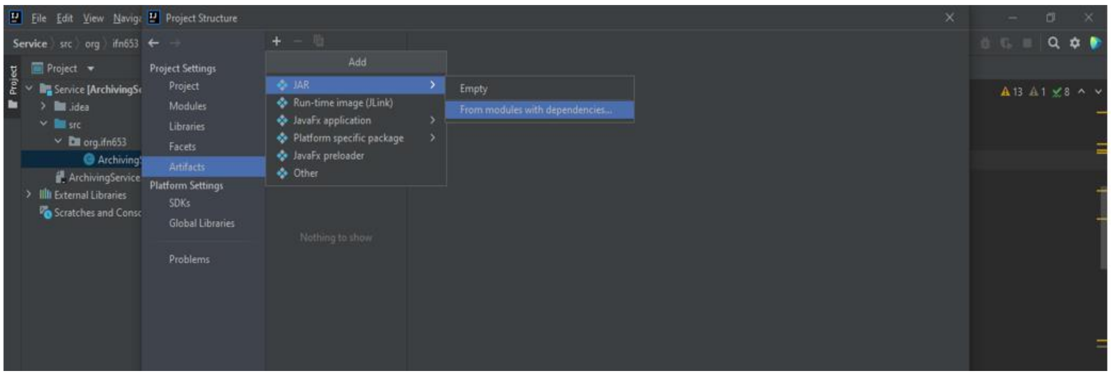
```
Connection conn = null;
Statement stmt;
Class.forName(JDBC_DRIVER);
conn = DriverManager.getConnection(DB_URL,USERDB,PASS);
stmt = conn.createStatement();
```

```
//String sql =  "INSERT INTO registration" + "VALUES
(id,first,last,age)";
String sql =  "INSERT INTO REGISTRATION " + "VALUES
('"+id+"','"+result+"','"+last+"','"+age+"')";
stmt.executeUpdate(sql);
System.out.println("Archived the application in the database...");
stmt.close();
conn.close();
```

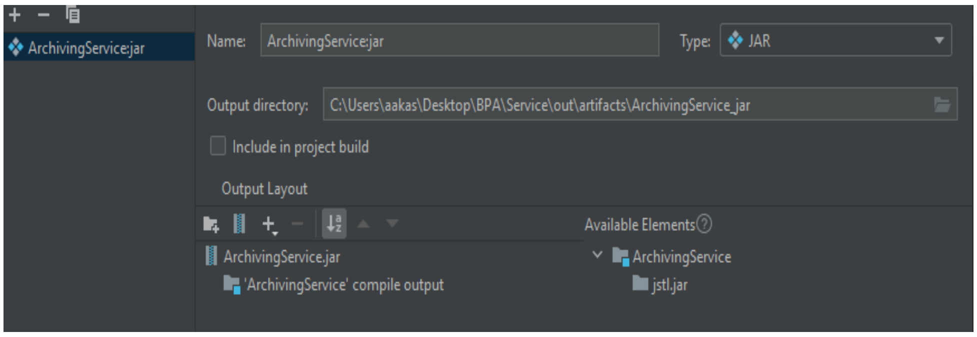
17. Before inserting the item, the user should ensure a table is created in the database to store the values. Failure to do so will result in an error.

DEPLOYING THE SERVICE

1. The next step is to compile the code to generate a class file. Click on Build > Compile.
2. After compiling the code, we need to build the artifact.
3. To build an artifact again to settings > Project structure >Artefacts
   Click on the add button and click on Add JARS

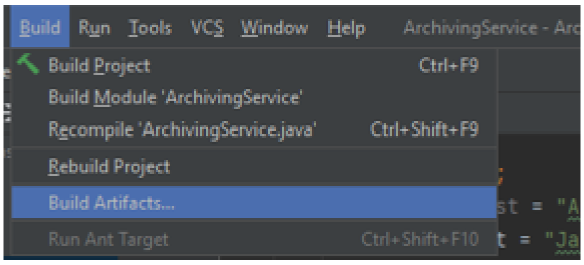


4. Click on modules and dependencies.
5. Remove all the other Jars, just keeping the "Archiving service compile output" file.

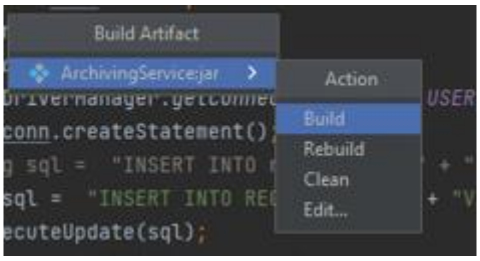


Name: ArchivingService:jar
Type: JAR
Output directory: C:\Users\aakas\Desktop\BPA\Service\out\artifacts\ArchivingService_jar
Include in project build
ArchivingService.jar
'ArchivingService' compile output

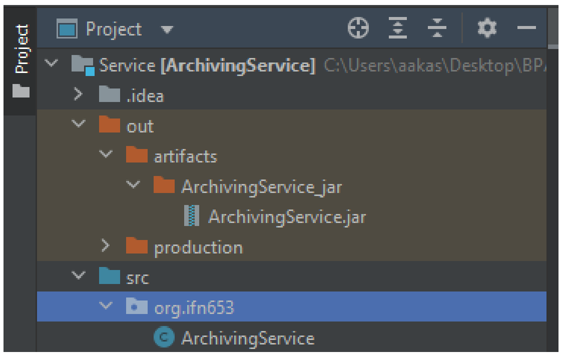
6. Once this step is complete, we need to go to build artifacts and click on build artifacts from JARS.


Build Run Tools VCS Window Help ArchivingService - Arc
Build Project Ctrl+F9
Build Module 'ArchivingService'
Recompile 'ArchivingService.java' Ctrl+Shift+F9
Rebuild Project
Build Artifacts...
Run Ant Target Ctrl+Shift+F10

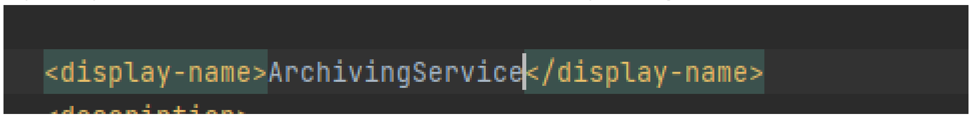
7. Then again, click on Build.



8. After building the artifact, a JAR file will be generated in the project structure



9. We need to copy this newly created artifact, i.e., our JAR, to the "yawlib" folder in the yawl installation directory.
10. The new JAR will be available in the project structure as follows:
    C:\Users\aakas\Desktop\BPA\Service\out\artifacts\ArchivingService_jar
    Copy the jar file and paste it into the yawlib folder in yawl installation.
11. The next step is to configure the configuration file. The easy step to do so is to go to the following path :
    C:\Users\aakas\Documents\YAWL-4.5\engine\apache-tomcat-7.0.65\webapps\resourceService
    Copy a duplicate folder of the "resourceservice" folder and rename it to our newly created service, i.e., "ArchivingService." This will help us to get a structure for our web.xml path and its required folder.
12. The next step is to make changes in the web.xml file as per the custom service we just created.
13. Open up the web.xml file and make the necessary changes.



```
<display-name>ArchivingService</display-name>
```

Change the service name along with the username and password we have created.

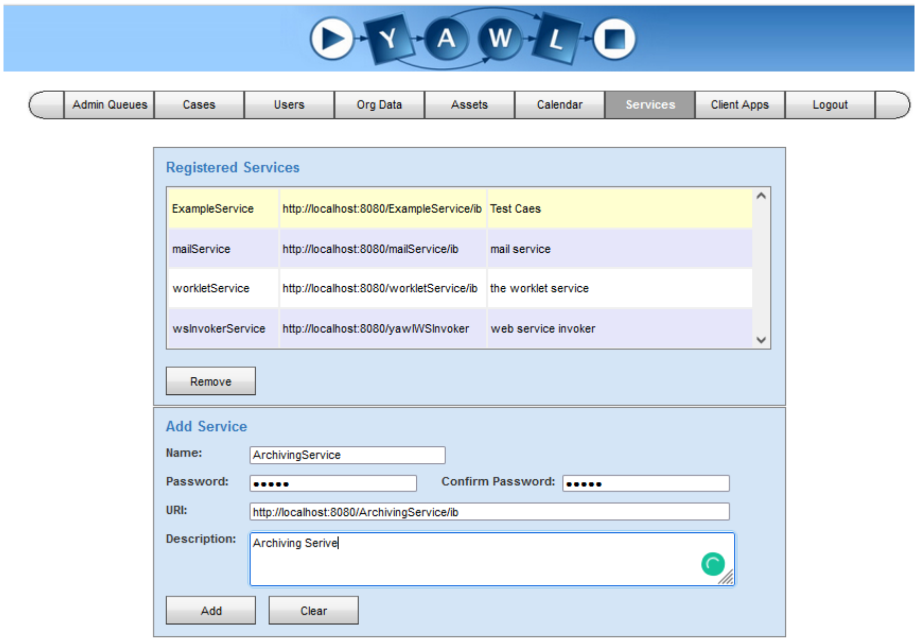
```
  <param-name>EngineLogonUserName</param-name>
<param-value>"akash"</param-value>
<description>
  <param-name>EngineLogonPassword</param-name>
<param-value>"akash"</param-value>
```

Note: Ensure the credentials are the same as our main class of our "Archiving service.java."

14. Next, we need to set up the mapping for the servlet.

```
<servlet-mapping>
    <servlet-name>ArchivingService</servlet-name>
    <url-pattern>/ib</url-pattern>
</servlet-mapping>
```

15. After configuring the web.xml file, we can start the yawl engine to run our newly created service.
16. Next, we need to go to the Yawl logon page and register our service



Registered Services

ExampleService	http://localhost:8080/ExampleService/ib	Test Caes
mailService	http://localhost:8080/mailService/ib	mail service
workletService	http://localhost:8080/workletService/ib	the worklet service
wsInvokerService	http://localhost:8080/yawlWSInvoker	web service invoker

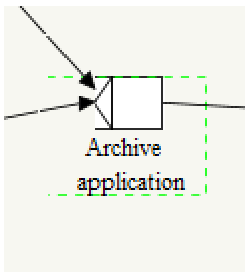
Remove

Add Service

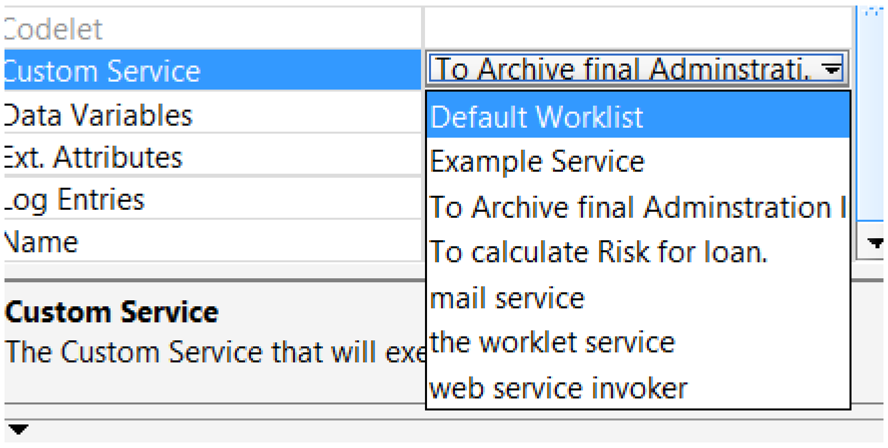
Name: ArchivingService
Password: ••••• Confirm Password: •••••
URI: http://localhost:8080/ArchivingService/ib
Description: Archiving Serive

Add Clear

17. Click on Add and register the service.
18. Once we have registered our service, we can go to our Yawl editor, select the" Archive application" task, and select our "ArchivingService," which we created.
19. Then if we go to the Yawl editor and select the task "Archiving task".



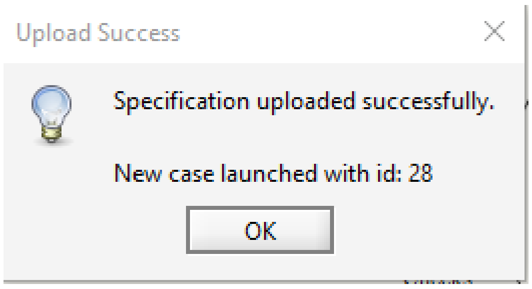
20. Then select the custom service "Archiving service".



Custom Service
The Custom Service that will exe

TESTING THE SERVICE

1) We will run the model while uploading the process to the Yawl engine.
We can see from the results that the values are saved in the database after the process is completed

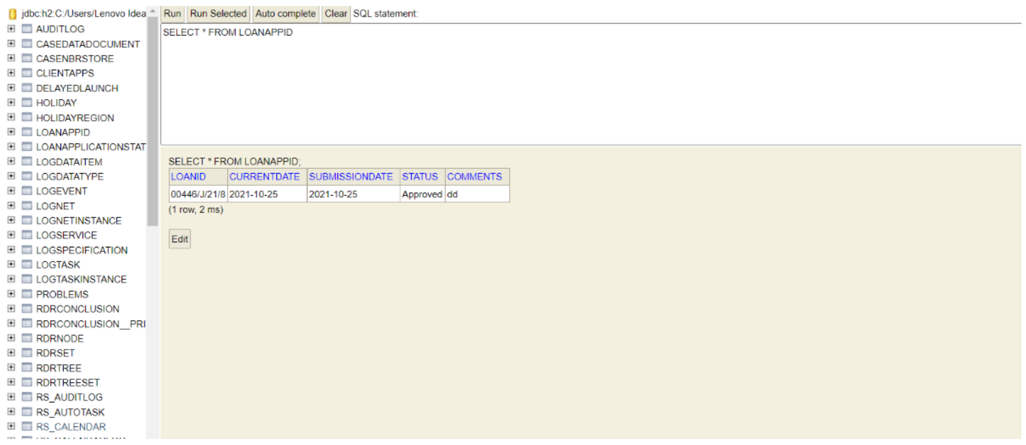


Upload Success

Specification uploaded successfully.

New case launched with id: 28

2) We can check that the service has been executed successfully and the values are stored in the database as required.



```
SELECT * FROM LOANAPPID
```

SELECT * FROM LOANAPPID;

LOANID	CURRENTDATE	SUBMISSIONDATE	STATUS	COMMENTS
00446/J/21/8	2021-10-25	2021-10-25	Approved	dd

(1 row, 2 ms)

The complete java code for this service will be included in appendix. The same development and deployment steps should be followed to develop the other service, "Risk Assessment Service" for the automating loan application process.

AN ORGANISATIONAL CHART

The organizational chart below represents the relational hierarchy of resources as they are organized in a bank.

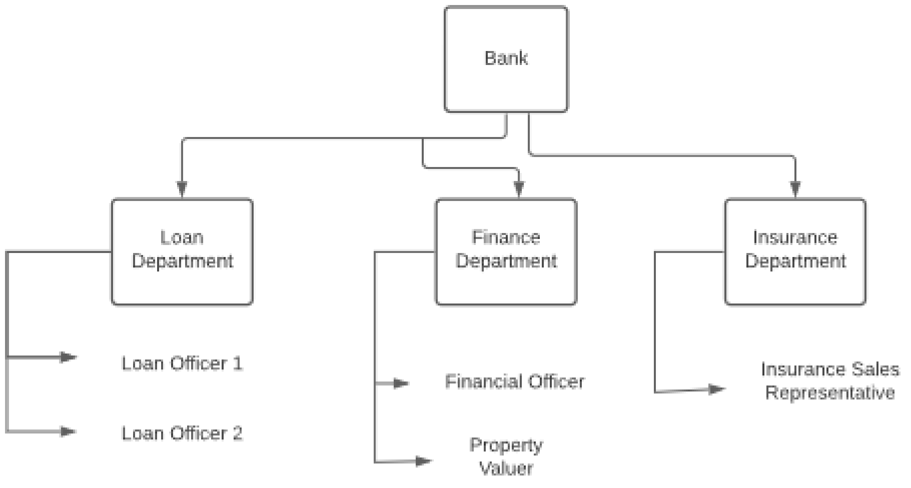


Figure 2: Organisation Model

Four roles are created in the organization hierarchy. The Loan Department consists of 2 loan Officers, of which loan officer 1 is responsible for the initial phase of the application process instances. Loan officer 2 is accountable for the verification of the loan application. The Loan officers can "Approve" and "Cancel" the application status depending on the application requirements.

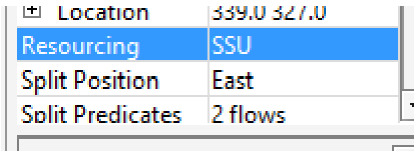
The Finance department performs check assessments on Property appraisal and credit history. The Financial officer checks the credit history of the applicant and passes the application for risk Assessment. The Property Valuer is responsible for the Property Appraisal check and passes the application for risk assessment.

The Insurance department offers the opportunity for the applicant to apply for home insurance, life insurance, credit card. If the applicants apply for any insurance, the insurance sales representative sends acceptance packages separately.

RESOURCES PERSPECTIVE

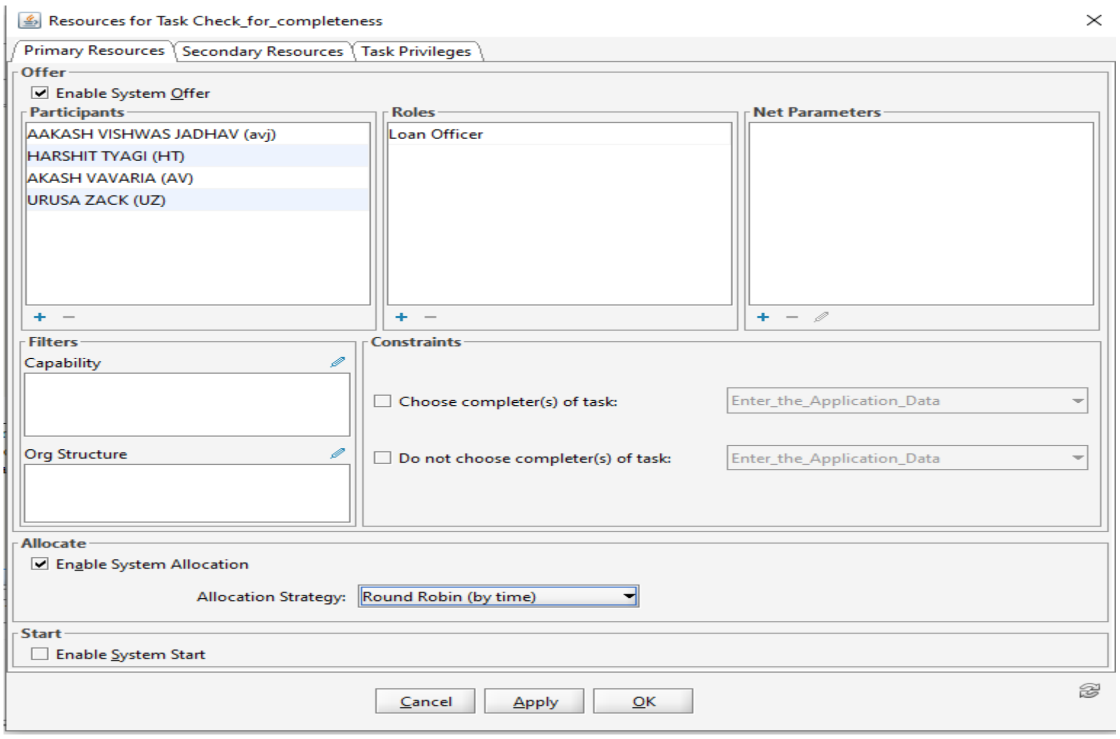
For the resource allocation

- Go to the yawl editor and open the loan application process model



Location	339.0 327.0
Resourcing	SSU
Split Position	East
Split Predicates	2 flows

- To allocate the resource, click on the respective task on the process model.
- Open- resources for Task Check complete and click + on Add roles
- After adding the roles, click on the Allocation strategy and select the strategy. In this case, for the loan officer, we will be selecting the "Round robin(by time)," as shown in the figure below.



Resources for Task Check_for_completeness

Primary Resources | Secondary Resources | Task Privileges

Offer

☑ Enable System Offer

Participants: AAKASH VISHWAS JADHAV (avj), HARSHIT TYAGI (HT), AKASH VAVARIA (AV), URUSA ZACK (UZ)

Roles: Loan Officer

Net Parameters

Filters: Capability, Org Structure

Constraints

☐ Choose completer(s) of task: Enter_the_Application_Data

☐ Do not choose completer(s) of task: Enter_the_Application_Data

Allocate

☑ Enable System Allocation

Allocation Strategy: Round Robin (by time)

Start

☐ Enable System Start

- Tick the enable system offer box in Primary resources., this will allow the roles to be assigned to the participant. For example, the "ROUND ROBIN (by time)" technique in "Enable System Allocation" will randomly assign the work to a participant. Depending on the requirements, many strategies can be implemented in the process.

REFERENCES

YAWL - User Manual Version 4.1. Yawlfoundation.org. (2021). Retrieved 21 October 2021, from http://www.yawlfoundation.org/manuals/YAWLUserManual4.1.pdf.

YAWL BPM. Yawlfoundation.github.io. (2021). Retrieved 18 October 2021, from https://yawlfoundation.github.io/.

APPENDIX

1.1 JAVA code for “Repayment amount Calculation”.

```
package org.ifn653.Unique_ID;

import java.util.ArrayList;
import java.util.List;
import org.jdom2.Element;
import org.yawlfoundation.yawl.elements.YDecomposition;
import org.yawlfoundation.yawl.elements.data.YParameter;
import org.yawlfoundation.yawl.resourcing.codelets.AbstractCodelet;
import
org.yawlfoundation.yawl.resourcing.codelets.CodeletExecutionException
;

public class RepaymentAmountCalculator extends AbstractCodelet {
    public RepaymentAmountCalculator() {
        this.setDescription("Calculate the repayment amount");
    }

    public Element execute(Element inData, List<YParameter> inParams,
List<YParameter> outParams) throws CodeletExecutionException {
        this.setInputs(inData, inParams, outParams);

        int NumberOfMonths;
        try {
            NumberOfMonths =
Integer.parseInt((String)this.getParameterValue("NumberOfMonths"));
        } catch (ClassCastException var9) {
            throw new CodeletExecutionException("Exception casting
input values to int types.");
        }

        Double LoanAmount =
(Double)this.getParameterValue("LoanAmount");
        this.setParameterValue("LoanAmount",
String.valueOf(LoanAmount));
        Double InterestRate =
(Double)this.getParameterValue("InterestRate");
        this.setParameterValue("InterestRate",
String.valueOf(InterestRate));
        double MonthlyRepaymentAmount = LoanAmount * InterestRate /
1200.0D * (Math.pow(1.0D + InterestRate / 1200.0D,
(double)NumberOfMonths) / (Math.pow(1.0D + InterestRate / 1200.0D,
(double)NumberOfMonths) - 1.0D));
        this.setParameterValue("MonthlyRepaymentAmount",
String.valueOf(MonthlyRepaymentAmount));
        return this.getOutputData();
    }

    public List<YParameter> getRequiredParams() {
        List<YParameter> parameters = new ArrayList();
        YParameter parameter = new YParameter((YDecomposition)null,
0);
        parameter.setDataTypeAndName("double", "LoanAmount",
"http://www.w3.org/2001/XMLSchema");
        parameter.setDocumentation("Enter the amount of loan
requested:");
        parameters.add(parameter);
        parameter = new YParameter((YDecomposition)null, 0);
        parameter.setDataTypeAndName("double", "InterestRate",
"http://www.w3.org/2001/XMLSchema");
        parameter.setDocumentation("Enter the monthly interest rate
in percentage:");
        parameters.add(parameter);
        parameter = new YParameter((YDecomposition)null, 0);
        parameter.setDataTypeAndName("int", "NumberOfMonths",
"http://www.w3.org/2001/XMLSchema");
        parameter.setDocumentation("Enter the total number of months
in the repayment period");
        parameters.add(parameter);
        parameter = new YParameter((YDecomposition)null, 1);
        parameter.setDataTypeAndName("double",
"MonthlyRepaymentAmount", "http://www.w3.org/2001/XMLSchema");
        parameter.setDocumentation("Repayment Amount");
        parameters.add(parameter);
        return parameters;
    }
}
```

1.2 JAVA code for “ArchivingService”

```
package org.Ifn653.ArchiveLoanApplication;
import java.sql.*;
import org.jdom2.Element;
import org.yawlfoundation.yawl.elements.data.YParameter;
import org.yawlfoundation.yawl.engine.interfce.WorkItemRecord;
import
org.yawlfoundation.yawl.engine.interfce.interfaceB.InterfaceBWebsideControl
ler;
import org.yawlfoundation.yawl.exceptions.YAWLException;
import java.io.IOException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;


public class ArchiveLoanApplication extends InterfaceBWebsideController {
    private final String USER = "akash";            //Service username
    private final String PASSWORD = "akash";        //Service password
    static final String USERDB = "";               //DB username
    static final String PASS = "";                  //DB password
    static final String JDBC_DRIVER = "org.h2.Driver";  //JDBC Driver
    static final String DB_URL = "jdbc:h2:~/test";
   // static final String DB_URL =
"jdbc:h2:C:C:\Users\aakas\Documents\YAWL-4.5\engine\apache-tomcat-
7.0.65\yawllib";       //DB URL
    private String handle;
    @Override
    public void handleEnabledWorkItemEvent(WorkItemRecord workItemRecord) {
        try {
            if (!connected()) handle = connect(USER, PASSWORD);
            WorkItemRecord wir = checkOut(workItemRecord.getID(), handle);
            String Status = getStatus(wir.getDataList(), "Status");
            String Comments = getComments(wir.getDataList(),"Comments");
            String LoanApplicationID = getLoanID(wir.getDataList(),
"LoanApplicationID");
            String SubmissionDate = getSubmissionDate(wir.getDataList(),
"SubmissionDate");
            DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE;
            String CurrentDate = formatter.format(LocalDate.now());
            Connection conn = null;
            Statement stmt;
            Class.forName(JDBC_DRIVER);
            conn = DriverManager.getConnection(DB_URL,USERDB,PASS);
            stmt = conn.createStatement();
            String sql =  "INSERT INTO Loanp " + "VALUES
('"+LoanApplicationID+"','"+CurrentDate+"','"+SubmissionDate+"','"+Status+"
','"+Comments+"')";
            stmt.executeUpdate(sql);
            stmt.close();
            conn.close();
        } catch (IOException | YAWLException /*| JDOMException*/ |
ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
    public void handleCancelledWorkItemEvent(WorkItemRecord wir) {
    }
    // these parameters are automatically inserted (in the Editor) into a
task
    // decomposition when this service is selected from the list
    public YParameter[] describeRequiredParams() {
        YParameter[] parameters = new YParameter[3];
        YParameter parameter = new YParameter(null,
YParameter._INPUT_PARAM_TYPE);
        parameter.setDataTypeAndName("string", "Status", XSD_NAMESPACE);
        parameters[1] = parameter;
        parameter = new YParameter(null, YParameter._INPUT_PARAM_TYPE);
        parameter.setDataTypeAndName("string", "Comments", XSD_NAMESPACE);
        parameters[2] = parameter;
        parameter = new YParameter(null, YParameter._INPUT_PARAM_TYPE);
        parameter.setDataTypeAndName("string", "LoanApplicationID",
XSD_NAMESPACE);
        parameters[3] = parameter;
        parameter = new YParameter(null, YParameter._INPUT_PARAM_TYPE);
        parameter.setDataTypeAndName("string", "SubmissionDate",
XSD_NAMESPACE);
        parameters[4] = parameter;
        return parameters;
    }

    private String getStatus(Element datalist,String Status){     // status
declaration
        if(datalist==null){
            return "error: no currency defined";
        }
        return datalist.getChildText(Status);
    }
    private String getComments(Element datalist,String Comments){        //
comments declaration
        if(datalist==null){
            return "error: no currency defined";
        }
        return datalist.getChildText(Comments);
    }
    private String getLoanID(Element datalist,String LoanApplicationID){
// Loan id declaration
        if(datalist==null){
            return "error: no currency defined";
        }
        return datalist.getChildText(LoanApplicationID);
    }
    private String getSubmissionDate(Element datalist, String
SubmissionDate){     // Submission date declaration
        if(datalist==null){
            return "error: no currency defined";
        }
        return datalist.getChildText(SubmissionDate);
    }
    private boolean connected() throws IOException{
        return handle != null && checkConnection(handle);
    }
}
```

1.3 Data Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="LoanApplication">
    <xs:sequence>
      <xs:element name="Title" type="xs:string" minOccurs="0" />
      <xs:element name="FirstName" type="xs:string" minOccurs="0" />
      <xs:element name="Lastname" type="xs:string" minOccurs="0" />
      <xs:element name="DOB" type="xs:date" minOccurs="0" />
      <xs:element name="Email" type="xs:string" minOccurs="0" />
      <xs:element name="Phone" type="xs:long" minOccurs="0" />
      <xs:element name="StreetName" type="xs:string" minOccurs="0" />
      <xs:element name="StreetNumber" type="xs:int" minOccurs="0" />
      <xs:element name="City" type="xs:string" minOccurs="0" />
      <xs:element name="PinCode" type="xs:int" minOccurs="0" />
      <xs:element maxOccurs="unbounded" minOccurs="0" name="PreviousAddress">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="StreetName" type="xs:string" />
            <xs:element name="StreetNumber" type="xs:int" />
            <xs:element name="City" type="xs:string" />
            <xs:element name="PinCode" type="xs:int" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="CurrentEmployer" type="xs:string" minOccurs="0" />
      <xs:element name="MonthlyNetSalary" type="xs:double" minOccurs="0" />
      <xs:element maxOccurs="unbounded" minOccurs="0" name="OtherIncome">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Type" type="xs:string" />
            <xs:element name="MonthlyNetIncome" type="xs:double" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="TotalMonthlyOutgoings" type="xs:double" minOccurs="0" />
      <xs:element name="ExcludingExistingLoans" type="xs:double" minOccurs="0" />
      <xs:element name="CreditCardRepayments" type="xs:string" minOccurs="0" />
      <xs:element maxOccurs="unbounded" name="BankAccounts">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="BankName" type="xs:string" minOccurs="0" />
            <xs:element name="TypeOfAccount" type="xs:string" minOccurs="0" />
            <xs:element name="AccountNumber" type="xs:string" minOccurs="0" />
            <xs:element name="CurrentCustomer" type="xs:boolean" minOccurs="0" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="PropertyInformation">
    <xs:sequence>
      <xs:element name="TypeOfProperty" type="xs:string" minOccurs="0" />
      <xs:element name="PropertyAddress" type="xs:string" minOccurs="0" />
      <xs:element name="StreetName" type="xs:string" minOccurs="0" />
      <xs:element name="StreetNumber" type="xs:int" minOccurs="0" />
      <xs:element name="City" type="xs:string" minOccurs="0" />
      <xs:element name="PostCode" type="xs:int" minOccurs="0" />
      <xs:element name="PurchasePrice" type="xs:double" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LoanInformation">
    <xs:sequence>
      <xs:element name="LoanAmount" type="xs:double" minOccurs="0" />
      <xs:element name="NumberOfMonths" type="xs:int" minOccurs="0" />
      <xs:element name="StartDate" type="xs:date" minOccurs="0" />
```

```
      <xs:element name="InterestRate" type="xs:double" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LifeInsuranceQuoteRequired">
    <xs:sequence>
      <xs:element name="LifeInsuranceRequired" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="CreditCardQuoteRequired">
    <xs:sequence>
      <xs:element name="CreditRequired" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="HomeInsuranceQuoteRequired">
    <xs:sequence>
      <xs:element name="HomeInsuranceRequired" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AdministrationInformation">
    <xs:sequence>
      <xs:element name="LoanApplicationID" type="xs:string" minOccurs="0" />
      <xs:element name="SubmissionDate" type="xs:date" minOccurs="0" />
      <xs:element name="RevisionDate" type="xs:date" minOccurs="0" />
      <xs:element name="Status" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Approved" />
            <xs:enumeration value="Complete" />
            <xs:enumeration value="Incomplete" />
            <xs:enumeration value="Assessed" />
            <xs:enumeration value="Rejected" />
            <xs:enumeration value="Canceled" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Comments" type="xs:string" minOccurs="0" />
      <xs:element name="Eligibility" type="xs:boolean" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="CurrentCreditCardInformation">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="LoanInFiveYears">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="LoanType" type="xs:string" />
            <xs:element name="Amount" type="xs:double" />
            <xs:element name="DurationInYears" type="xs:int" />
            <xs:element name="OutstandingAmount" type="xs:double" />
            <xs:element name="MonthlyRepayments" type="xs:double" />
            <xs:element name="InterestRate" type="xs:double" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="CreditCardInfo">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Provider" type="xs:string" />
            <xs:element name="CreditLimit" type="xs:double" />
            <xs:element name="OutstandingAmount" type="xs:double" />
            <xs:element name="MonthlyRepayments" type="xs:double" />
            <xs:element name="InterestRate" type="xs:double" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="PublicRecordInformation">
    <xs:sequence>
      <xs:element name="CourtJudgementInformation" type="xs:int" minOccurs="0" />
      <xs:element name="BankruptcyInformation" minOccurs="0"
  maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="DateRaised" type="xs:date" />
            <xs:element name="Current" type="xs:boolean" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="CreditAssessment">
    <xs:sequence>
      <xs:element name="CreditAssessmentResult">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="AAA" />
            <xs:enumeration value="AA" />
            <xs:enumeration value="A" />
            <xs:enumeration value="BBB" />
            <xs:enumeration value="BB" />
            <xs:enumeration value="B" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

```
 <xs:complexType name="RiskAssesment">
  <xs:sequence>
   <xs:element name="RiskWeight" type="xs:int" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="PropertyAppraisal">
  <xs:sequence>
   <xs:element name="PropertyAppraiserIdentifier" type="xs:string" />
   <xs:element name="SurroundingPropertiesValue1" type="xs:double" />
   <xs:element name="SurroundingPropertiesValue2" type="xs:double" />
   <xs:element name="SurroundingPropertiesValue3" type="xs:double" />
   <xs:element name="EstimatedPropertyMarketValue" type="xs:double" />
   <xs:element name="Comments" type="xs:string" minOccurs="0" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="RepaymentSchedule">
  <xs:sequence>
   <xs:element name="MonthlyRepaymentAmount" type="xs:double" />
   <xs:element name="NumberOfRepayment" type="xs:int" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="LifeInsuranceQuote">
  <xs:sequence>
   <xs:element name="LifeInsuranceTotalCost" type="xs:double" />
   <xs:element name="InsuranceSalesRepresentativeName" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="HomeInsuranceQuote">
  <xs:sequence>
   <xs:element name="HomeInsuranceTotalCost" type="xs:double" />
   <xs:element name="InsuranceSalesRepresentativeName" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="CreditCardQuote">
  <xs:sequence>
   <xs:element name="CardLimit" type="xs:double" />
   <xs:element name="FinanceOfficerName" type="xs:string" />
   <xs:element name="NormalInterestRate" type="xs:double" />
   <xs:element name="DiscountedInterestRate" type="xs:double" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="AgreementSummary">
  <xs:sequence>
   <xs:element name="ConditionsAgreed" type="xs:boolean" minOccurs="0" />
   <xs:element name="RepaymentAgreed" type="xs:boolean" minOccurs="0" />
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```