Loan assessment process lab

YAWL

Eric Kwok 02741172

Table of Contents

1.0. Introduction

YAWL (Yet Another Workflow Language) was developed by Wil van der Aalst (Eindhoven University of Technology, the Netherlands) and Arthur ter Hofstede (Queensland University of Technology, Australia) in 2002. The language was based on the one hand on Petri nets, a well-established concurrency theory with a graphical representation, and on the other hand on the well-known workflow patterns, which is a generally accepted benchmark for the suitability of a process specification language (see section 5.0. Further readings for references to learn more about Petri nets and workflow patterns). YAWL was developed in the effort to address some of the shortcomings of existing workflow management systems and workflow languages; it extends Petri nets with dedicated constructs to support multiple instance, cancellation and inclusive OR-join (The YAWL foundation, 2014, p. 9).

1.1. Features

One of the distinguishing features of YAWL is that it has a formal foundation, that is, both a precisely defined syntax and a precisely defined semantics. It removes any ambiguity associated with the interpretation of complex constructs and their interplay, and also allows for the development of sophisticated verification techniques that allow the detection of inherent flaws in an executable process model before it deployed (Hofstede, Aalst, Adams, & Russell, 2010, p. 15). According to the YAWL foundation (2014, p. 9), other distinctive features of YAWL include:

- YAWL offers comprehensive support for the control-flow patterns. It is the most powerful process specification language for capturing control-flow dependencies.
- The data perspective in YAWL is captured through the use of XML Schema, XPath and XQuery.
- YAWL offers comprehensive support for the resource patterns. It is the most powerful process specification language for capturing resourcing requirements.
- YAWL has been developed independent from any commercial interests. It simply aims to be the most powerful language for process specification.
- For its expressiveness, YAWL offers relatively few constructs (compare this e.g. to BPMN).
- YAWL offers unique support for exceptional handling, both those that were and those that were not anticipated at design time.
- YAWL offers unique support for dynamic workflow through the Worklets approach. Workflows can thus evolve over time to meet new and changing requirements.
- YAWL aims to be straightforward to deploy. It offers a number of automatic installers and an intuitive graphical design environment.
- The Declare component (released through declare.sf.net) provides unique support for specifying workflows in terms of constraints. This approach can be combined with the Worklet approach thus providing very powerful flexibility support.
- YAWL's architecture is Service-oriented and hence one can replace existing components with one's own or extend the environment with newly developed components.
- The YAWL environments supports the automated generation of forms. This is particularly useful for rapid prototyping purposes.
- Tasks in YAWL can be mapped to human participants, web services, external applications or to Java classes.

- Through the C-YAWL approach a theory has been developed for the configuration of YAWL models. For more information on process configuration visit www.processconfiguration.com.
- Simulation support is offered through a link with the ProM (www.processmining.org) environment. Through this environment it is also possible to conduct post-execution analysis of YAWL processes (e.g. in order to identify bottlenecks).

1.2. Architecture and components

YAWL system is structured as a RESTful service-oriented architecture. The characteristics of such architecture contribute to keeping the YAWL system lightweight and platform-neutral, which have been two of the key underlying design principles. While other architectures enabling additional infrastructure-level functionality to be exploited, it was not found to be strictly necessary in the context of the YAWL system. The engine is completely agnostic with regards to the services interacting with it; that is, totally unaware of the operations of external services, so that each could be served in a generic way, it provides a flexible and extensible framework for plugging in additional (custom) services into the YAWL system (Hofstede, Aalst, Adams, & Russell, 2010).

The YAWL system provides a specific embodiment of the Workflow Reference Model (WRM) interfaces of the Workflow Management Coalition in the setting of a service-oriented architecture. Also, in many respects, the YAWL interface extends and deviates from the WRM interfaces. Thus, it cannot be said that the YAWL system complies with the WRM. It is merely inspired by the WRM, but significantly differs from it in many respects (Hofstede, Aalst, Adams, & Russell, 2010).

The YAWL system can be presented in a three tier view as shown in figure 1.0 (Hofstede, Aalst, Adams, & Russell, 2010), which consists of YAWL services at the business logic layer encapsulate resources in the data layer, and provide functionality to user-facing services (or applications) at the presentation layers.

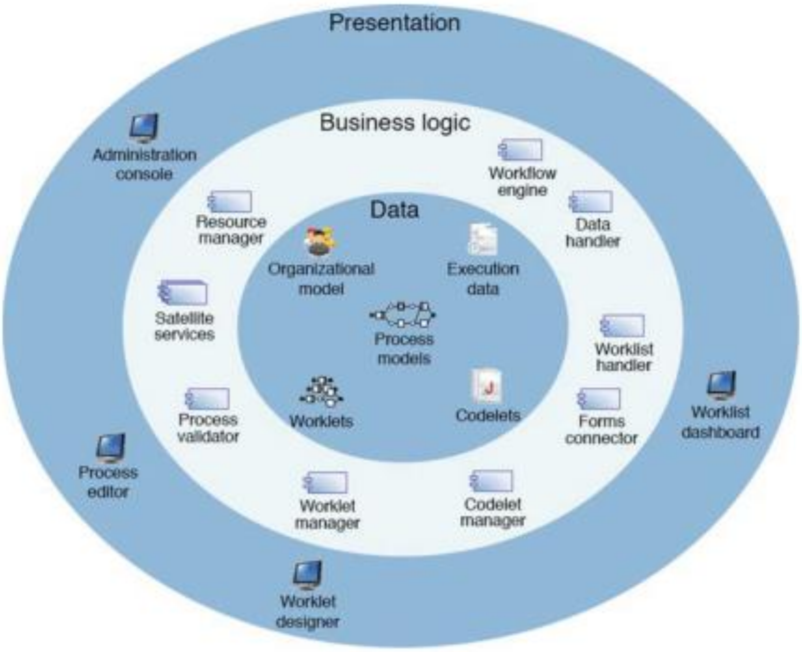


Figure 1.0. The YAWL system architecture

The core service of the YAWL system is the workflow engine, it interacts with other services in the YAWL system through four interfaces as shown in figure 1.1 (Hofstede, Aalst, Adams, & Russell, 2010), three of which correspond to the WRM. The engine interfaces are the following:

- Interface A, which corresponds to interface 1 (and partially to interface 5) of the WRM, and provides endpoints for uploading and unloading process specifications, registering or removing references to external services and basic user connections and disconnections.
- Interface B, which corresponds to interfaces 2, 3, and 4 of the WRM, and provides endpoints for services to establish a session with the engine, launch process instances, check work items in and out of the engine, and retrieve process data and state information.
- Interface E, which (partially) corresponds to interface 5, and provides endpoints for the retrieval and analysis of process logs.

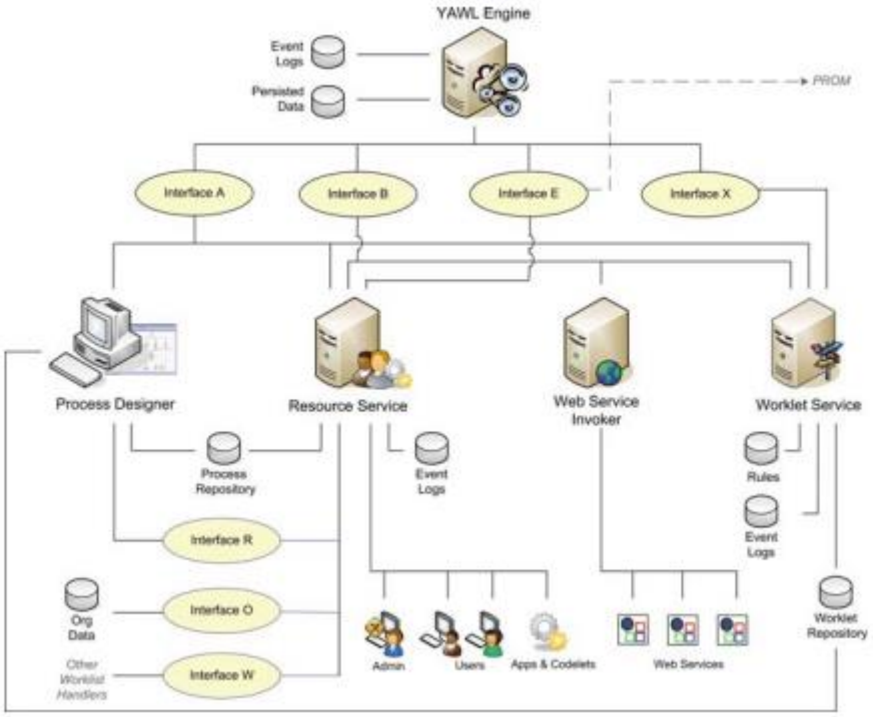


Figure 1.1. YAWL system's core services and their interfaces

1.3. BPMN to YAWL

YAWL is an executable language to enable process automation; it is a detailed specification that consists of data types, data extraction and conversion steps, application bindings, resource allocation, and distribution policies, among others. In comparison to BPMN, BPMN is a language that has a complete different abstraction level, which used for the purpose of business analysis and improvement. Due to BPMN and YAWL share many common concepts (see figure 1.2) (Hofstede, Aalst, Adams, & Russell, 2010), BPMN models obtained from the business analysis phase can be easily mapped into the equivalent YAWL specifications.

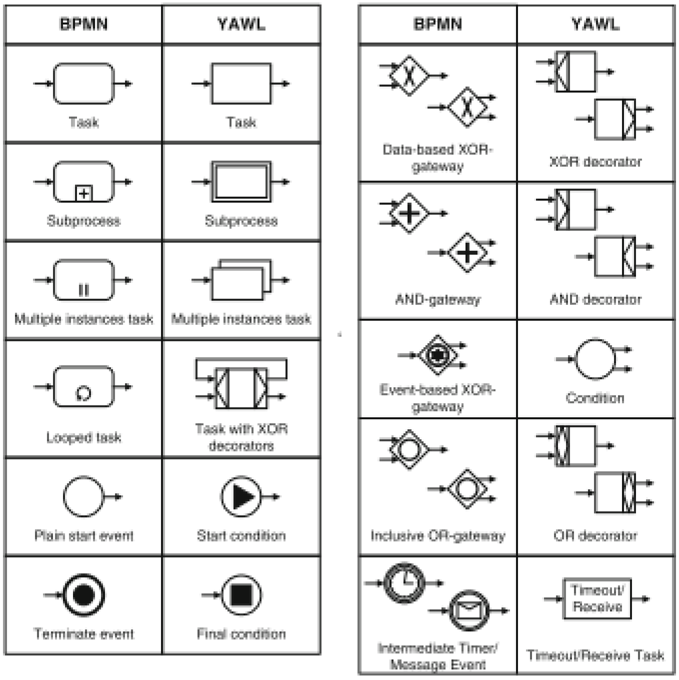


Figure 1.2. Mapping of tasks, events and gateways

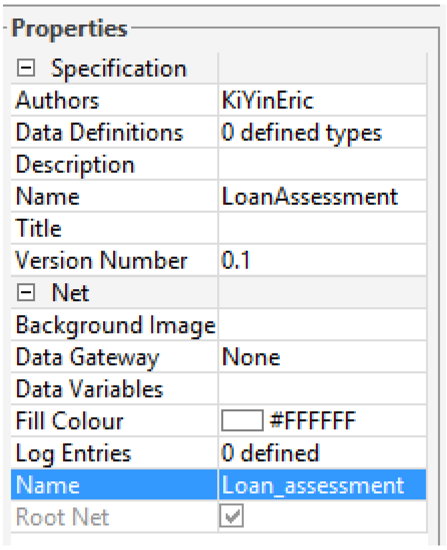
2.0. Implement the process

The following sections will be demonstrating how the loan assessment process can be implemented in YAWL. Please note that due to YAWL is a very flexible workflow management system, there could be multiple (or perhaps better) ways of achieving the same end result that what have been shown in the following lab exercise. However, the selected way of implementation is to keep the loan assessment process in YAWL semantically as similar as possible to the given BPMN. For example, instead of having an extra task to check application form completeness and then return the incomplete application to the applicant for resubmission, YAWL is capable of carrying out the check through its dynamic form and preventing an incomplete form being submitted in the first place.

2.1. Control flow perspective

The editor in YAWL is the tool that enables workflow designers to graphically define complex process models, and to analyse and export these models to the engine. One way to start YAWL editor, click on the YAWL – Editor icon located at start/all programs/YAWL folder. The editor will be opened with a blank canvas. Click on the create a new specification button, at the top left of the menu toolbar, or click on file in the menu and choose new. A blank net called 'Net' by default will be shown, this is where the loan assessment workflow will be created, configured and validated.

The properties pane at the left of the canvas is used to view and modify all of the properties of the selected specification, net, decomposition and net element (task, condition or flow). To give the specification and net a more descriptive name, update the specification and net name properties as shown in figure 2.0.



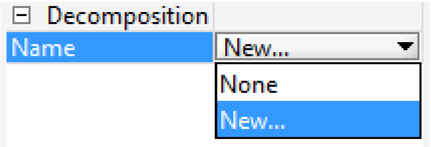
Properties

⊟ Specification	
Authors	KiYinEric
Data Definitions	0 defined types
Description	
Name	LoanAssessment
Title	
Version Number	0.1
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	
Fill Colour	□ #FFFFFF
Log Entries	0 defined
Name	Loan_assessment
Root Net	☑

Figure 2.0. Properties pane

In theory, the input condition represent the process begins when loan applications being received after the applicants made a submission through the loan provider's website. However, for the purpose of simplicity and being the main focus of the lab exercise is to teach how business processes can be automated with YAWL. Instead of building a website that interacts with the process by external service, the applicant has been defined as a role within YAWL and additional tasks will be added to simulate the online loan application submission, i.e. submit and complete application task.

Click on the atomic task button □ in the elements palette, or right click in an empty area of the canvas, and choose atomic task. Left mouse click on the right of the input condition to place an atomic task. Set the decomposition of the task by select New… in the drop down box of the decomposition name at the properties pane (see figure 2.1). Type 'Submit application' in the new decomposition dialog and click the OK button.



⊟ Decomposition	
Name	New…

Figure 2.1. Decomposition properties

Repeat the process to create all the loan assessment tasks as shown in table 2.0 and the loan assessment net should end up appearing as figure 2.2.

Check application form completeness	CompleteApplication	Check credit history	Assess loan risk
Appraise property	Assess eligibility	Reject application	Prepare and send acceptance pack
Prepare and send home insurance quote	VerifyRepaymentAgreement	Take final decision	Update status

Table 2.0. Loan assessment tasks

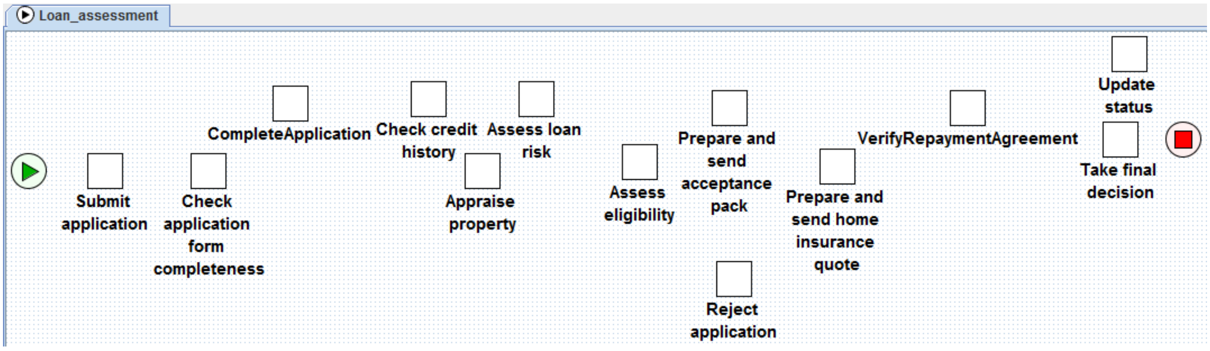


Figure 2.2. Loan assessment tasks

To establish the flow of execution, connect the input condition to submit application task by finding the flow connectors that appear as a small cross as you hover your mouse over the sides of the submit application task. Hold the left mouse button down over a flow connector and draw a line by dragging the mouse from the flow connector on the input condition to the one on the submit application task, figure 2.2 shows an established flow relation.

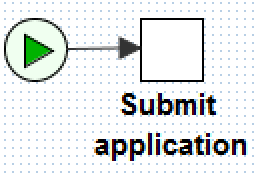


Figure 2.2. Flow relation established

Repeat the process to create flow relations from submit application task to check application form completeness task. Next, a XOR split decorator needs to be added to check application form completeness task to specify the possible branches that will lead to the succeeding task depending on the check outcome. Select check application form completeness task and select XOR split from the join type drop down box of the task section at the properties pane, this will enable multiple flow relations from the task. Create one flow relations from check application form completeness task to complete application task. The branch will be executed if an application is found incomplete, such application is returned to the applicant for them to complete and resubmit. In order to resubmit, a XOR split and XOR join decorator is added to the complete application task and check application form completeness task respectively, an outgoing flow relation from the complete application task to check application form completeness task is also created. Another flow relation is needed from complete application task to the update status task, and finally to the end condition, this branch will be executed if the application is not complete and resubmitted within five days and it will update the application status to cancel and then the process will come to its end. If the application is completed, an AND split is needed to enable a credit check and property appraisal to execute simultaneously. Add an additional task with a AND split decorator to the net, the task is used as an empty/routing task that does not require decomposition, it is simply for routing purposes, i.e. to enable tasks that follows to execute simultaneously. Join the check application form completeness task to the routing task, then join the routing task to the appraise property task and check credit history, and continue on with assess loan risk task (see figure 2.3).

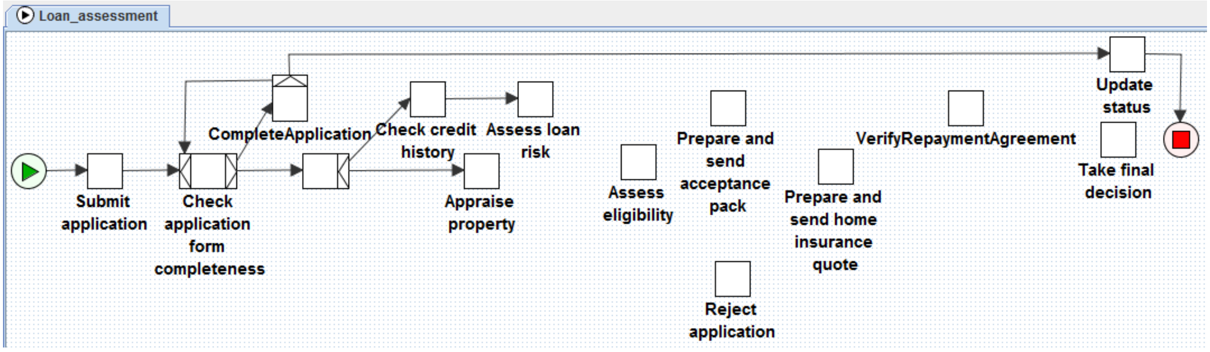


Figure 2.3. Flow relations of check application form completeness task

Next, add an AND join and XOR split decorator to assess eligibility task to enable synchronisation of previous tasks, then establish flows from both assess loan risk and appraise property task to assess eligibility. For the outgoing flows, connect assess eligibility task to reject application and prepare and send acceptance pack task. Add XOR split decorator to prepare and send acceptance pack task, create one flow to prepare and send home insurance quote task and another to verify repayment agreement task. Add a XOR join decorator to verify repayment agreement task to allow an outgoing flow from prepare and send insurance quote task to connect to it. For the reject application task, create flow to the end condition. Add another XOR split decorator to verify repayment agreement task and a XOR join decorator to update status task, establish one flow to take final decision task and one to update status task. Lastly, connect both take final decision task and update status task to the end condition (see figure 2.4).

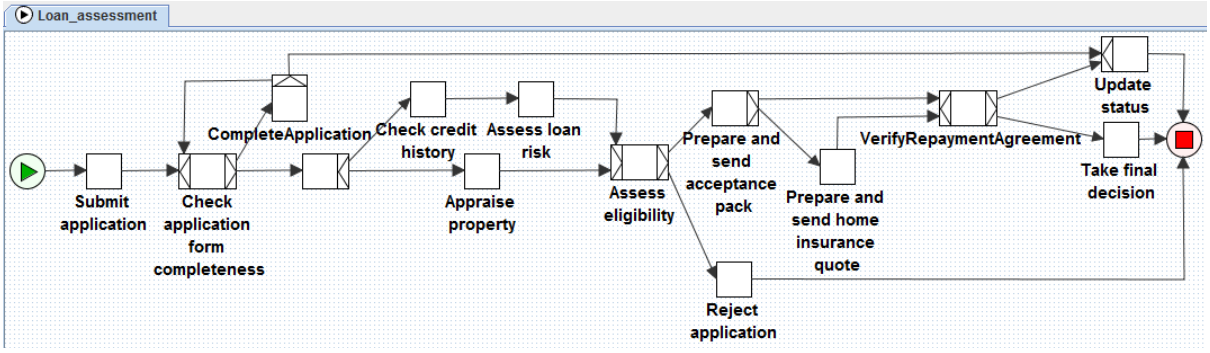
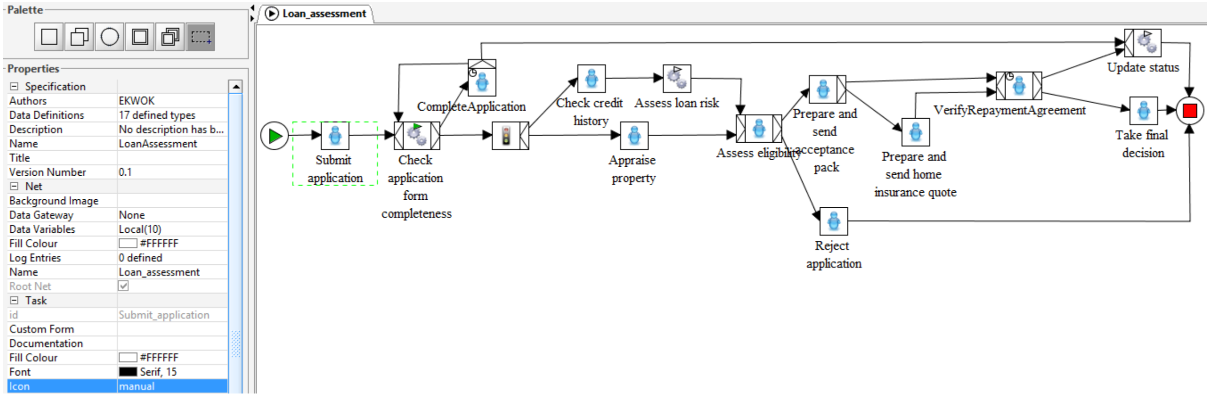


Figure 2.4. Completed control flow perspective

To make the YAWL specification more easily understood, select each task and add task icon from the task icon properties as shown in figure 2.5.



Palette

Properties

⊟ Specification	
Authors	EKWOK
Data Definitions	17 defined types
Description	No description has b...
Name	LoanAssessment
Title	
Version Number	0.1
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	Local(10)
Fill Colour	□ #FFFFFF
Log Entries	0 defined
Name	Loan_assessment
Root Net	☑
⊟ Task	
id	Submit_application
Custom Form	
Documentation	
Fill Colour	□ #FFFFFF
Font	■ Serif, 15
Icon	manual

Figure 2.5. Loan assessment with task icons

To save the specification, click on the save button 💾, and the save dialog appears, choose the location to be saved at and click save.

2.2. Data perspective

After the control flow of the process have been established, the next thing to consider is the data requirements. Some of the decisions that need to be made when modelling data includes:

- Data that participants need in each work item.
- Data that must return to the engine once the work item is complete.
- Way of data moves between tasks and net in a running workflow.
- Use of data to choose between flows.

YAWL uses XML schema to define data documents that are passed from net to task and back during the life of a workflow instance. User-defined data types are also supported, by allowing for the definition of XML Schema complex types, which are added to a specification and then may be used to define variables based on those types. The remainder of the section is to step through how to implement the data perspective to the loan assessment process in YAWL. The following complex elements shown in table 2.1 needs to be created in order to capture the required data. The complex elements with indented complex elements indicate that such complex element is based on another complex element, for example, the loan application complex type is based on ApplicantInformation complex type, PropertyInformation complex type and so on.

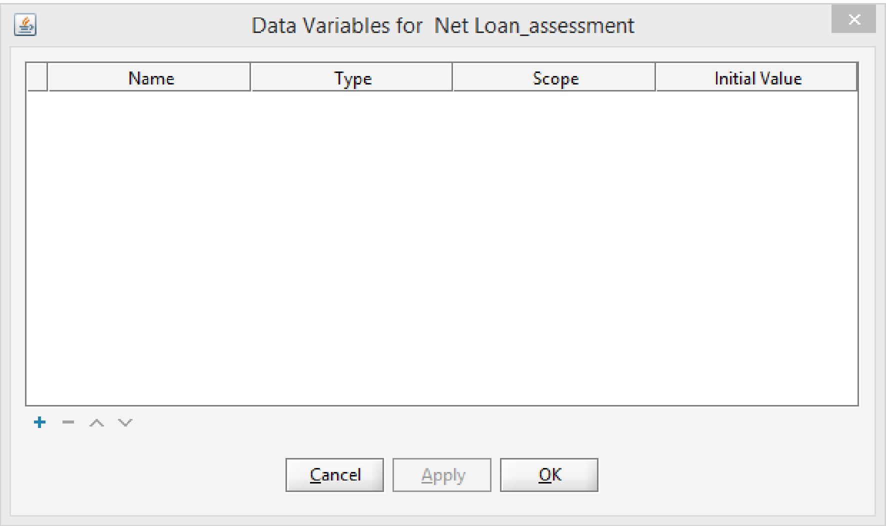
Complex element	Sub-component (simple element)	Type
Loan application		
ApplicantInformation	Name	String
	Surname	String
	Email	String
	HomePhone	Int
	CellPhone	Int
	CurrentAddress	String
	PreviousAddress	String
	CurrentEmployer	String
	MonthlyNetRevenue	Double
	BankName	String
	TypeOfAccount	String

	AccountNumber	String
PropertyInformation	TypeOfProperty	String
	PropertyAddress	String
	PurchasingPrice	Double
LoanInformation	LoanAmount	Double
	NumberOfYears	Double
	StartDate	Date
	InterestRate	Double
	InterestType	String
InsuranceQuoteRequired	Required	Boolean
AdministrationInformation	ApplicationIdentifier	String
	SubmissionDateAndTime	DateTime
	RevisionDateAndTime	DateTime
	Status	String
	Comments	String
	Eligibility	Boolean
	LoanOfficerIdentifier	String
CreditHistoryReport		
FinancialOfficerIdentifier	ID	String
ApplicantCreditInformation	LoanType	String
	Amount	Double
	Duration	Double
	InterestRate	Double
OverdueCreditAccounts	CreditType	String
	DefaultAmount	Double
	Duration	Double
	InterestRate	Double
CurrentCreditCardInformation	Provider	String
	StartDate	Date
	EndDate	Date
	InterestRate	Double
PublicRecordInformation	CourtJudgementInformation	String
	BankruptcyInformation	String
CreditAssessment	Result	String
PropertyAppraisal	PropertyAppraiserIdentifier	String
	SurroundingPropertiesValue1	Double
	SurroundingPropertiesValue2	Double
	SurroundingPropertiesValue3	Double
	EstimatedPropertyMarketValue	Double
	Comments	String
RepaymentSchedule	MonthlyRepaymentAmount	Double
	NumberOfRepayment	Int
HomeInsuranceQuote	HomeInsuranceTotalCost	Double
	AdditionalCostOnMonthlyLoanRepayment	Double
	InsuranceSalesRepresentativeIdentifier	String

Table 2.1. Loan application data requirements

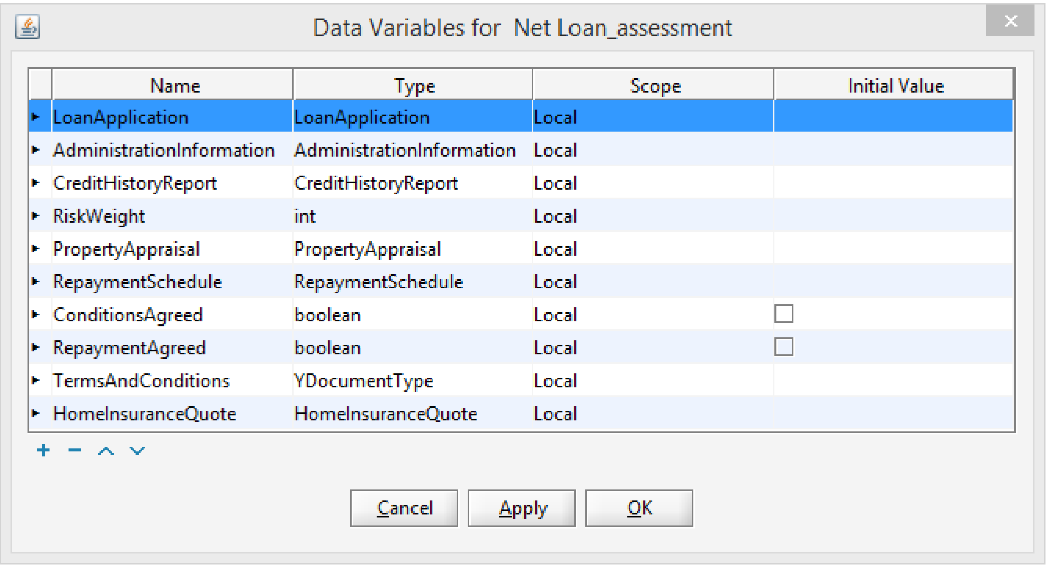
To define the data type, select the data definitions property in the specification section of the properties pane, then click on the action button (at the right of the property). Enter the XML schema data type definition in appendix A into the data definitions dialog, in the event of errors found, the bottom pane will show the error details. Click the done button.

Once the data type has been defined, it can be used when creating net and task variables. Net variable is created to store information relating to the loan assessment net that tasks within the net may need to read or update. Select the data variables property in the net section of the properties pane, then click on the action button to open the data variables dialog (see figure 2.6). Click on the + button at the bottom toolbar to add a new variable. Enter LoanApplication in the name field, click on the drop down arrow at the type field, and notice all the defined complex elements are now listed as a data type that can be used. Select LoanApplication as the type, local for scope and click OK. Follow the previous steps and create the remainder net variables as shown in figure 2.7.



Name	Type	Scope	Initial Value

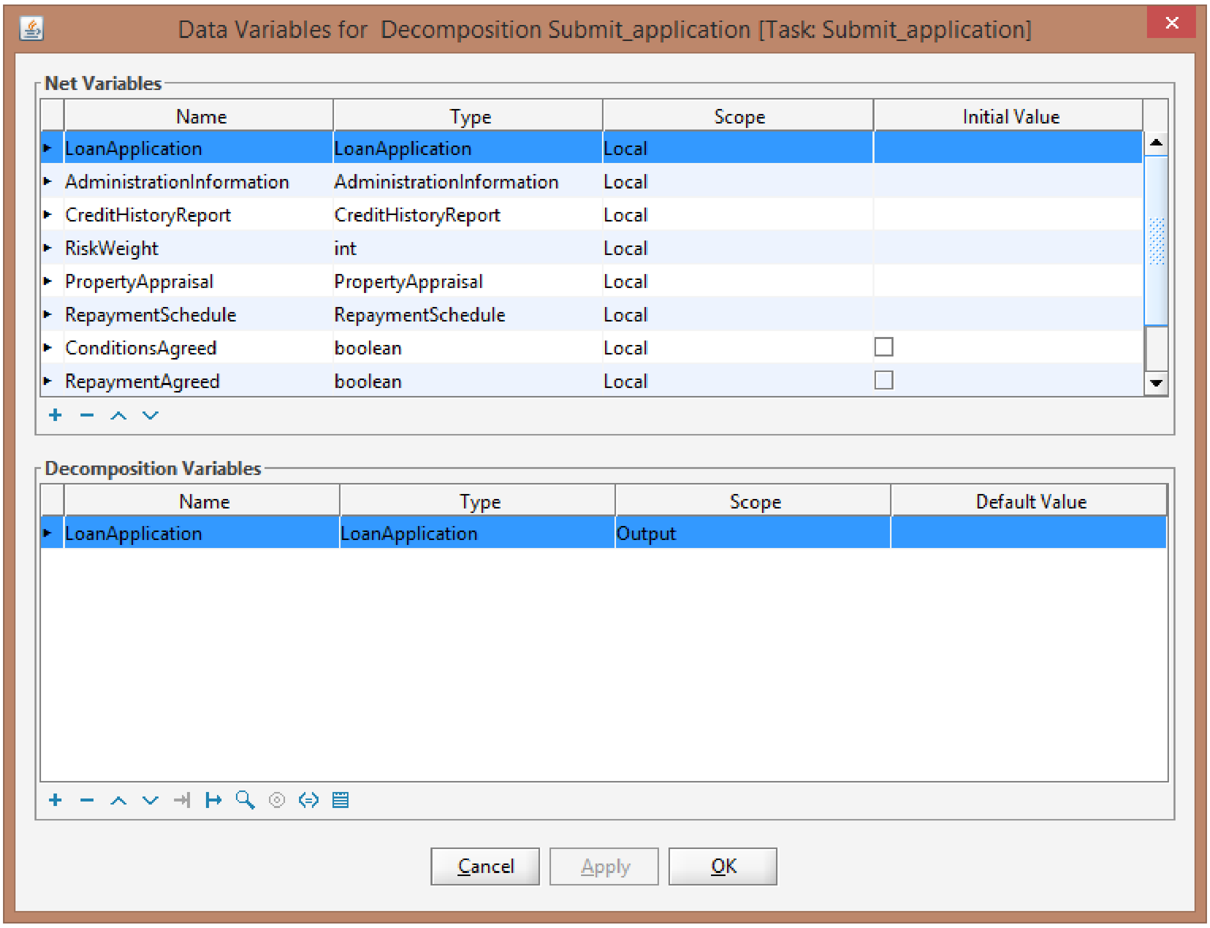
Figure 2.6. Data variable dialog



Name	Type	Scope	Initial Value
LoanApplication	LoanApplication	Local	
AdministrationInformation	AdministrationInformation	Local	
CreditHistoryReport	CreditHistoryReport	Local	
RiskWeight	int	Local	
PropertyAppraisal	PropertyAppraisal	Local	
RepaymentSchedule	RepaymentSchedule	Local	
ConditionsAgreed	boolean	Local	□
RepaymentAgreed	boolean	Local	□
TermsAndConditions	YDocumentType	Local	
HomeInsuranceQuote	HomeInsuranceQuote	Local	

Figure 2.7. Net variables of loan assessment net

The loan assessment net now requires task variables to be added so that it can transfer information between workflow users, and transfer of data between the net and its tasks. Task variables can be added to tasks in a similar way of how net variables are added. Select submit application task, then select the data variables property in the decomposition section of the properties pane, then click on the action button to open the data variables dialog. Click on the + button at the bottom toolbar of the decomposition variable section to add a new task variable. Enter LoanApplication in the name field. Click on the drop down arrow at the type field, select LoanApplication as the type, output for scope and click OK (see figure 2.8).



Data Variables for Decomposition Submit_application [Task: Submit_application]

Net Variables

Name	Type	Scope	Initial Value
LoanApplication	LoanApplication	Local	
AdministrationInformation	AdministrationInformation	Local	
CreditHistoryReport	CreditHistoryReport	Local	
RiskWeight	int	Local	
PropertyAppraisal	PropertyAppraisal	Local	
RepaymentSchedule	RepaymentSchedule	Local	
ConditionsAgreed	boolean	Local	□
RepaymentAgreed	boolean	Local	□

Decomposition Variables

Name	Type	Scope	Default Value
LoanApplication	LoanApplication	Output	

Figure 2.8. Submit application task variable

Follow the previous steps and create the task variables for each of the task listed in table 2.2.

Task	Variable name	Type	Scope
Check application form completeness	LoanApplication	LoanApplication	Input only
CompleteApplication	LoanApplication	LoanApplication	Input & output
	AdministrationInformation	AdministrationInformation	Input only
Update status	AdministrationInformation	AdministrationInformation	Input & output
Check credit history	LoanApplication	LoanApplication	Input only
	AdministrationInformation	AdministrationInformation	Input only
	CreditHistoryReport	CreditHistoryReport	Output

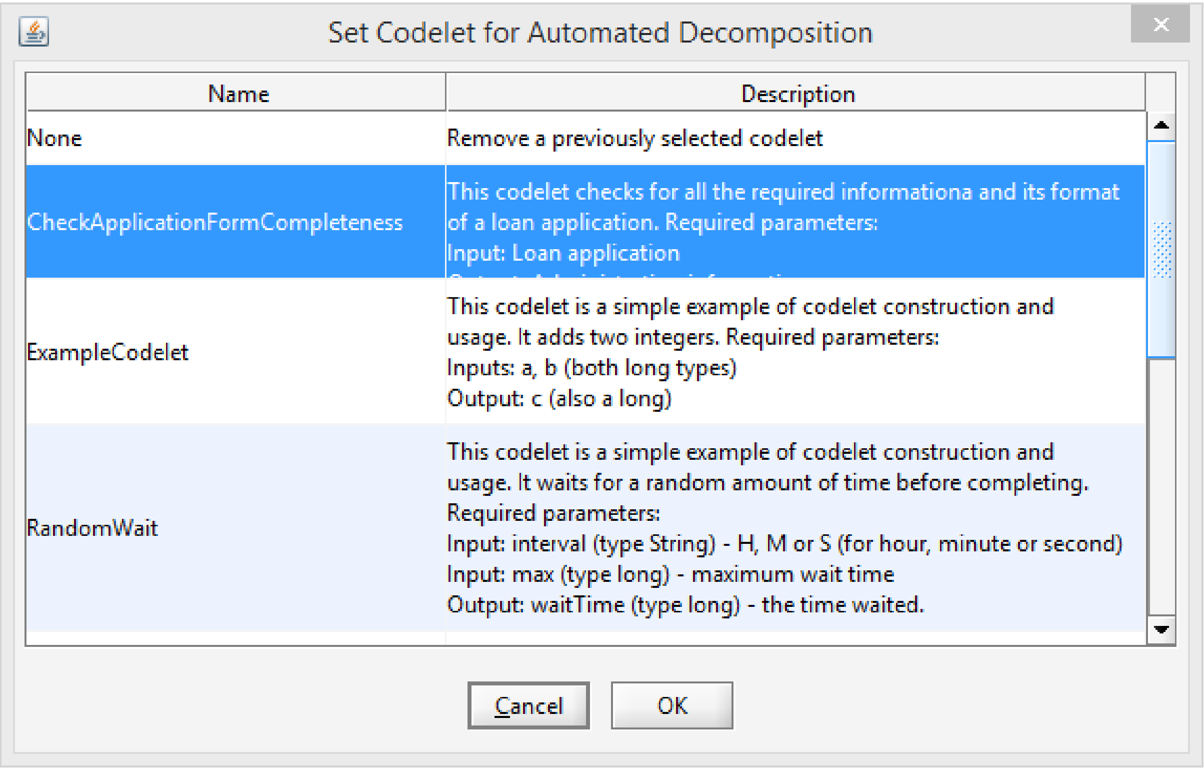
			only
Appraise property	LoanApplication	LoanApplication	Input only
	AdministrationInformation	AdministrationInformation	Input only
	PropertyAppraisal	PropertyAppraisal	Output only
Assess loan risk	RiskWeight	Int	Input & output
Assess eligibility	LoanApplication	LoanApplication	Input only
	AdministrationInformation	AdministrationInformation	Input & output
	RiskWeight	Int	Input
	PropertyAppraisal	PropertyAppraisal	Input only
Reject application	LoanApplication	LoanApplication	Input only
	AdministrationInformation	AdministrationInformation	Input & output
Prepare and send acceptance pack	LoanApplication	LoanApplication	Input only
	AdministrationInformation	AdministrationInformation	Input only
	RepaymentSchedule	RepaymentSchedule	Output only
Prepare and send home insurance pack	HomeInsuranceQuote	HomeInsuranceQuote	Output only
	TermsAndConditions	YDocumentType	Output only
VerifyRepaymentAgreement	LoanApplication	LoanApplication	Input only
	AdministrationInformation	AdministrationInformation	Input only
	ConditionsAgreed	Boolean	Output only
	RepaymentAgreed	Boolean	Output only
	RepaymentSchedule	RepaymentSchedule	InputOnly
Take final decision	LoanApplication	LoanApplication	Input only
	AdministrationInformation	AdministrationInformation	Input & output
	ConditionsAgreed	Boolean	Input only
	RepaymentAgreed	Boolean	Input only
	RepaymentSchedule	RepaymentSchedule	Input only

Table 2.2. Loan assessment task variables

Check application form completeness task will require an associated codelet (Java class) to enable a script to be executed to check the loan application and assign the check results to the AdministrationInformation variables. Create a Java class by copy and paste the Java codes provided in appendix B.

A detailed technical discussion of the codelet construction is beyond the scope of the lab exercise. However in essence, what the codelet does is to check for null value of each field in the loan application, if null is found, it will note that in the comment field accordingly. In addition, it also check to make sure the home phone and cell phone number has 10 digits in total. If it doesn't satisfy the criteria of a completed application, it will assign a status of incomplete to the application. It also set the case I.D as the application identifier and the submission date and time.

Place the CheckApplicationFormCompletness class file into the following YAWL's codelets folder (create the directories if it doesn't exists) – [YAWL INSTALL DIR]\engine\apache-tomcat-7.0.55\webapps\resourceService\WEB-INF\classes\org\yawlfoundation\yawl\resourcing\codelets. The root directory is depends on where YAWL is installed. In order for the new codelet to become available, the YAWL engine needs to be started (or restart the engine if it is currently running). Click on the control panel icon located at start/all programs/YAWL folder, then click the start button. To associate the codelet with the check application form completeness task, automate the task by ticking the automated checkbox in the decomposition section of the properties pane. Select the codelet property, then click on the action button (at the right of the property) to open the set codelet for automated decomposition dialog box. All the available codelets will be listed in the dialog (see figure 2.9). Select CheckApplicationFormCompleteness codelet and click the OK button. A green arrow will appear in the task to indicate that the task is now automated and has a codelet specified.

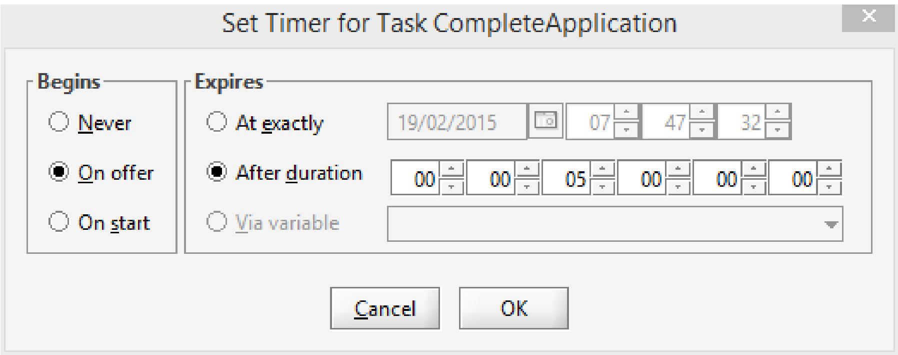


Set Codelet for Automated Decomposition

Name	Description
None	Remove a previously selected codelet
CheckApplicationFormCompleteness	This codelet checks for all the required informationa and its format of a loan application. Required parameters: Input: Loan application
ExampleCodelet	This codelet is a simple example of codelet construction and usage. It adds two integers. Required parameters: Inputs: a, b (both long types) Output: c (also a long)
RandomWait	This codelet is a simple example of codelet construction and usage. It waits for a random amount of time before completing. Required parameters: Input: interval (type String) - H, M or S (for hour, minute or second) Input: max (type long) - maximum wait time Output: waitTime (type long) - the time waited.

Figure 2.9. Available codelets

Both CompleteApplication and VerifyRepaymentAgreement task in the loan assessment process require a timer task to ensure timely response from applicants. If an application is not received back from the applicant in the required time frame, the task will expire and move onto the next task. Select the CompleteApplication task, then select the Timer property in the task section of the properties pane, then click on the action button (at the right of the property) to open the set timer dialog box and complete the set up as shown in figure 2.10. Click the OK button, the timer property is now set to Offer: P5D, where P stands for period and D for days. Repeat the same steps for VerifyRepaymentAgreement task, and set to 14 days for the duration.



Set Timer for Task CompleteApplication

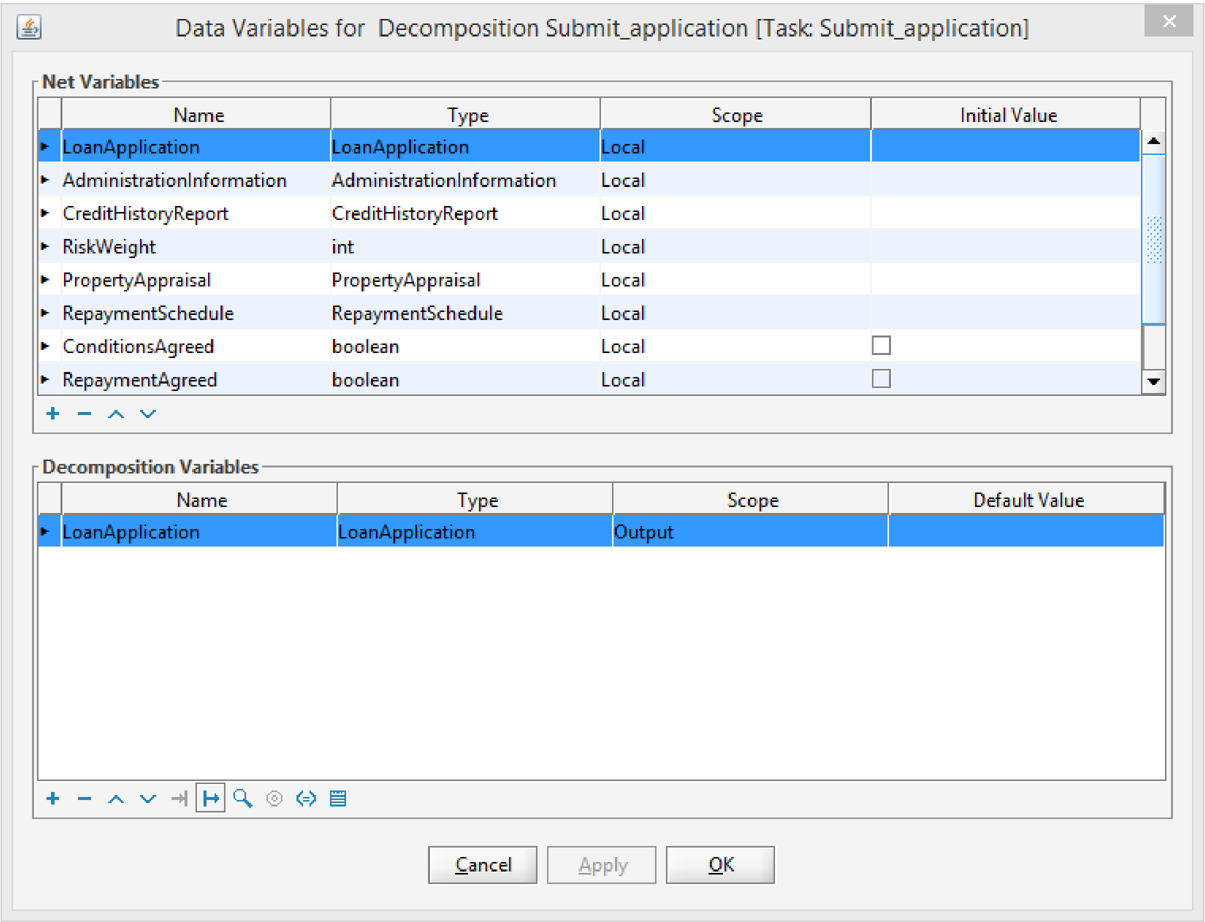
Begins
- ○ Never
- ● On offer
- ○ On start

Expires
- ○ At exactly 19/02/2015 07 47 32
- ● After duration 00 00 05 00 00 00
- ○ Via variable

Figure 2.10. Timer settings of complete application task

After setting up all the net and task variables, data bindings needs to be defined using XQuery expressions to enable value assign to a variable, and how a value is passed between net-level and task-level variables and vice-versa. Both input and output bindings can be assigned to any tasks (depending on its usage type) to allow the passing of state between nets and their tasks, and between tasks and workflow engine, users and web services.

To create an output binding, select the submit application task, open the data variables for decomposition dialog from the properties pane, select the LoanApplication task variable by clicking on the arrow next to the variable in the decomposition variables section as shown in figure 2.11.



Data Variables for Decomposition Submit_application [Task: Submit_application]

Net Variables

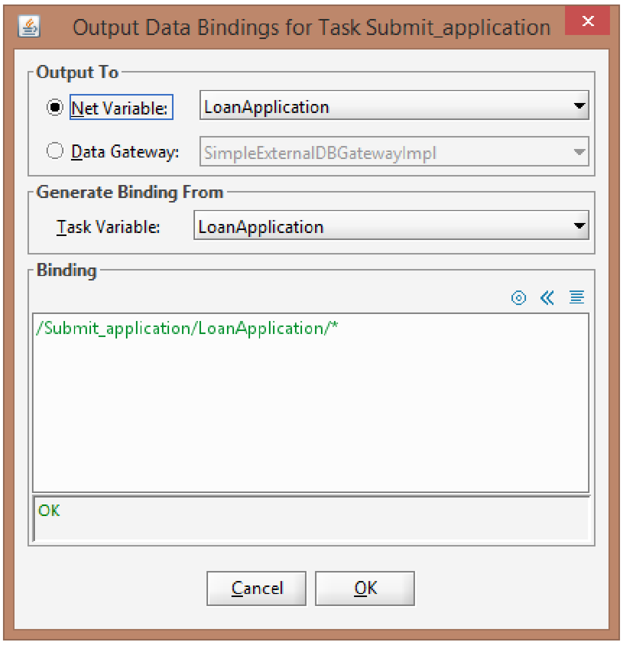
Name	Type	Scope	Initial Value
LoanApplication	LoanApplication	Local	
AdministrationInformation	AdministrationInformation	Local	
CreditHistoryReport	CreditHistoryReport	Local	
RiskWeight	int	Local	
PropertyAppraisal	PropertyAppraisal	Local	
RepaymentSchedule	RepaymentSchedule	Local	
ConditionsAgreed	boolean	Local	□
RepaymentAgreed	boolean	Local	□

Decomposition Variables

Name	Type	Scope	Default Value
LoanApplication	LoanApplication	Output	

Figure 2.11 Data variables for submit application task

Click on the output bindings button ↦ on the lower toolbar, and the output data bindings dialog box will appear. All the net variables are listed in the net variable drop down box at the output to section, and all the task variable(s) of the task are listed in the task variable drop down box in the generate binding from section. Make sure the both net and task variable are selected as shown in figure 2.12, and then there are two ways to add the XQuery, 1. click the generate and insert binding button ◎, or 2. type the XQuery directly into the XQuery box, click the OK button. To see the binding just created, click on the quick view bindings button 🔍 in the data variable dialog. Figure 2.13 shows the binding summary dialog box with the data bindings created for the task.



● Net Variable: LoanApplication

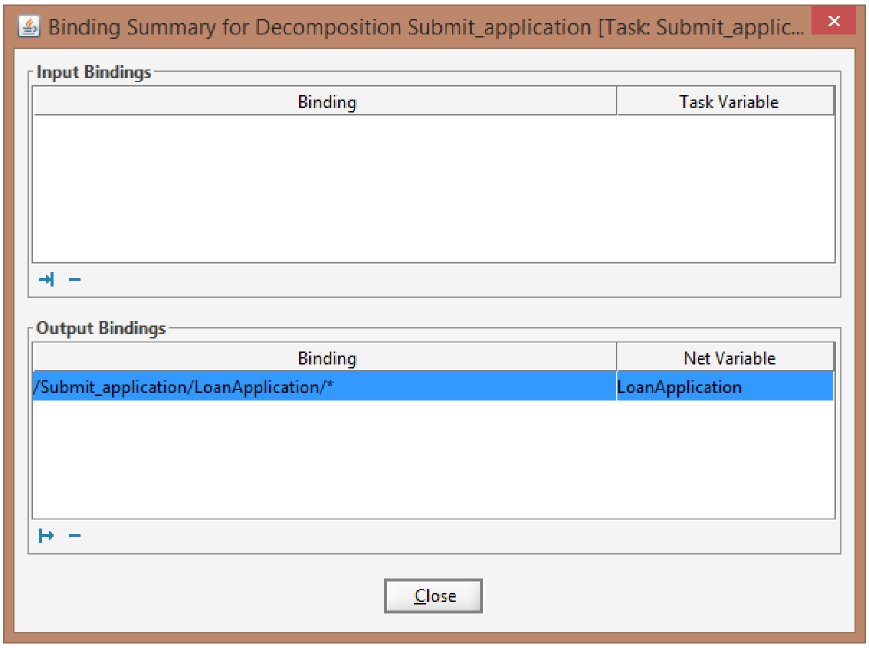
○ Data Gateway: SimpleExternalDBGatewayImpl

Task Variable: LoanApplication

```
/Submit_application/LoanApplication/*
```

OK

Figure 2.12. Passing the completed loan application values to a net variable


Binding Summary for Decomposition Submit_application [Task: Submit_applic...
Input Bindings
Output Bindings
Close

Input Bindings

Binding	Task Variable

Output Bindings

Binding	Net Variable
/Submit_application/LoanApplication/*	LoanApplication

Figure 2.13. Established data bindings with XQuery mapping

Creating an input binding is done in a similar way as the output binding by clicking the input bindings button instead of the output bindings button.

Follow the previous steps and create data bindings for each of the task listed in table 2.3.

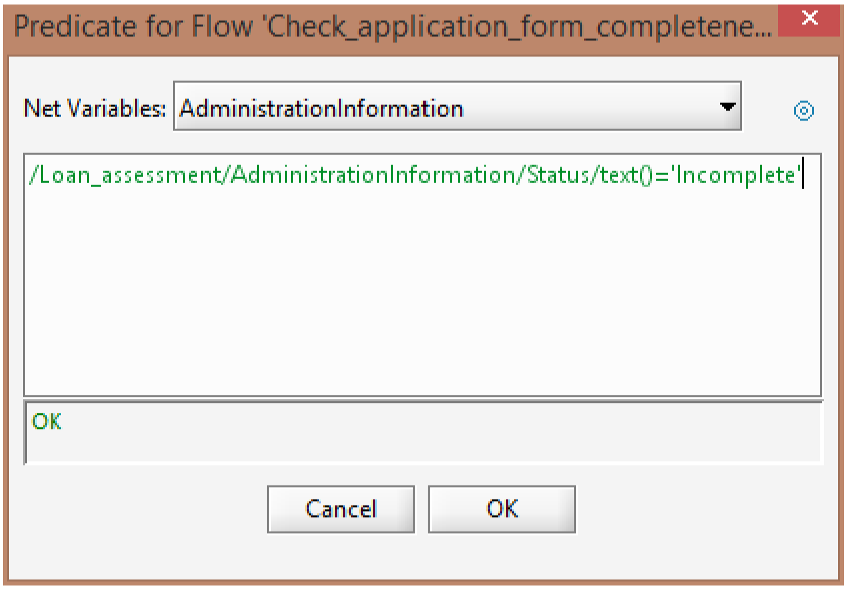
Task	Task/net variable	Expression	Parameter type
CompleteApplication	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	LoanApplication	{/CompleteApplication/LoanApplication/*}	Output
Check application form completeness	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Check_application_form_completeness/AdministrationInformation/*	Output
Update status	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	AdministrationInformation	<ApplicationIdentifier>{/Update_status/AdministrationInformation/ApplicationIdentifier/text()}</ApplicationIdentifier> <SubmissionDateAndTime>{/Update_status/AdministrationInformation/SubmissionDateAndTime/text()}</SubmissionDateAndTime><RevisionDateAndTime>{current-DateTime()}</RevisionDateAndTime> <Status>Canceled</Status> <Comments>{/Update_status/AdministrationInformation/Comments/text()}</Comments> <Eligibility>{false()}</Eligibility> <LoanOfficerIdentifier>{/Update_status/AdministrationInformation/LoanOfficerIdentifier/text()}</LoanOfficerIdentifier>	Output
Check credit history	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	CreditHistoryReport	{/Check_credit_history/CreditHistoryReport/*}	Output
Appraise property	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	PropertyAppraisal	{/Appraise_property/PropertyAppraisal/*}	Output
Assess loan risk	RiskWeight	{if(/Loan_assessment/CreditHistoryReport/CreditAssessment/Result/text()='B') then 0 else if(/Loan_assessment/CreditHistoryReport/CreditAssessmen	Input

		t/Result/text()='BB') then 20 else if(/Loan_assessment/CreditHistoryReport/CreditAssessment/Result/text()='BBB') then 40 else if(/Loan_assessment/CreditHistoryReport/CreditAssessment/Result/text()='A') then 60 else if(/Loan_assessment/CreditHistoryReport/CreditAssessment/Result/text()='AA') then 80 else 100}	
	RiskWeight	{/Assess_loan_risk/RiskWeight/text()}	Output
Assess eligibility	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	RiskWeight	{/Loan_assessment/RiskWeight/text()}	Input
	PropertyAppraisal	{/Loan_assessment/PropertyAppraisal/*}	Input
	AdministrationInformation	<ApplicationIdentifier>{/Assess_eligibility/AdministrationInformation/ApplicationIdentifier/text()}</ApplicationIdentifier> <SubmissionDateAndTime>{/Assess_eligibility/AdministrationInformation/SubmissionDateAndTime/text()}</SubmissionDateAndTime> <RevisionDateAndTime>{current-dateTime()}</RevisionDateAndTime> <Status>Assessed</Status> <Comments>{/Assess_eligibility/AdministrationInformation/Comments/text()}</Comments> <Eligibility>{/Assess_eligibility/AdministrationInformation/Eligibility/text()}</Eligibility> <LoanOfficerIdentifier>{/Assess_eligibility/AdministrationInformation/LoanOfficerIdentifier/text()}</LoanOfficerIdentifier>	Output
Reject application	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	<ApplicationIdentifier>{/Loan_assessment/AdministrationInformation/ApplicationIdentifier/text()}</ApplicationIdentifier> <SubmissionDateAndTime>{/Loan_assessment/AdministrationInformation/SubmissionDateAndTime/text()}</SubmissionDateAndTime> <RevisionDateAndTime>{current-dateTime()}</RevisionDateAndTime> <Status>Rejected</Status> <Comments>{/Loan_assessment/AdministrationInformation/Comments/text()}</Comments> <Eligibility>{/Loan_assessment/AdministrationInformation/	Input

		Eligibility/text()}</Eligibility> <LoanOfficerIdentifier>{/Loan_assessment/AdministrationInformation/LoanOfficerIdentifier/text()}</LoanOfficerIdentifier>	
	AdministrationInformation	{/Reject_application/AdministrationInformation/*}	Output
Prepare and send acceptance pack	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	RepaymentSchedule	{/Prepare_and_send_acceptance_pack/RepaymentSchedule/*}	Output
Prepare and send home insurance quote	TermsAndConditions	{/Prepare_and_send_home_insurance_quote/TermsAndConditions/*}	Output
	HomeInsuranceQuote	{/Prepare_and_send_home_insurance_quote/HomeInsuranceQuote/*}	Output
VerifyRepaymentAgreement	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	RepaymentSchedule	{/Loan_assessment/RepaymentSchedule/*}	Input
	RepaymentAgreed	{/VerifyRepaymentAgreement/RepaymentAgreed/text()}	Output
	ConditionsAgreed	{/VerifyRepaymentAgreement/ConditionsAgreed/text()}	Output
Take final decision	LoanApplication	{/Loan_assessment/LoanApplication/*}	Input
	AdministrationInformation	{/Loan_assessment/AdministrationInformation/*}	Input
	RepaymentSchedule	{/Loan_assessment/RepaymentSchedule/*}	Input
	RepaymentAgreed	{/Loan_assessment/RepaymentAgreed/text()}	Input
	ConditionsAgreed	{/Loan_assessment/ConditionsAgreed/text()}	Input
	AdministrationInformation	{/Take_final_decision/AdministrationInformation/*}	Output

Table 2.3. Parameter mappings

It is now possible to define which flow should be activated at runtime at each of the decision points, i.e. XOR splits. Click on the check application form completeness task, select the split predicates property in the task section of the properties pane, then click on the action button (at the right of the property) to open the split predicates dialog box.   All the outgoing flows of the split and its associated flow expression (predicate) will be listed in the split predicates dialog box that appears. There should be two target task listed – CompleteApplication and unnamed task. The arrow buttons at the bottom toolbar of the list allow reordering the evaluation sequence of the predicates. The bottom predicate will form the default flow that gets executed if all other predicates evaluated to be false. Ensure the unnamed target task is at the bottom in this instance. To specify the predicate, select the CompleteApplication target task, then click on the action button and complete the set up as shown in figure 2.14. Enter /Loan_assessment/AdministrationInformation/Status/text()='Incomplete' in the predicate for flow dialog box and click the OK button. Leave the predicate of the unnamed task as true() to allow the flow will always execute to avoid deadlock when all other predicates are false (see figure 2.15). Repeat the steps to define rest of the flow details as listed in table 2.4, and ensure to follow the evaluation sequence of the predicates.

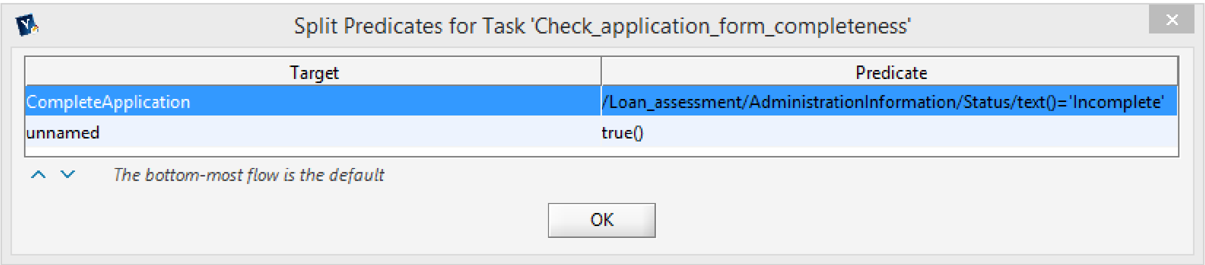


Net Variables: AdministrationInformation

/Loan_assessment/AdministrationInformation/Status/text()='Incomplete'

OK

Figure 2.14. Predicate for flow Split predicates


Split Predicates for Task 'Check_application_form_completeness'
OK

Target	Predicate
CompleteApplication	/Loan_assessment/AdministrationInformation/Status/text()='Incomplete'
unnamed	true()

The bottom-most flow is the default

Figure 2.15. Predicate for flow

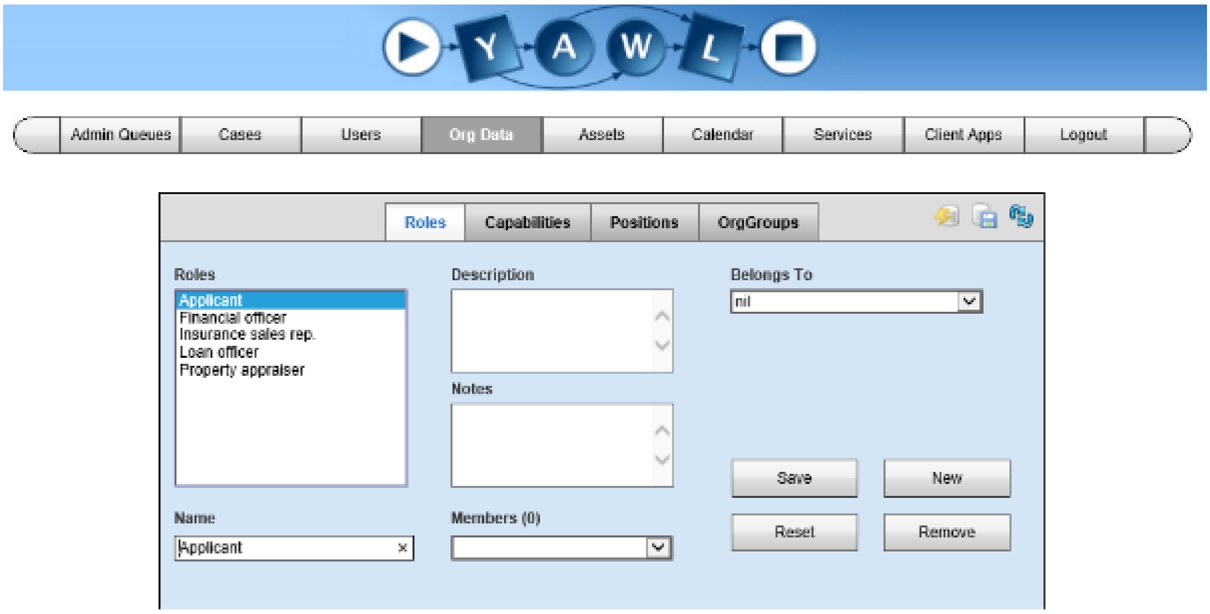
Task	Target task	Predicate
CompleteApplication	Update status	timer(CompleteApplication)='expired'
	Check application form completeness	true()
Assess eligibility	Reject application	/Loan_assessment/AdministrationInformation/Eligibility/text()='false'
	Prepare and send acceptance pack	true()

Prepare and send acceptance pack	Prepare and send home insurance quote	/Loan_assessment/LoanApplication/InsuranceQuoteRequired/Required/text()='true'
	VerifyRepaymentAgreement	true()
VerifyRepaymentAgreement	Update status	timer(VerifyRepaymentAgreement)='expired'
	Take final decision	true()

Table 2.4. Loan assessment flow details

2.3. Resource perspective

Now that the loan assessment process is completed, it is time to specify which participants should be doing which tasks. In order to assign tasks to participants, participants and their roles must first be defined in YAWL control centre. Navigate to the resource service's web UI either by click on the logon button in the YAWL control panel or by pointing your browser at http://localhost:8080/resourceService, log on with the generic user id 'admin' and password 'YAWL'. Go to the organisational data management screen showing in figure 2.16 by clicking on the Org data heading. To add a new role, click the new button. Enter loan officer in the name field, and click the add button, the role will appear in the roles box after it has been added. Now add the financial officer, property appraiser, insurance sales rep. and applicant in the same way.



Roles
Applicant
Financial officer
Insurance sales rep.
Loan officer
Property appraiser

Description

Notes

Belongs To: nil

Name: Applicant

Members (0)

Figure 2.16. Organisational data management

Once all roles have been added, click on the users heading to go to the user management screen (see figure 2.17) to add the participants.

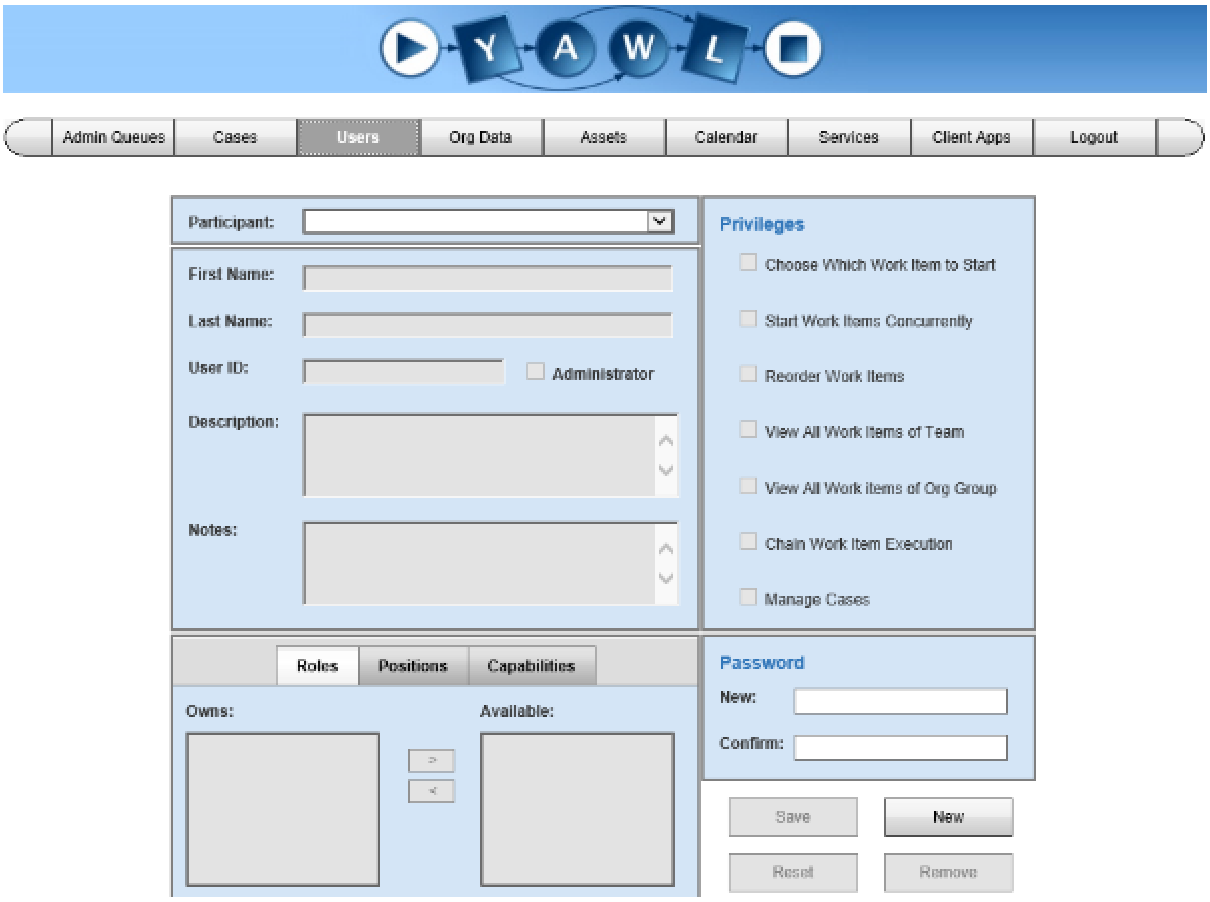


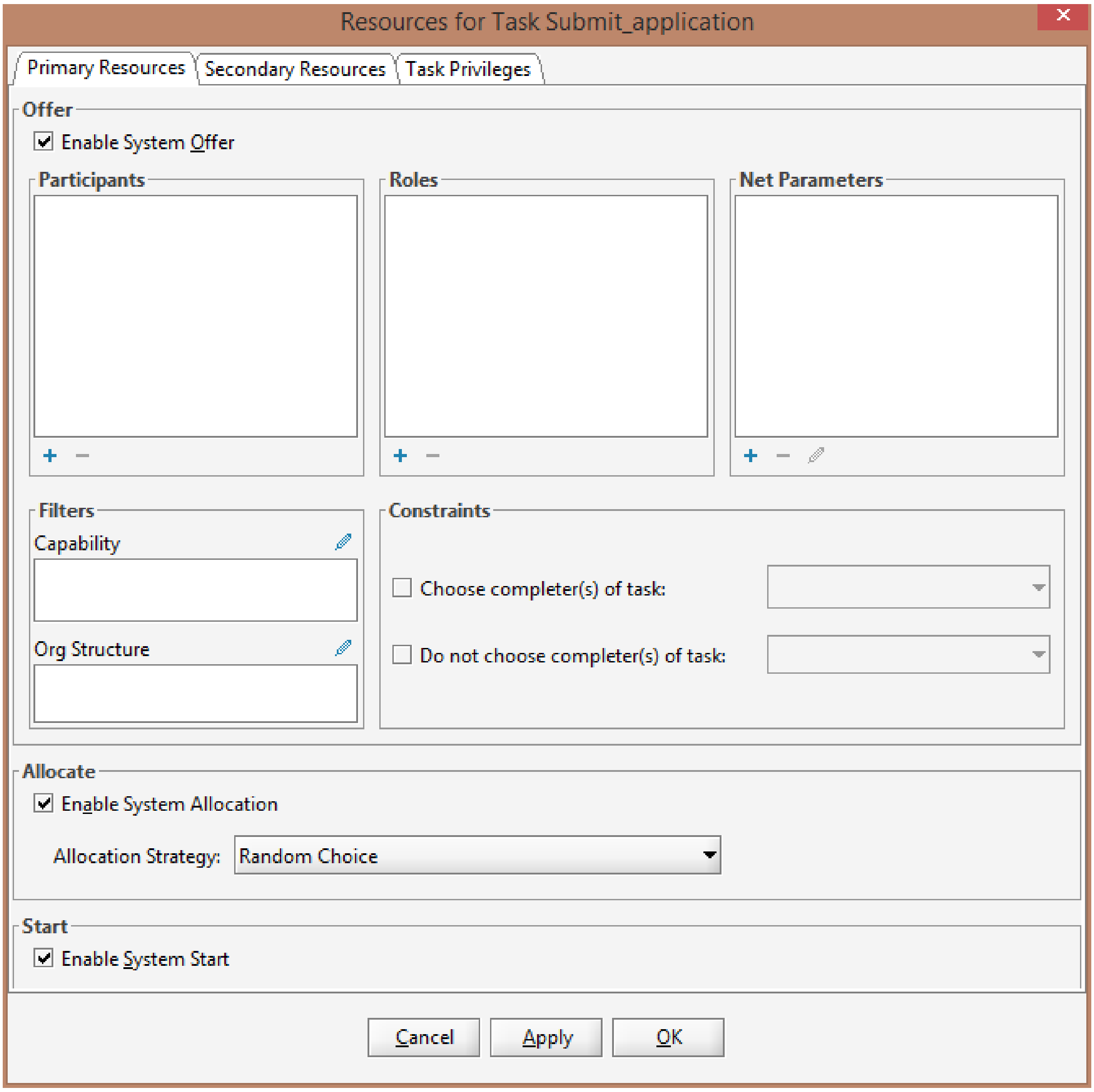
Figure 2.17. User management

To add a new participant, click the new button and enter the participant details. For the purpose of the exercise, enter loan officer in the first name field, A in the last name field, LOA as the user I.D, and give it a password in the password section. To assign Loan officer A to the role of Loan officer, ensure the roles tab is selected at the bottom left section of the screen. The 'owns' box shows the role(s) a participant currently holds and the 'available' box shows all the roles that a participant can be assigned to. Select loan officer from the list of available roles and click on the left pointing arrow to assign the loan officer role to loan officer A. Create the remaining users listed in table 2.5.

First name	Last name	User ID	Roles
Loan officer	B	LOB	Loan officer
Finance officer	A	FOA	Finance officer
Property appraiser	A	PAA	Property appraiser
Insurance sales rep.	A	ISRA	Insurance sales rep.
Applicant	A	AA	Applicant

Table 2.5. Loan assessment process participants

Once the above roles and participants are defined, resources can be allocated to tasks by specifying the interaction strategy via YAWL editor. Click on submit application task and select the resourcing property in the task section of the properties pane, then click on the action button (at the right of the property) to open the resources dialog box, and tick the enable system offer, allocation and start as shown in figure 2.18. What it does is when a case start, the system will automatically offer, allocate, and immediately start the work items of the task at run time.



Offer
☑ Enable System Offer

□ Choose completer(s) of task:

□ Do not choose completer(s) of task:

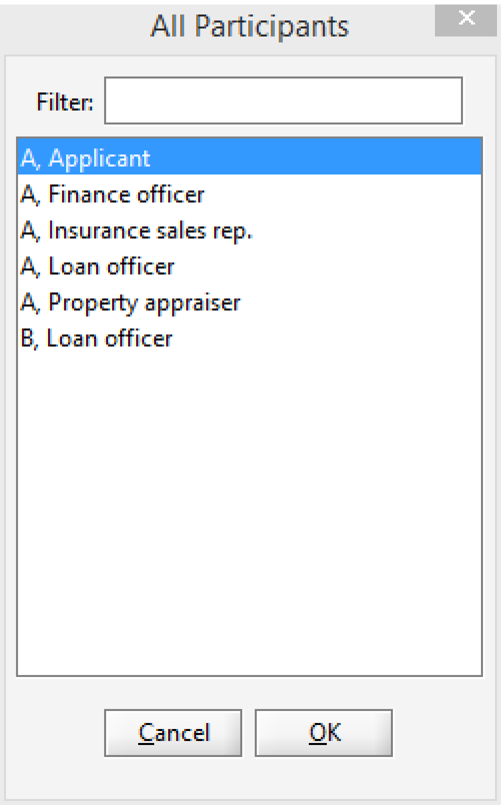
Allocate
☑ Enable System Allocation

Allocation Strategy: Random Choice

Start
☑ Enable System Start

Figure 2.18. Resources dialog box

Task then can be assigned to specific participants and/or roles, click + sign in the participants section, select A, Applicant from the all participants list and click OK (see figure 2.19), then click the OK button again to close the resources dialog. The applicant is now the primary resources that the work item of the selected task will be offered to at runtime by the system.



A, Applicant
A, Finance officer
A, Insurance sales rep.
A, Loan officer
A, Property appraiser
B, Loan officer

Figure 2.19. Participants dialog

Use the resourcing properties to set up resources for the tasks listed in table 2.6.

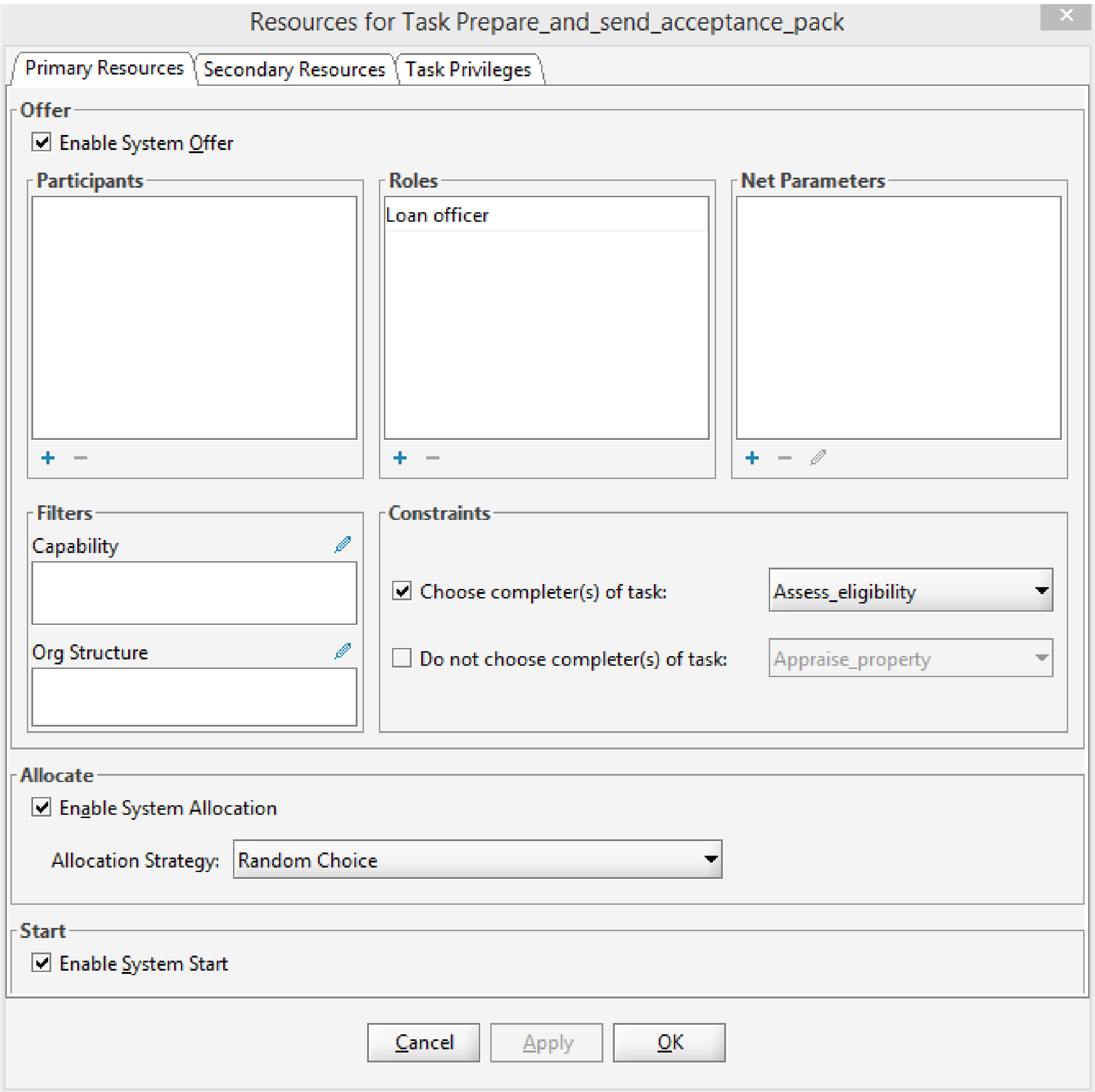
Task	Enable system offer	Enable system allocation	Enable system start	Participants	Roles
CompleteApplication	✓	✓	✓	Applicant A	
Check credit history	✓	✓	✓		Financial officer
Appraise property	✓	✓	✓		Property appraiser
Assess eligibility	✓	✓	✓		Loan officer
Reject application,	✓	✓	✓		Loan officer
Prepare and home insurance quote	✓	✓	✓		Insurance sales rep.

Table 2.6. Resource setting

The remaining three tasks – prepare and send acceptance pack, VerifyRepaymentAgreement and take final decision requires business rules to be defined in its resourcing.

- Prepare and send acceptance pack

Tick enable system offer, enable system allocate and enable system start. Add loan officer to the roles box. Add runtime constraints to ensure the task are allocated to the same loan officer who have previously completed work items of assess eligibility task in the current process instance (see figure 2.20).



Resources for Task Prepare_and_send_acceptance_pack

Offer

☑ Enable System Offer

Participants:

Roles: Loan officer

Net Parameters:

Filters

Capability:

Org Structure:

Constraints

☑ Choose completer(s) of task: Assess_eligibility

□ Do not choose completer(s) of task: Appraise_property

Allocate

☑ Enable System Allocation

Allocation Strategy: Random Choice

Start

☑ Enable System Start

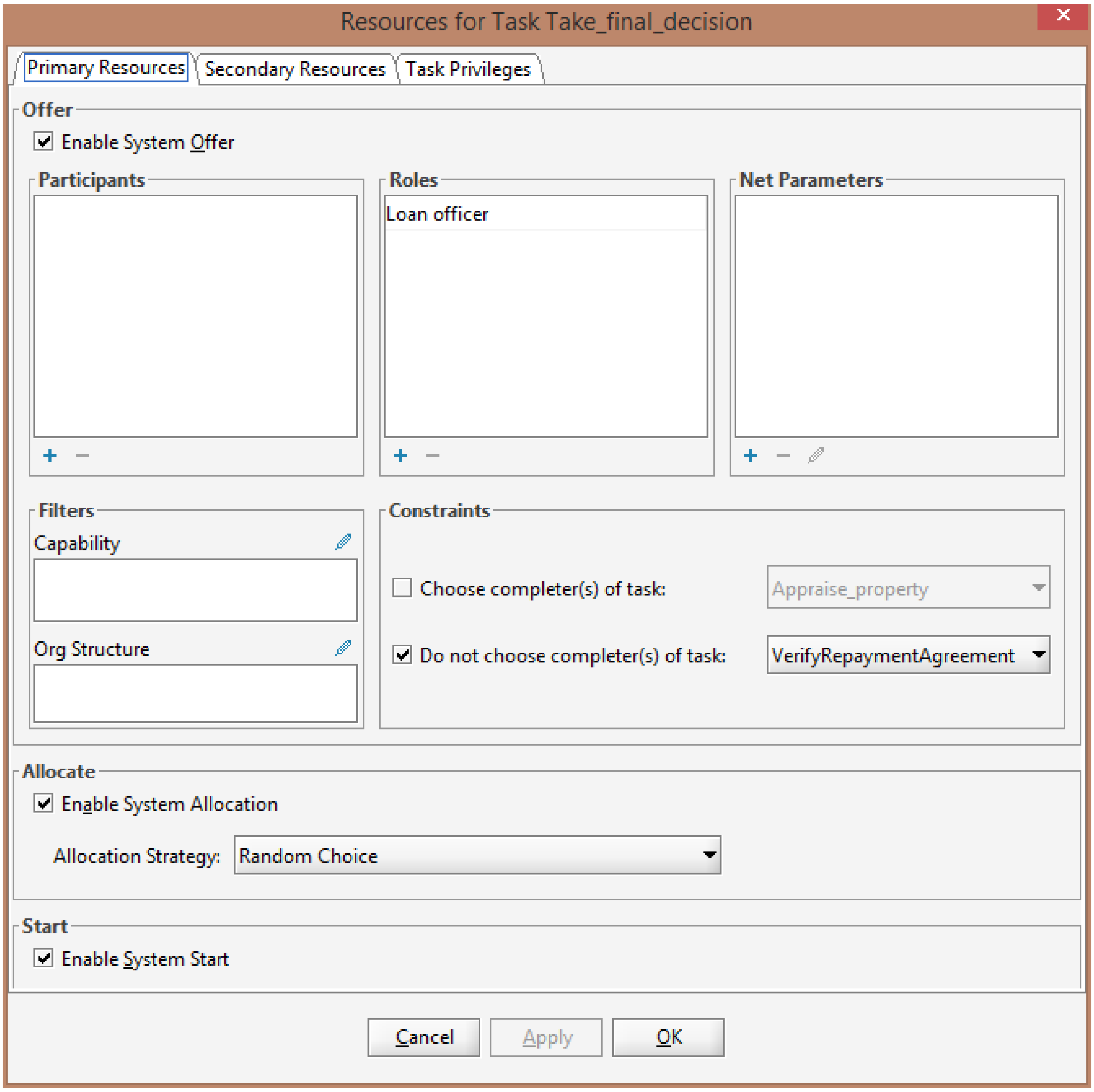
Figure 2.20. Resourcing with choose completer(s) of task constraints

- VerifyRepaymentAgreement

Tick enable system offer, enable system allocate and enable system start. Add loan officer to the roles box. Add runtime constraints to ensure the task are allocated to the same loan officer who have previously completed work items of prepare and send acceptance pack task in the current process instance.

- Take final decision

Tick enable system offer, enable system allocate and enable system start. Add loan officer to the roles box. Add runtime constraints to ensure the task are allocated to a different loan officer who have previously completed work items of VerifyRepaymentAgreement task in the current process instance (see figure 2.21).



Resources for Task Take_final_decision

Offer

☑ Enable System Offer

Participants

Roles

Loan officer

Net Parameters

Filters

Capability

Org Structure

Constraints

□ Choose completer(s) of task: Appraise_property

☑ Do not choose completer(s) of task: VerifyRepaymentAgreement

Allocate

☑ Enable System Allocation

Allocation Strategy: Random Choice

Start

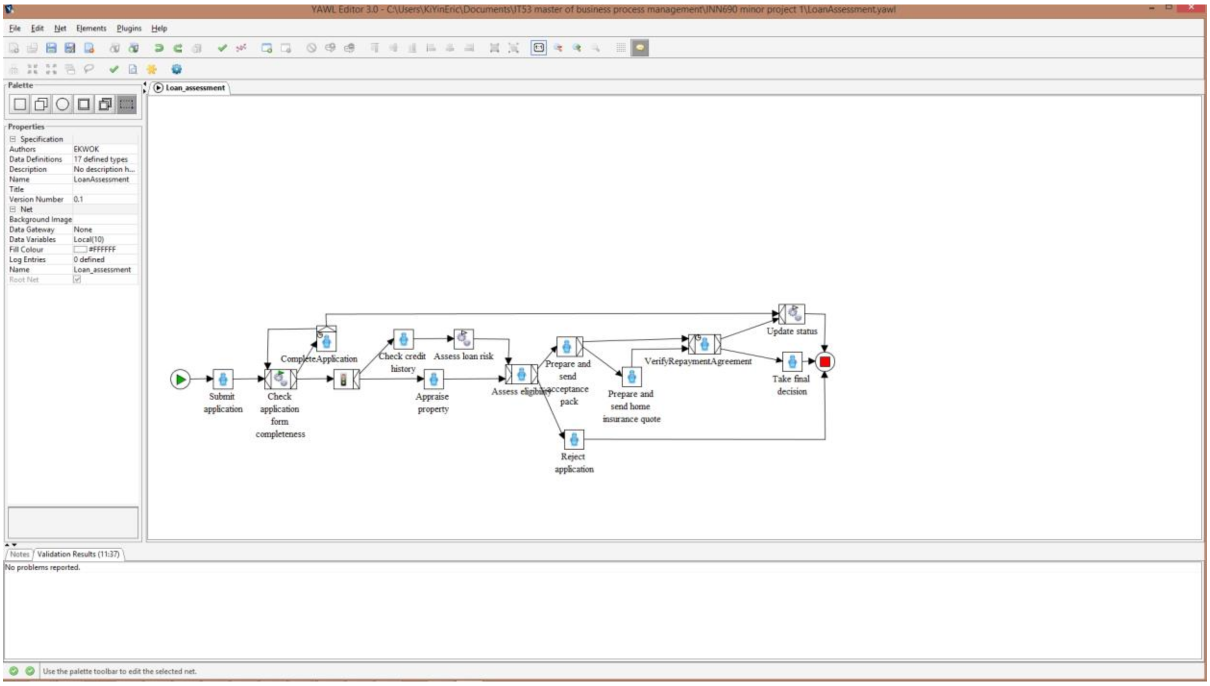
☑ Enable System Start

Figure 2.21. Resourcing with do not choose completer(s) of task constraints

3.0. Validate and deploy the process

The loan assessment net is now fully completed, it is important to check the process that just been created for any problems that would prevent the process from executing and complete successfully.

Click the validate button ✓ on the tool bar, and YAWL will validate the specification (verification will also run by default each time the specification is saved). Any problems found and its potential solution will be shown at the info pane, under the validation results tab. Figure 3.0 shows the validation result.



YAWL Editor 3.0 - C:\Users\KiYinEric\Documents\IT53 master of business process management\INN690 minor project 1\LoanAssessment.yawl

⊟ Specification	
Authors	EKWOK
Data Definitions	17 defined types
Description	No description h...
Name	LoanAssessment
Title	
Version Number	0.1
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	Local(10)
Fill Colour	□ #FFFFFF
Log Entries	0 defined
Name	Loan_assessment
Root Net	☑

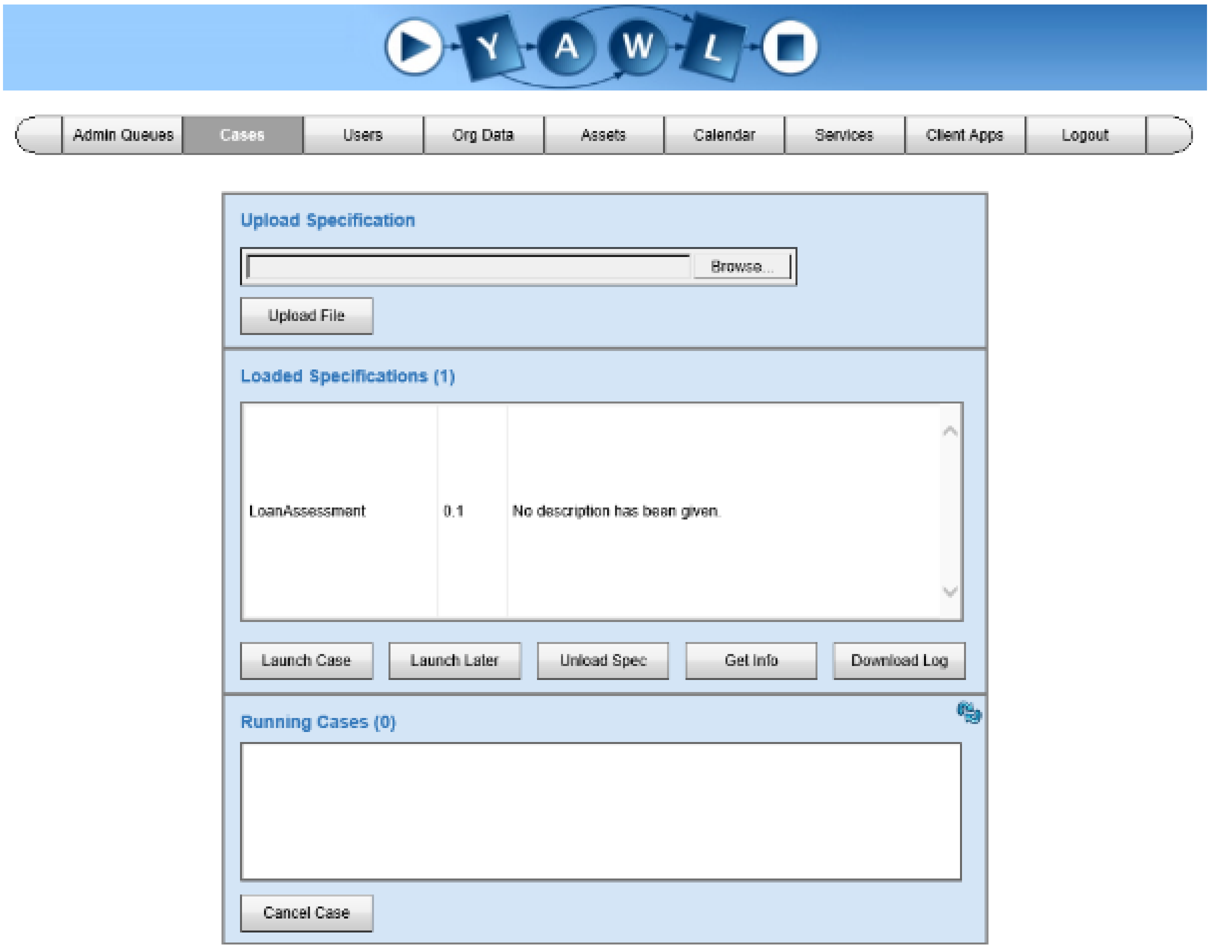
No problems reported.

Use the palette toolbar to edit the selected net.

Figure 3.0. Loan assessment process with no problems found

If desired, further detailed analysis can be conducted by using the analyse this specification toolbar button . A number of potential problems with a workflow can be automatically spotted with analysis. Examples include spotting potential deadlock situations, unnecessary cancellation set members, and unnecessary or-join decorators.

After the loan assessment net have been validated to be correct, it is ready for deployment. Click the upload this specification to YAWL engine button and follows the prompt to deploy. Or alternatively, click the logon button in YAWL control panel to bring up YAWL control centre, and click on cases in the top menu as shown in figure 3.1. To upload the workflow specification, click on the browse button to bring up the file browser dialog box, double click on the loan assessment file to select the file, then click the upload file button. When the specification is uploaded, it is validated against the YAWL specification schema for validity. If there is a problem with the upload, an appropriate error message is displayed in a popup dialog. If the specification has been uploaded successfully, it will be shown in the loaded specifications section (see figure 3.1).

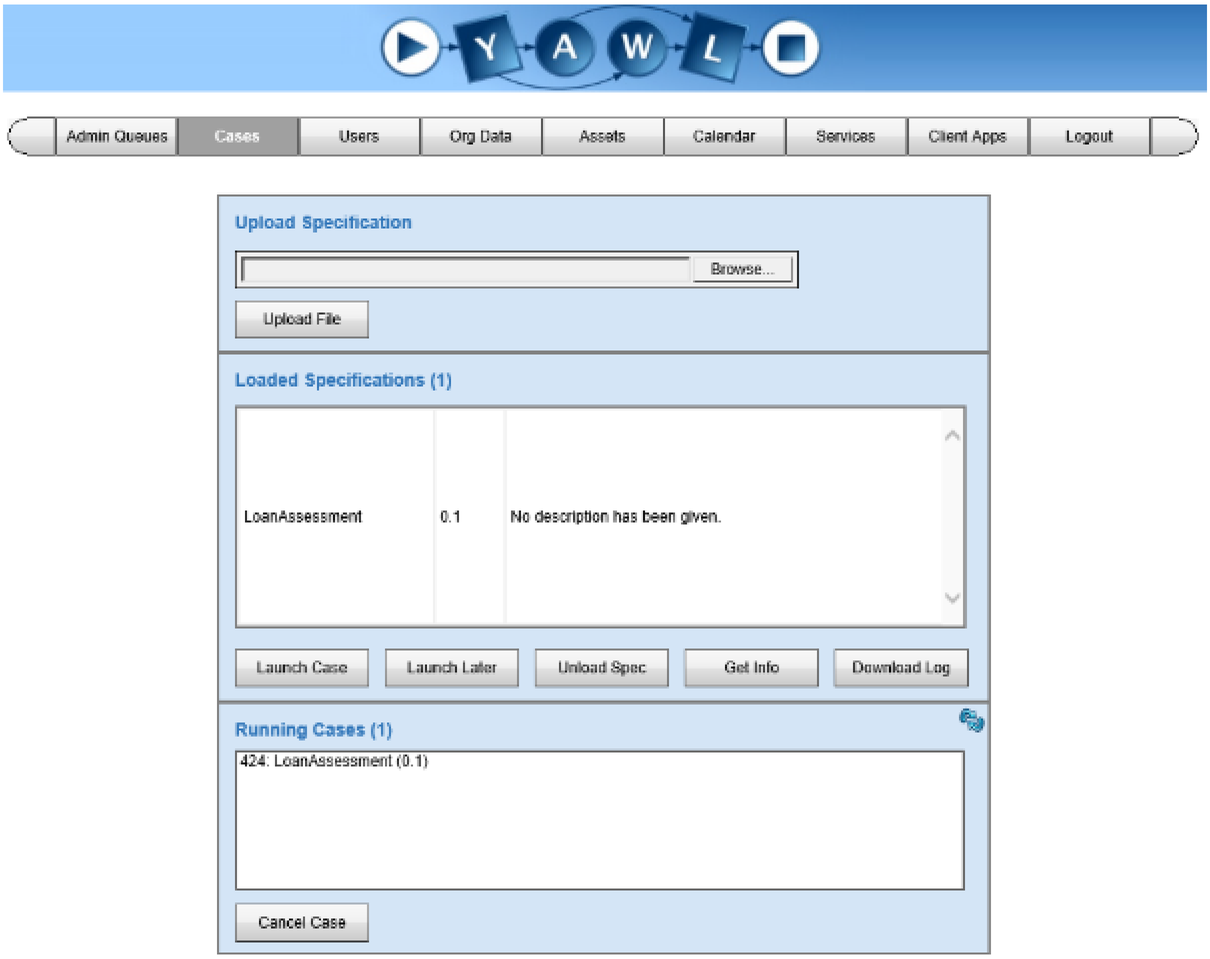


LoanAssessment	0.1	No description has been given.

Figure 3.1. Case management

4.0. Execute the process

After the workflow specification have been uploaded, to launch a case, select the loan assessment specification and click the launch case button. All the current active/running case(s) of the loan assessment process will be shown in the running cases section (see figure 4.0). It is now ready to give the loan assessment process a test run to see its execution. Logout of YAWL control centre and log back on as the applicant, use the password that was created during the setup of participants.

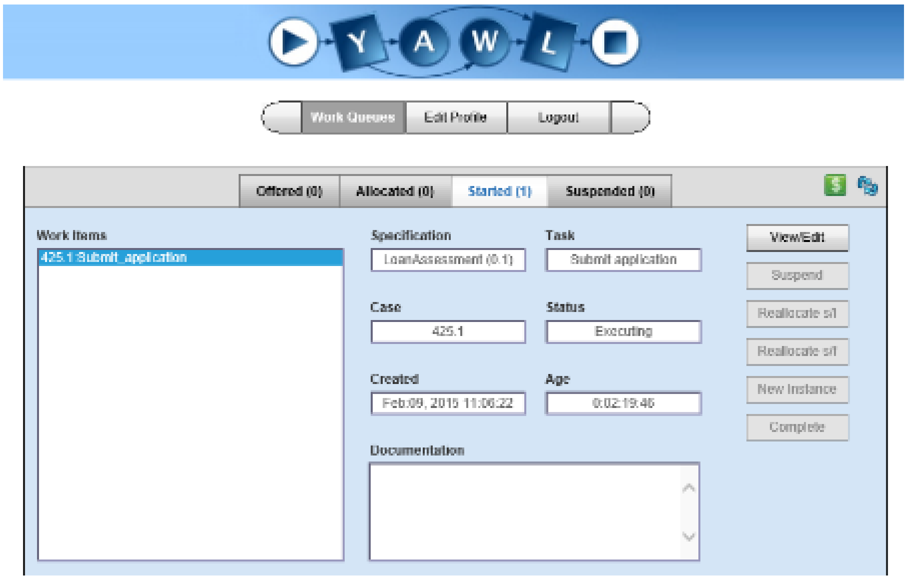


LoanAssessment	0.1	No description has been given.

424: LoanAssessment (0.1)

Figure 4.0. Case management showing running cases

Click on started in the menu, select the submit application task that is listed under work items (see figure 4.1) and click the view/edit button to bring up the loan application. Fill in sample data as shown in figure 4.2 and click the complete button.



Work Items: 425.1:Submit_application

Specification: LoanAssessment (0.1)

Task: Submit application

Case: 425.1

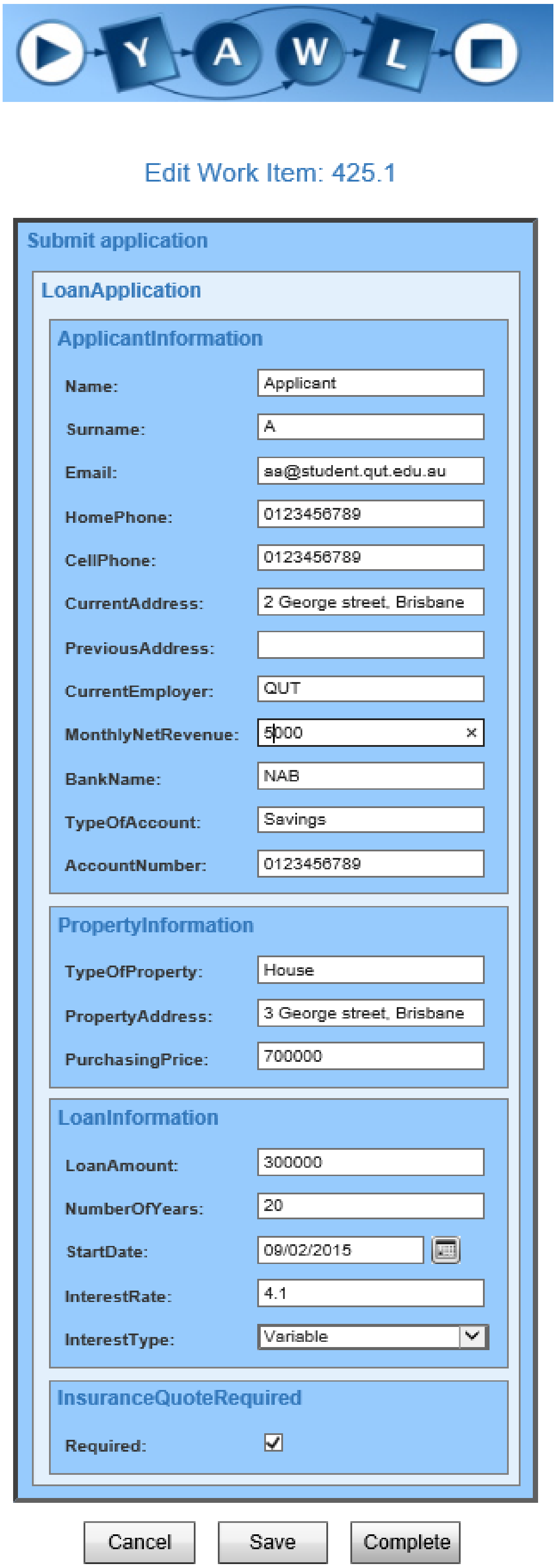
Status: Executing

Created: Feb:09, 2015 11:06:22

Age: 0:02:19:46

Documentation:

Figure 4.1. Applicant screen with submit application task



Edit Work Item: 425.1

Submit application

LoanApplication

ApplicantInformation

Name:	Applicant
Surname:	A
Email:	aa@student.qut.edu.au
HomePhone:	0123456789
CellPhone:	0123456789
CurrentAddress:	2 George street, Brisbane
PreviousAddress:	
CurrentEmployer:	QUT
MonthlyNetRevenue:	5000
BankName:	NAB
TypeOfAccount:	Savings
AccountNumber:	0123456789

PropertyInformation

TypeOfProperty:	House
PropertyAddress:	3 George street, Brisbane
PurchasingPrice:	700000

LoanInformation

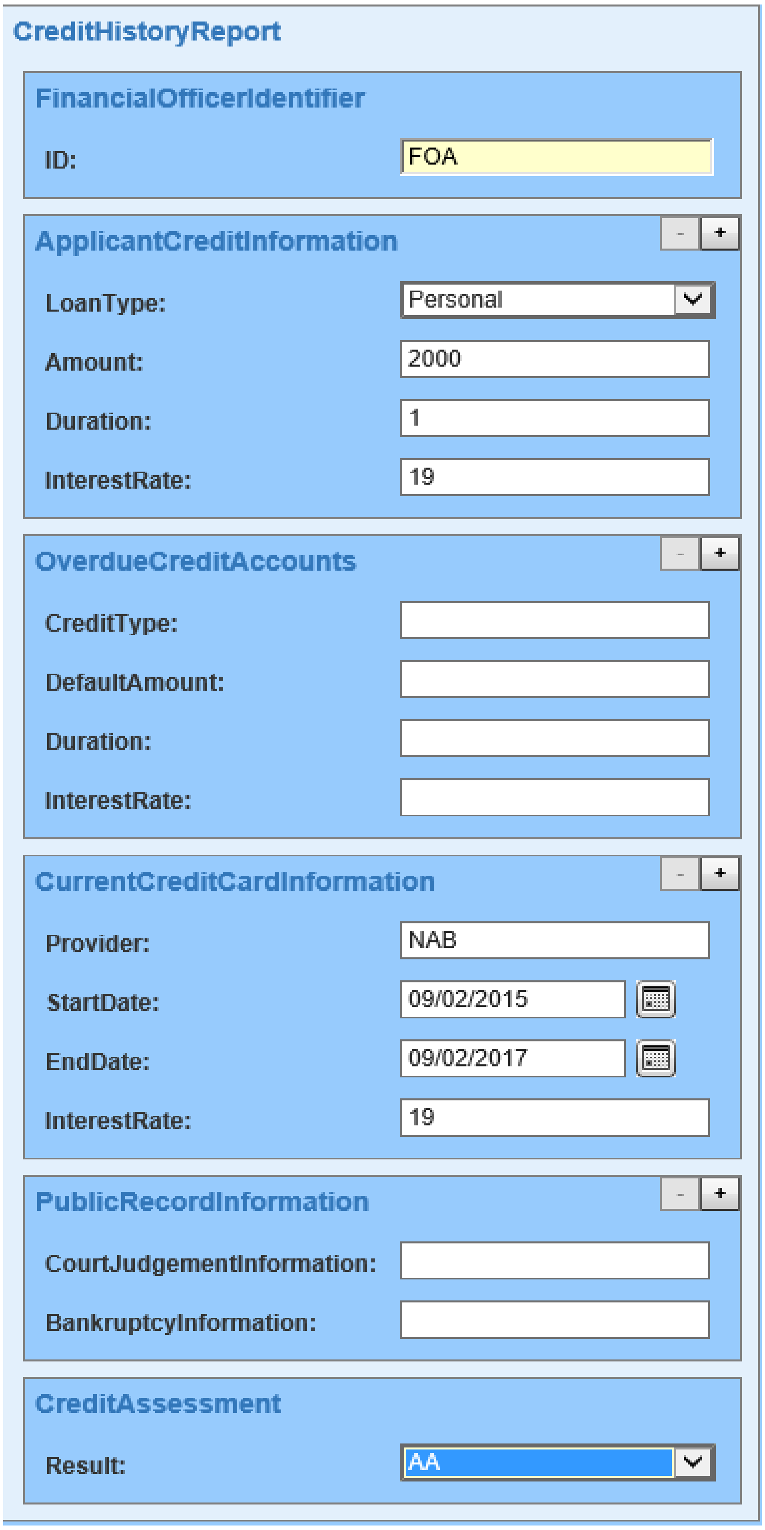
LoanAmount:	300000
NumberOfYears:	20
StartDate:	09/02/2015
InterestRate:	4.1
InterestType:	Variable

InsuranceQuoteRequired

Required:	☑

Figure 4.2. Loan application form

Logout and log back in as the financial officer. Click on started in the menu, select the check credit history task listed under work items and click the view/edit button to bring up the credit history report. Fill in sample data as shown in figure 4.3 and click the complete button. Note that how an uneditable copy of the completed loan application and administration information are attached. The information in the administration information section are automatically populated by the codelet as part of the check application form completeness task.



CreditHistoryReport

FinancialOfficerIdentifier

ID: FOA

ApplicantCreditInformation

LoanType: Personal

Amount: 2000

Duration: 1

InterestRate: 19

OverdueCreditAccounts

CreditType:

DefaultAmount:

Duration:

InterestRate:

CurrentCreditCardInformation

Provider: NAB

StartDate: 09/02/2015

EndDate: 09/02/2017

InterestRate: 19

PublicRecordInformation

CourtJudgementInformation:

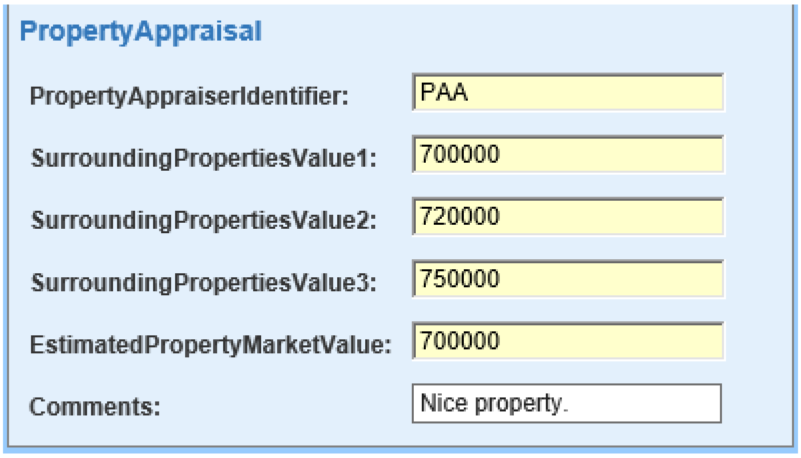
BankruptcyInformation:

CreditAssessment

Result: AA

Figure 4.3. Credit history report

Logout and log back in as the property appraiser. Click on started in the menu, select the appraise property task listed under work items and click the view/edit button to bring up the property appraisal. Fill in sample data as shown in figure 4.4 and click the complete button.

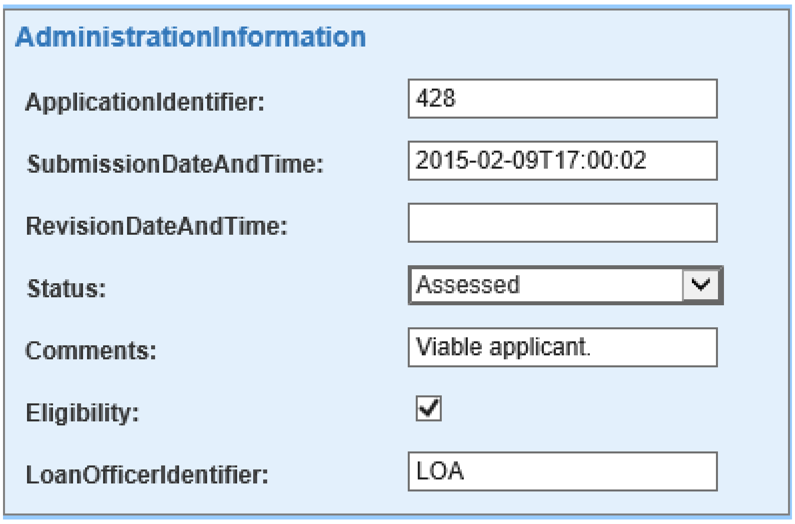


PropertyAppraisal

PropertyAppraiserIdentifier:	PAA
SurroundingPropertiesValue1:	700000
SurroundingPropertiesValue2:	720000
SurroundingPropertiesValue3:	750000
EstimatedPropertyMarketValue:	700000
Comments:	Nice property.

Figure 4.4. Property appraisal

Depending on which loan officer were randomly allocated with the assess eligibility task, logout and log back in as either Loan officer A or Loan officer B. Click on started in the menu, select the assess eligibility task listed under work items and click the view/edit button to bring up the loan application, property appraisal and risk weight. The risk weight is determined by the assess loan risk automatic task. Fill in sample data as shown in figure 4.5 and click the complete button.

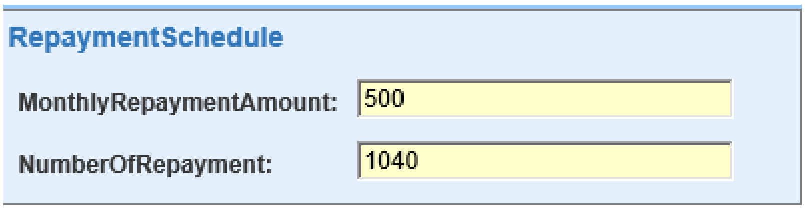


AdministrationInformation

ApplicationIdentifier:	428
SubmissionDateAndTime:	2015-02-09T17:00:02
RevisionDateAndTime:	
Status:	Assessed
Comments:	Viable applicant.
Eligibility:	☑
LoanOfficerIdentifier:	LOA

Figure 4.5. Administration information

Next, refresh the work items list, notice prepare and send acceptance pack is now appear under started work items for same loan officer. That's because in the resource setup, it has been defined that the same loan officer completed assess eligibility task will prepare and send acceptance pack. Select the task and click the view/edit button to fill out the repayment schedule as shown in figure 4.6.



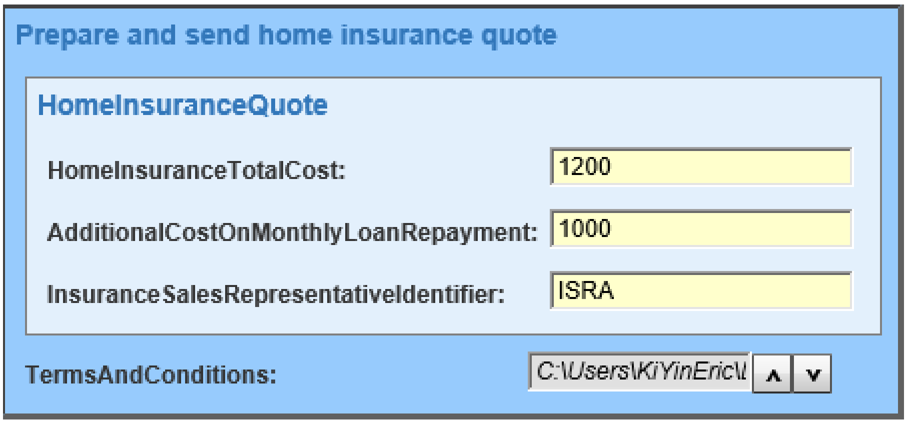
RepaymentSchedule

MonthlyRepaymentAmount: 500

NumberOfRepayment: 1040

Figure 4.6. Repayment schedule

Due to the applicant had requested for insurance quote in the loan application, logout and log back in as insurance sales rep. Click on started in the menu, select the prepare and send home insurance quote task listed under work items and click the view/edit button to bring up the home insurance quote. Fill in sample data as shown in figure 4.7 and click the complete button. The up arrow next to the terms and conditions field is to enable documents to be uploaded, click on the arrow and follow the prompt to upload a document.



Prepare and send home insurance quote

HomeInsuranceQuote

HomeInsuranceTotalCost: 1200

AdditionalCostOnMonthlyLoanRepayment: 1000

InsuranceSalesRepresentativeIdentifier: ISRA

TermsAndConditions: C:\Users\KiYinEric\

Figure 4.7. Home insurance quote

The same loan officer who completed the prepare and send acceptance pack task will verify repayment agreement. Select the task and click the view/edit button, check both conditions and repayment agreed checkboxes, click the complete button.

Finally, the take final decision task will be carried out by a different loan officer who completed the verify repayment agreement task. Logout and log back in as loan officer A or B depending on who has the take final decision task. Click on started in the menu, select the take final decision task listed under work items and click the view/edit button to bring up the application, change status to approved and click the complete button.

The above process execution demo shows one possibility of how the process can be executed, please feel free to try out different scenarios with different test data to see how alternative paths of the process can be executed at different decision points.

5.0. Further reading

Various subject areas mentioned in the lab exercise are a topic of its own, which cannot be covered entirely or in great details. In addition to the references list of the lab exercise, the below additional references are suggested further reading into such subject areas.

Aalst, W. v., & Hee, K. v. (2002). *Workflow management: models methods, and systems.* MIT press.

The YAWL foundation. (2010). YAWL - technical manual: verison 2.1. Retrieved February 19, 2015, from http://yawlfoundation.org/manuals/YAWLTechnicalManual2.1.pdf

The YAWL user group. (2015). *YAWL user group*. Retrieved from http://yaug.org

Workflow Patterns Initiative. (2015, February 19). *Workflow patterns*. Retrieved from http://www.workflowpatterns.com

References

Hofstede, A. H., Aalst, W. M., Adams, M., & Russell, N. (2010). *Modern business process automation: YAWL and its support enviornment.* Berlin: Springer.

The YAWL foundation. (2014, September). YAWL - user manual: version 3.0. Retrieved February 19, 2015, from http://yawlfoundation.org/manuals/YAWLUserManual3.0.pdf

Appendix A: XML schema data type definition

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="ApplicantInformation">
    <xs:sequence>
      <xs:element name="Name" type="xs:string" minOccurs="0" />
      <xs:element name="Surname" type="xs:string" minOccurs="0" />
      <xs:element name="Email" type="xs:string" minOccurs="0" />
      <xs:element name="HomePhone" type="xs:int" minOccurs="0" />
      <xs:element name="CellPhone" type="xs:int" minOccurs="0" />
      <xs:element name="CurrentAddress" type="xs:string" minOccurs="0" />
      <xs:element name="PreviousAddress" type="xs:string" minOccurs="0" />
      <xs:element name="CurrentEmployer" type="xs:string" minOccurs="0" />
      <xs:element name="MonthlyNetRevenue" type="xs:double" minOccurs="0" />
      <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: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="PurchasingPrice" 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="NumberOfYears" type="xs:double" minOccurs="0" />
    <xs:element name="StartDate" type="xs:date" minOccurs="0" />
    <xs:element name="InterestRate" type="xs:double" minOccurs="0" />
    <xs:element name="InterestType" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Variable" />
          <xs:enumeration value="Fixed" />
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="InsuranceQuoteRequired">
  <xs:sequence>
    <xs:element name="Required" type="xs:boolean" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AdministrationInformation">
  <xs:sequence>
    <xs:element name="ApplicationIdentifier" type="xs:string" minOccurs="0" />
    <xs:element name="SubmissionDateAndTime" type="xs:dateTime" minOccurs="0" />
    <xs:element name="RevisionDateAndTime" type="xs:dateTime" minOccurs="0" />
    <xs:element name="Status" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Complete" />
          <xs:enumeration value="Incomplete" />
          <xs:enumeration value="Assessed" />
          <xs:enumeration value="Rejected" />
          <xs:enumeration value="Canceled" />
          <xs:enumeration value="Approved" />
        </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:element name="LoanOfficerIdentifier" type="xs:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="FinancialOfficerIdentifier">
  <xs:sequence>
    <xs:element name="ID" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ApplicantCreditInformation">
  <xs:sequence>
    <xs:element name="LoanType" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
```

```
          <xs:enumeration value="Household" />
          <xs:enumeration value="Personal" />
          <xs:enumeration value="Domestic" />
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="Amount" type="xs:double" minOccurs="0" />
    <xs:element name="Duration" type="xs:double" minOccurs="0" />
    <xs:element name="InterestRate" type="xs:double" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="OverdueCreditAccounts">
  <xs:sequence>
    <xs:element name="CreditType" type="xs:string" minOccurs="0" />
    <xs:element name="DefaultAmount" type="xs:double" minOccurs="0" />
    <xs:element name="Duration" type="xs:double" minOccurs="0" />
    <xs:element name="InterestRate" type="xs:double" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CurrentCreditCardInformation">
```

```
  <xs:sequence>
    <xs:element name="Provider" type="xs:string" minOccurs="0" />
    <xs:element name="StartDate" type="xs:date" minOccurs="0" />
    <xs:element name="EndDate" type="xs:date" minOccurs="0" />
    <xs:element name="InterestRate" type="xs:double" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="PublicRecordInformation">
  <xs:sequence>
    <xs:element name="CourtJudgementInformation" type="xs:string" minOccurs="0" />
    <xs:element name="BankruptcyInformation" type="xs:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CreditAssessment">
  <xs:sequence>
    <xs:element name="Result">
      <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="LoanApplication">
  <xs:sequence>
    <xs:element name="ApplicantInformation" type="ApplicantInformation" />
    <xs:element name="PropertyInformation" type="PropertyInformation" />
    <xs:element name="LoanInformation" type="LoanInformation" />
    <xs:element name="InsuranceQuoteRequired" type="InsuranceQuoteRequired" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CreditHistoryReport">
  <xs:sequence>
    <xs:element name="FinancialOfficerIdentifier" type="FinancialOfficerIdentifier" />
    <xs:element name="ApplicantCreditInformation" type="ApplicantCreditInformation" maxOccurs="unbounded" />
    <xs:element name="OverdueCreditAccounts" type="OverdueCreditAccounts" maxOccurs="unbounded" />
    <xs:element name="CurrentCreditCardInformation" type="CurrentCreditCardInformation" maxOccurs="unbounded" />
    <xs:element name="PublicRecordInformation" type="PublicRecordInformation" maxOccurs="unbounded" />
    <xs:element name="CreditAssessment" type="CreditAssessment" />
  </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="HomeInsuranceQuote">
   <xs:sequence>
     <xs:element name="HomeInsuranceTotalCost" type="xs:double" />
     <xs:element name="AdditionalCostOnMonthlyLoanRepayment" type="xs:double" />
     <xs:element name="InsuranceSalesRepresentativeIdentifier" type="xs:string" />
   </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Appendix B: Check application form completeness codelet

```
package org.yawlfoundation.yawl.resourcing.codelets;


import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import org.jdom2.Element;
import org.yawlfoundation.yawl.engine.interfce.WorkItemRecord;
import org.yawlfoundation.yawl.elements.data.YParameter;
import org.yawlfoundation.yawl.resourcing.codelets.AbstractCodelet;
import org.yawlfoundation.yawl.resourcing.codelets.CodeletExecutionException;


public class CheckApplicationFormCompleteness extends AbstractCodelet {


        public CheckApplicationFormCompleteness(){
                super();
                setDescription("This codelet checks for all the required informationa and its format of a loan application.  Required
parameters:<br> " +
                 "Input: Loan application<br>" +
                 "Output: Administration information");
```

```
}

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

        // Set the inputs passed in the base class
        setInputs(inData, inParams, outParams);

        //      Get the data values required
        Element Name = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("Name");
        Element Surname = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("Surname");
        Element Email = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("Email");
        Element HomePhone = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("HomePhone");
        Element CellPhone = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("CellPhone");
        Element CurrentAddress = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("CurrentAddress");
        Element CurrentEmployer = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("CurrentEmployer");
        Element MonthlyNetRevenue = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("MonthlyNetRevenue");
        Element BankName = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("BankName");
        Element TypeOfAccount= inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("TypeOfAccount");
        Element AccountNumber = inData.getChild("LoanApplication").getChild("ApplicantInformation").getChild("AccountNumber");
        Element TypeOfProperty = inData.getChild("LoanApplication").getChild("PropertyInformation").getChild("TypeOfProperty");
        Element PropertyAddress = inData.getChild("LoanApplication").getChild("PropertyInformation").getChild("PropertyAddress");
        Element PurchasingPrice = inData.getChild("LoanApplication").getChild("PropertyInformation").getChild("PurchasingPrice");
```

```
    Element LoanAmount = inData.getChild("LoanApplication").getChild("LoanInformation").getChild("LoanAmount");

    Element NumberOfYears = inData.getChild("LoanApplication").getChild("LoanInformation").getChild("NumberOfYears");

    Element StartDate = inData.getChild("LoanApplication").getChild("LoanInformation").getChild("StartDate");

    Element InterestRate = inData.getChild("LoanApplication").getChild("LoanInformation").getChild("InterestRate");

    Element InterestType = inData.getChild("LoanApplication").getChild("LoanInformation").getChild("InterestType");

    String Comments = "";


    // Check the loan application form fields and add comments accordingly

    if(Name==null) {

            Comments = "Name is required.  ";

    }


    if(Surname==null) {

            Comments = Comments + "Surname is required.  ";

    }


    if(Email==null) {

     Comments = Comments + "Email is required.  ";

}


if(HomePhone==null) {

    Comments = Comments + "HomePhone is required.  ";
```

```
}
else if (HomePhone.getText().length()!=10) {
    Comments = Comments + "Home phone must be 10 digits.  ";
}

if(CellPhone==null) {
    Comments = Comments + "CellPhone is required.  ";
}
else if (CellPhone.getText().length()!=10) {
    Comments = Comments + "Cell phone must be 10 digits.  ";
}

if(CurrentAddress==null) {
    Comments = Comments + "CurrentAddress is required.  ";
}

if(CurrentEmployer==null) {
    Comments = Comments + "CurrentEmployer is required.  ";
}

if(MonthlyNetRevenue==null) {
    Comments = Comments + "MonthlyNetRevenue is required.  ";
```

```
}

if(BankName==null) {
    Comments = Comments + "BankName is required.  ";
}

if(TypeOfAccount==null) {
    Comments = Comments + "TypeOfAccount is required.  ";
}

if(AccountNumber==null) {
    Comments = Comments + "AccountNumber is required.";
}

if(TypeOfProperty==null) {
    Comments = Comments + "TypeOfProperty is required.";
}

if(PropertyAddress==null) {
    Comments = Comments + "PropertyAddress is required.";
}
```

```
if(PurchasingPrice==null) {

   Comments = Comments + "PurchasingPrice is required.";

}


if(LoanAmount==null) {

   Comments = Comments + "LoanAmount is required.";

}


if(NumberOfYears==null) {

   Comments = Comments + "NumberOfYears is required.";

}


if(StartDate==null) {

   Comments = Comments + "StartDate is required.";

}


if(InterestRate==null) {

   Comments = Comments + "InterestRate is required.";

}


if(InterestType==null) {

   Comments = Comments + "InterestType is required.";
```

```
            }



            // Set the output data
               WorkItemRecord wir = this.getWorkItem();
               String caseID = wir.getRootCaseID();
               DateFormat DateFormat = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
               Date Date = new Date();
               String DateAndTime = DateFormat.format(Date);
            String AdministrationInformation = "<ApplicationIdentifier>"+caseID+"</ApplicationIdentifier>
<SubmissionDateAndTime>"+DateAndTime+"</SubmissionDateAndTime>";


            if(Comments!=""){
               setParameterValue("AdministrationInformation", AdministrationInformation+" <Status>Incomplete</Status>
<Comments>"+Comments+"</Comments>");
            }
            else {
               setParameterValue("AdministrationInformation", AdministrationInformation+" <Status>Complete</Status>");
            }


               // Return the element created in the base class and containing the result
                return getOutputData();
            }
```

```
       public List<YParameter> getRequiredParams() {
               List<YParameter> params = new ArrayList<YParameter>();

        YParameter param = new YParameter(null, YParameter._INPUT_PARAM_TYPE);
        param.setDataTypeAndName("LoanApplication", "LoanApplication", XSD_NAMESPACE);
        params.add(param);

               param = new YParameter(null, YParameter._OUTPUT_PARAM_TYPE);
        param.setDataTypeAndName("AdministrationInformation", AdministrationInformation", XSD_NAMESPACE);
        params.add(param);

        return params;
    }
}
```