Automation for Web Microservices

Treesakul Tongsaree
Phat Thaveepholcharoen
Pakpoom Kunalittipol

Bachelor of Engineering in Software Engineering
International College
King Mongkut's Institute of Technology Ladkrabang
Academic Year 2018
KMITL-2019-IC-B-003-007

Thesis – Academic Year 2018

Bachelor of Engineering in Software Engineering
International College
King Mongkut's Institute of Technology Ladkrabang

Title: Automation for Web Microservices

Authors

1. Ms. Treesakul Tongsaree Student ID: 58090016
2. Mr. Phat Thaveepholcharoen Student ID: 58090025
3. Mr. Pakpoom Kunalittipol Student ID: 58090027

Approved for submission

V. Hirankitti

.....

(Assistant Professor Dr. Visit Hirankitti)
Advisor

Date 26/6/2019

Acknowledgements

We would like to express our very great gratitude to our advisor, Asst.Prof.Dr. Visit Hirankitti. We really appreciated for his valuable and constructive suggestions during the planning and development of this thesis. His willingness to give his time so generously has been very much appreciated.

Also, we would like to express our deep gratitude to Asst.Prof.Dr. Chaiwat Nuthong, Asst.Prof.Dr. Isara Anantavrasilp, Assoc.Prof.Dr. Veera Boonjing, Dr. Ukrit Watchareeruetai, Dr. Montri Phothisonothai and Dr. Natthapong Jungteerapanich, for their enthusiastic encouragement and useful critiques of this thesis.

Furthermore, we would like to extend our thanks to International College staff and our colleagues including friends and classmates for all their support and help.

Last by not least, we would like to express our gratitude to our parents and all other members of the family for their support and encouragement throughout our study and develop this thesis.

Abstract

The aim of this project is to create a developing environment, so-called "AutoWeb", for constructing and deploying a web application. Such a web application is created easily from an executable workflow which integrates several collaborative microservices to serve a business procedure.

AutoWeb consists of three components; Workflow Composer, Microservice Binder, and Workflow Execution Engine. Workflow Composer is an authorizing tool for composing as well as validating a graphical BPMN (Business Process Model and Notation) workflow diagram. A microservice available on AutoWeb framework together with its necessary input data will be selected and bound into a BPMN task in a workflow diagram using our Microservice Binder mechanism. Once the workflow has been created and bound correctly, a corresponding workflow object represented in JSON will be generated and that JSON workflow object will finally be executed by Workflow Execution Engine and the outcome of such execution can be considered as the result of running a web application.

AutoWeb is developed using Python web framework Django and JavaScript. On the client-side, the Workflow Composer and the Microservice Binder are developed using React; which is a JavaScript web framework. On the server-side, the Workflow Execution Engine employs Finite State Machine (FSM) as an execution model, which is developed in Python.

AutoWeb is our novel contribution for automating web microservices as the parts of an executable workflow. It improves collaboration and flexibility in the way people work together, and also increases business productivity. To provide a proof of concept how AutoWeb works, we deploy AutoWeb to create a few real-world applications, such as a simple home automation based on an IoT (internet of things) technology and a job web application.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

The way businesses operate in everyday life is usually full of routine works which sometimes are inefficient and unproductive. In addition, a business process often needs collaboration between staff where timing and synchronization are great important. Business workflows are invented as a tool to manage and improve these routine works. In this project, we want to make these business workflows executable. Therefore, our aim is to create an easy-to-use development environment of a web application created from a business workflow. The resulting application improves not only business efficiency, but also increases business collaboration and productivity.

1.2 Objectives

1. To develop a development environment for creating web applications from programmable and executable workflows.

2. To develop a tool for composing a workflow according to the BPMN (Business Process Model Notation) standard.

3. To develop a tool for binding microservices to tasks of a workflow which allows web input and output form interface to be customizable for each task.

4. To develop a workflow execution engine based on a finite state machine model and implemented in Python.

5. To test workflow development environment by applying it to develop a few real-world business-oriented application

1.3 Scope of Work

The scope of this project can be listed as follows:

- To develop a workflow composer for creating workflow diagram according to the standard of BPMN 2.0.
- To develop a validator for validating BPMN diagrams against BPMN grammars.
- To develop a microservice binder for binding microservices to tasks of workflow.
- To develop an HTML-form interface customization tool for capturing input and rendering output for a microservice within a task of a workflow.
- To develop a workflow execution engine which can execute a workflow based on the finite state machine model.

1.4 Thesis Structure

This thesis consists of seven chapters which are organized as follows:

- Chapter 1 Introduction - refers to the motivation, objectives, scope of work, and thesis structure of this thesis.
- Chapter 2 Related works – is a literature survey and a comparative study of a few software products which are related to our project.
- Chapter 3 Background knowledge - explains the knowledge and technology necessary for the reader to understand the thesis.
- Chapter 4 Requirement analysis/ System Architecture and Design – presents the requirement of our system, its use case diagram, and its architecture.

- Chapter 5 Software Development - explains the concepts, tools, and techniques used for developing AutoWeb.

- Chapter 6 Results - refers to the results of AutoWeb demonstration, which consists of the user interfaces of the software and its real-world applications.

- Chapter 7 Conclusion and Future work - is about the conclusion of the thesis, future work and development.

Chapter 2
Related Works

This chapter primarily presents a comparison and evaluation between existing software related to a workflow management system; which are Microsoft Flow, Camunda BPM, Activiti, and Workflow Engine, the latter is the work done previously which is the basis of our project.

2.1 Microsoft Flow

Microsoft Flow is an online workflow service that automates events across the most common apps and services. For example, it allows users to create a flow that automatically generates a page in Quick Notes after a form has been submitted. Microsoft Flow provides more than 220 services, including Facebook, Gmail, Microsoft services, and others. Microsoft Flow also allows users to create their own form by connecting to Microsoft Forms platform [1].

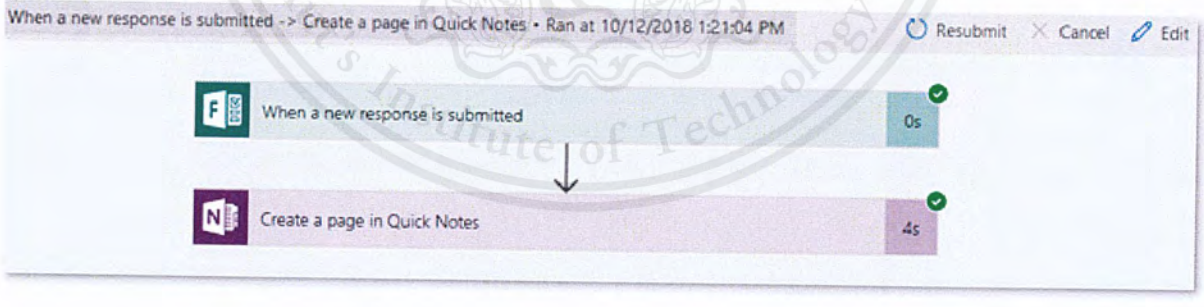


Figure 1: A screen of Microsoft Flow for composing business flows

Microsoft Flow, however, consists of two types of flows, scheduled flows and flow-chart-like business flows. A scheduled flow performs one or more tasks depended on specified time set, while a business flow performs a defined set of steps that leads to a desired outcome without time limitations. There are actions, triggers, and conditions

available for each flow. An action is given at the first step, which can be followed by a trigger, an action, or a condition and so on until the end of the flow.

To compare with our project, there are some similarities, such as actions and tasks, scheduled flows, and time events. However, there are some major differences in detail. First, since Microsoft Flow only begins with one action, it is *not well-grounded for parallel executions*. Secondly, Microsoft Flow *does not support UI customization*, while this project provides the ability to custom user interface for every single task. Finally, though Microsoft business flow *supports a limited number of business models*, BPMN 2.0 has appeared to be standard language that is more powerful by supporting more business models.

2.2 Camunda BPM

Camunda BPM is a Java-based open source utilized basically to mechanize Business Process Model and Notation (BPMN) 2.0 procedures. Camunda's working procedure initially handles graphs and DMN choice tables. Also, it transforms a client's model into an executable application. Camunda additionally contains a ready-to-use web application, called *Tasklist*, which allows end users to work on the tasks assigned to them. It also supports HTML form creation [2].

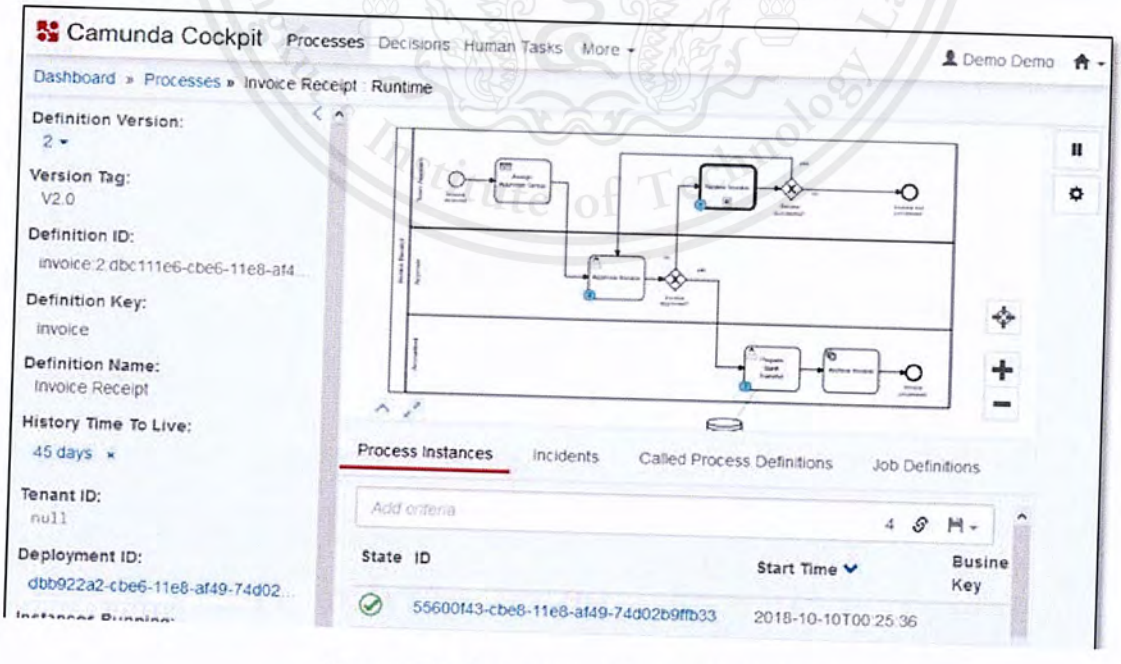


Figure 2: A screen of Camunda BPM

For its comparison with this project, there are a number of differences. First, Camuda *does not provide the tool for designing forms using drag-and-drop feature for user interfaces* as they prefer that users build their own forms in their preferred language, while this project supports full user interface customization. Secondly, Camunda allows connection with external services, where *programing knowledge in Java is required.* In contrast, connecting tasks to third parties in our project only requires a service's address to connect to a task. Moreover, *Camunda modeler is only available on a desktop application.*

2.3 Activiti

Activiti is a Java-based open-source BPMN engine supporting real-world process automation need.

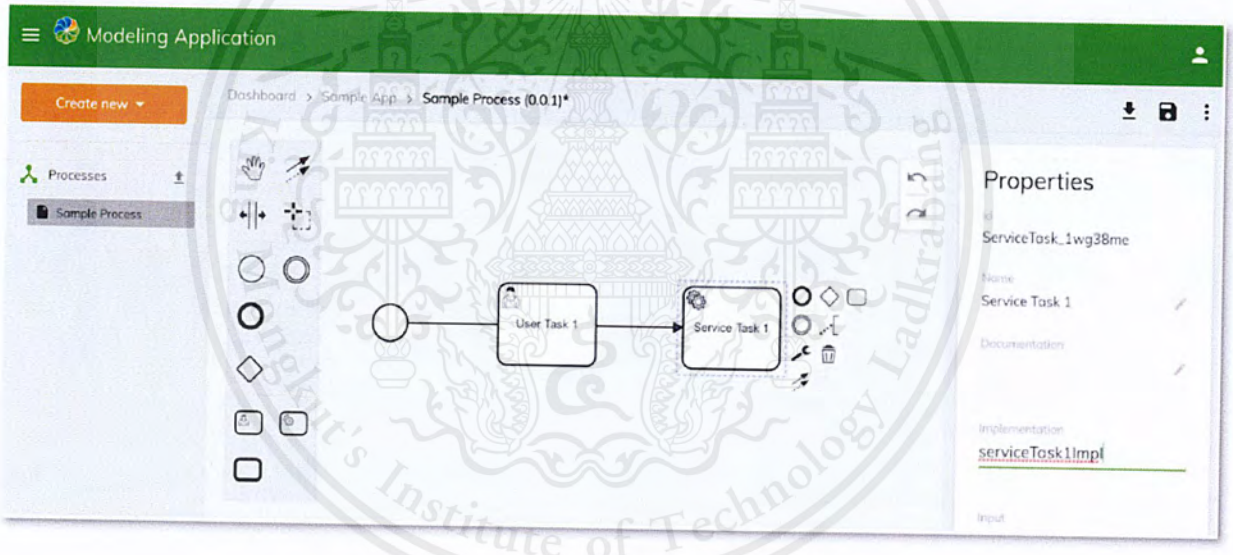


Figure 3: A screen of Activiti for composing BPMN

Though both in our project and Activiti a business process is modeled in BPMN 2.0, there are a few distinctions. Firstly, *Activiti provides a form builder, however, it is only available in enterprise edition*, while our project supports any types of users. Secondly, Activiti *requires programming knowledge* to bind tasks in Java to make a REST API call to a third-party web service.

2.4 Workflow Engine

This is the previous work done in our research group which serves as the basis for our work. The core of such project is an automated business process engine which manages and monitors the state of activities in a workflow. The workflow engine allocates tasks to different executors while communicating data among participants [3]. With this engine, authorized user is allowed to:

- create their own workflow using BPMN 2.0 to design the workflow,
- download the XML script files that has been generated from the created workflow,
- download the generated forms from created flows,
- save and edit an unfinished flow, in order to continue drawing it later on,
- browse user's XML local files, and
- collaborate in real-time with other users.

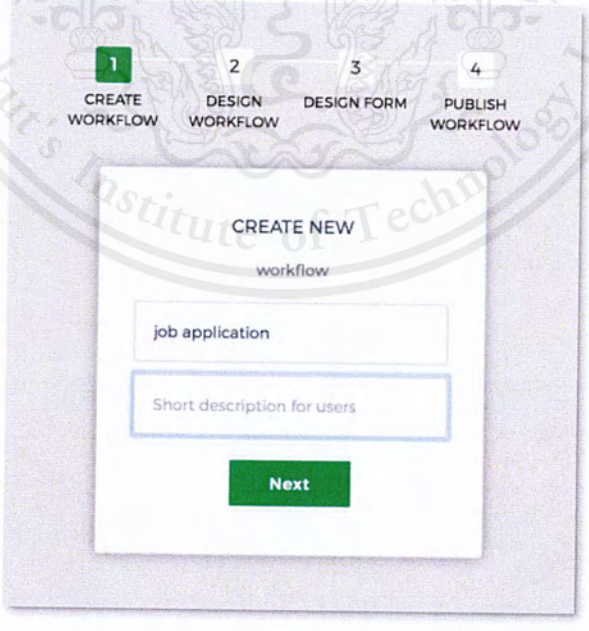


Figure 4: A screen of Workflow Engine for creating a new workflow

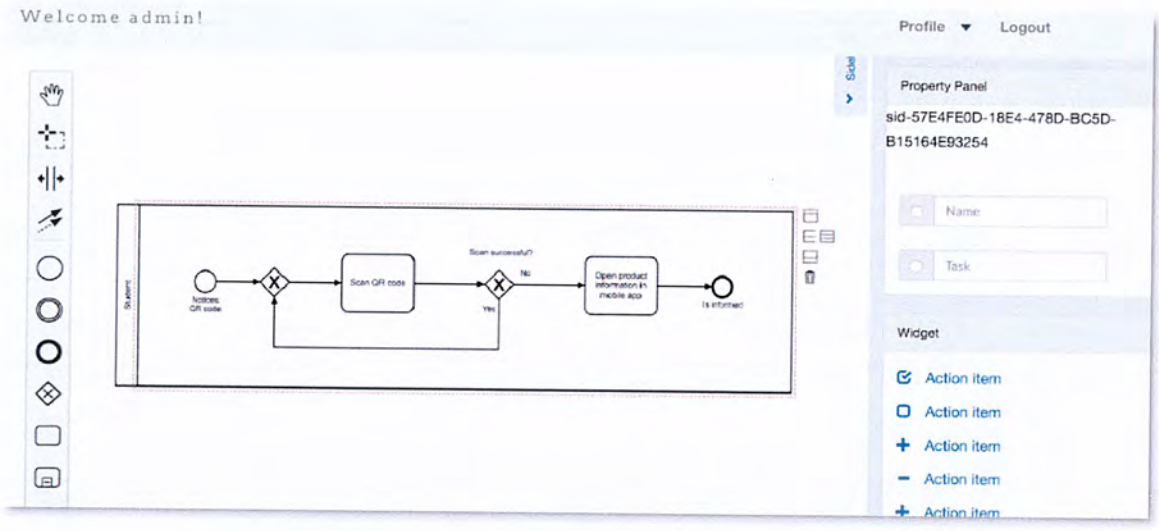


Figure 5: A screen of Workflow Engine - BPMN drawing tool

Though Workflow Engine is conceptually comparable to our project, there are a few major differences. Firstly, Workflow Engine supports only basic BPMN 2.0 elements, *it includes events, such as time-events*. As a result, it limits by a capacity of a workflow. Secondly, forms are provided in Workflow Engine, however, *drag-and-drop UI customization is not available*. Lastly, *third-party services are not supported by Workflow Engine*, whilst AutoWeb is designed to work with all standard web services.

2.5 Comparison with other related works

To compare between a flowchart and a business workflow, obviously a flowchart can explain a process but in a simpler way that a business workflow can; and Microsoft flow makes use of a flow-chart-like business workflow.

Firstly, BPMN consists of more elements than those in a high-level flowchart, more complex business process patterns, such as those with exceptions, decisions and events. With these models among other things, detailed requirements for IT development can be easily created. Secondly, with enterprise architecture, other organizational aspects are considered beyond a simple order of activities, such as events, collaborators, external systems, etc. This is a comparison table which compare our project and other related work.

Table 1: Comparison between related work in workflow automation

Feature / Purpose	**Microsoft Flow**	**Camunda BPM**	**Activiti**	**Workflow Engine**	**Our Project**
Using BPMN2.0		Yes	Yes	Yes	Yes
Creating Web-based application	Yes		Yes	Yes	Yes
Deploying Fully customizable forms	Yes**	No	Yes*	Yes	Yes
Allowing developers to add their own existing backend services	Yes	Yes			Yes
Requiring Java class to make REST API calls		Yes	Yes		
Free to use	Yes*			Yes	Yes
Supporting third-party services	Yes				Yes

* Some services are only available for a premium user

** Customized by using external service called Microsoft Forms

Chapter 3
Background Knowledge

In this chapter, relevant background knowledge is described in order to get a better understanding of the project, which can be divided into five parts.

3.1 Business Process Model and Notation (BPMN)

According to Object Management Group, Inc., Business Process Model and Notation (BPMN) provides a language for businesses with the ability of expressing internal business procedures in a graphical notation and this representation allows organizations the ability to communicate these procedures in a standard form. The main purpose is to ensure that people in an organization and their partners can understand clearly the business procedure they both get involved [4].

3.1.1 Core elements

A BPMN diagram consists of core elements categorized into four groups: connecting objects, flow objects, swimlanes, and artifacts and data objects. Each group has symbols which represent different action within the group.

Connecting objects

BPMN diagrams are basically graphs, which means they consist of nodes and edges. However, BPMN edges are directed, so they can be understood as flows. Graphically, a **flow** is represented with arrows linking between elements in a process. BPMN 2.0 defines three types of flows.

Table 2: Types of Flow notation

Type	Notation	Description
Sequence flow		A connector that connects between two flow objects in a process.
Message flow		A flow that represents communication between a process and an external entity, for example a process pool.
Association flow		A flow that annotates a relationship between an artifact and data object or flow objects.

Flow objects

Flow objects are the main elements of BPMN, which consists of three core elements: activities, events, and gateways.

- **Activities**

An activity represents a unit of work performed in the process. There are two types of activities, which are tasks and subprocesses.

Task is a single and atomic action that occurs in a business process, which cannot be broken down any further. Being an action, it means that the label starts with a verb following by a noun or so-called VERB-NOUN, e.g. send an email. There are eight types of tasks, however. only three are commonly used. There are also a types of loop task notations used to define a task's looping behavior. There are also a types of loop task notations used to define a task's looping behavior.

Table 3: Types of Task notation

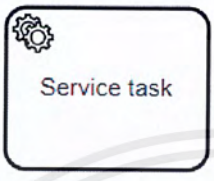
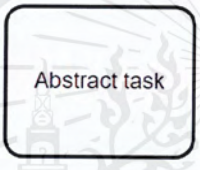
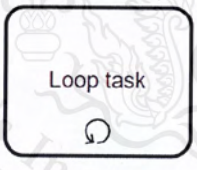
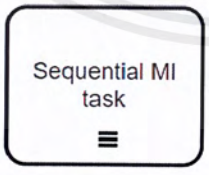
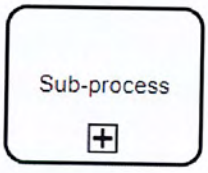
Type	Notation	Description
User task		A task performed by user.
Service task		A task that uses a web service, an automated application, or another kind of service to complete the task.
Abstract task		A task which its type is undefined.

Table 4: Types of Loop Task notation

Type	Notation	Description
Loop task		A task that occurs repeatedly.
Sequential multiple instance task		A task that occurs multiple times sequentially. The horizontal lines mean that the task must be completed before repeating the occurrence.
Parallel multiple instance task		A task that occurs multiple times parallelly. The vertical lines mean that the task can be executed without any particular order.

Subprocess is an activity containing subparts that can be interpreted as another process flow.

Table 5: Subprocess notation

Type	Notation	Description
Subprocess		A subprocess represents a set of tasks within a business process.

- **Gateways**

Gateways control process flow by splitting it into alternative paths. There are multiple gateway types. The most commonly used ones are exclusive gateways (XOR) and parallel gateways (AND).

Table 6: Types of Gateway notation

Type	Notation	Description
Exclusive gateway		A gateway with only one of its outgoing sequence flows is enabled based on its specified condition.
Parallel gateway		A gateway that all its outgoing sequence flows are enabled parallelly and unconditionally.

- **Events**

Events describe something that happens during the course of a process. There are three types of events: start events, intermediate events, and end events. However, these types can also be categorized as either a catching event or a throwing event. Catching events are events that react to a trigger, while throwing events are assumed to be events that trigger themselves.

Table 7: Types of Event notation

Type	Notation	Description	
Start event	 Start event	It is a catching even that catches information. It indicates where and how a process starts. There are several types of start events, however only a few are commonly used.	
		 None	A none start event signifies manual start by a task performer. A subprocess must have a none start event.
		 Message	A message start event means that the process is triggered when receiving a message.
		 Timer	A timer start event signifies a scheduled process, normally labelled to indicate the schedule, such as a countdown or a defined schedule.
Intermediate event	 Intermediate event	It triggers changes that affect the process execution, in other words, during the execution.	
		 message catching event	A message catching event signifies that the process is triggered when a message is received during the process.
		 message throwing event	Similarly, a message throwing event indicates that a message is sent instead of being received.

		 intermediate timer event	An intermediate timer event is triggered based on the defined schedule or countdown.
End event	 End event	It is a throwing event that indicates the end of a path in a process. There are different types of end events, yet only a few are widely used.	
		 None	A none end event signifies that when it is reached, no result will be thrown.
		 Message	A message end event indicates that a message will be sent when the event is reached.
		 Terminate	A terminate end event ends the process or subprocess, even if the other parallel paths are still running.

Swimlanes

A pool is simply a container carrying flows and flow nodes. Pools are split into 2 types. Firstly, a white-box pool is a pool that the elements are visible or known. Different from a white-box pool, a black-box pool is an empty pool: the elements and connectors inside are not known. However, black-box pool is best used when the pool does not reference a process but only an external process participant.

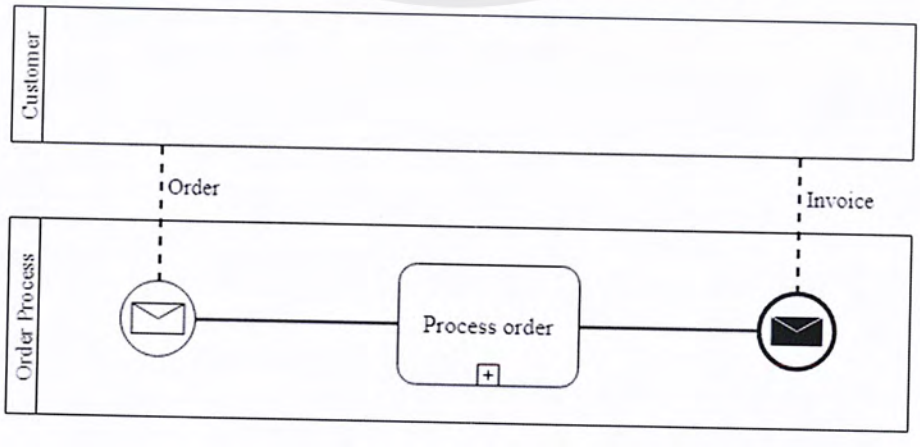


Figure 6: Example of Swimlanes

Clearly, the top pool named Customer is a black-box pool that its flow nodes and connectors in are not visible. The bottom pool is a white-box pool since its flow elements are visible.

A lane is basically a role in a pool. Usually, a pool refers to an organization, which has different roles or responsible departments. Those roles are called lanes. As an example of a bank BPMN diagram. The bank has 2 departments as lanes involved in the process, which are a risk analysis department and a sales department. A lane can also be divided into multiple nested lanes. As the example, a sales department is divided into 2 lanes: a sales manager and a sales representative.

Figure 7: Example of lane

Artifacts and data objects

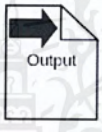
Artifacts are BPMN's elements that describe the process. There are two types of artifacts: text annotation and group.

Table 8: Types of Artifacts notation

Type	Notation	Description
Text annotation	Text Annotation	A text annotation is used to describe a flow node on the diagram. It contains a content called text element, while the association is a connector used to link the text to a flow node.
Group		A group is used to categorize flow elements to show how they are related for readability.

Data objects show inputs and outputs of activities. They do not affect the process flow. There are different kinds of data objects.

Table 9: Types of Data Object notation

Type	Notation	Description
Data object	 Data object	It is a piece of temporary data stored inside the process while it is running.
Data input	 Input	A data input represents data requirement of a certain task waiting for the data in order to proceed.
Data output	 Output	A data output is a result produced when a process generates data.
Data collection		A data collection is an array of data objects, such as a survey.
Data store	 Data store	A data store represents persistent data, such as data stored in a database.

3.1.2 Example of BPMN 2.0 in practice

- **Online Purchase**

This is a simple example of an online purchase described by a BPMN diagram. This process contains 5 different elements, which are pool, lanes, process, tasks, gateways, and events.

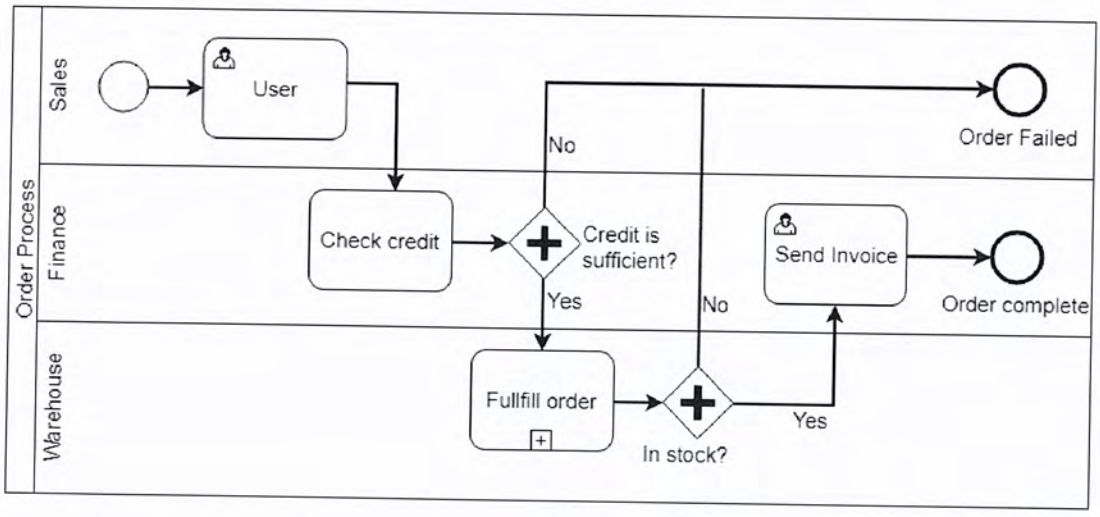


Figure 8: An example of BPMN diagram for an online purchases process

Firstly, the whole rectangle is called a pool, basically contains 3 swimming lanes. The lanes are different roles in the pool. The thin-lined circle is the starting point, called a start event. It specifies where the process starts. The sales department then receives an order as it states in a rounded rectangle with a user symbol on the top left corner or so-called a user activity.

Later on, the finance department system performs the 'check credit' service activity. The next element is a gateway, used to control how sequence flows interact as they join and split within a process. The gateway in this diagram is called a XOR gateway, which gives the direction of the flow based on the decision. In this case, the decision is whether the credit is sufficient. If not, the flow continues to the end event, which is a thick circle specifying that the process is complete. If the credit is sufficient, the warehouse then performs the 'fulfill order' subprocess; a subprocess is the division of an activity as well as a task. After that, the warehouse checks if the product(s) in the order are in stock. If so, the finance then sends out the invoice and the flow goes to the end state; which means the order is complete. Otherwise, the flow goes to the other end state named 'order failed'.

- **Order Placement**

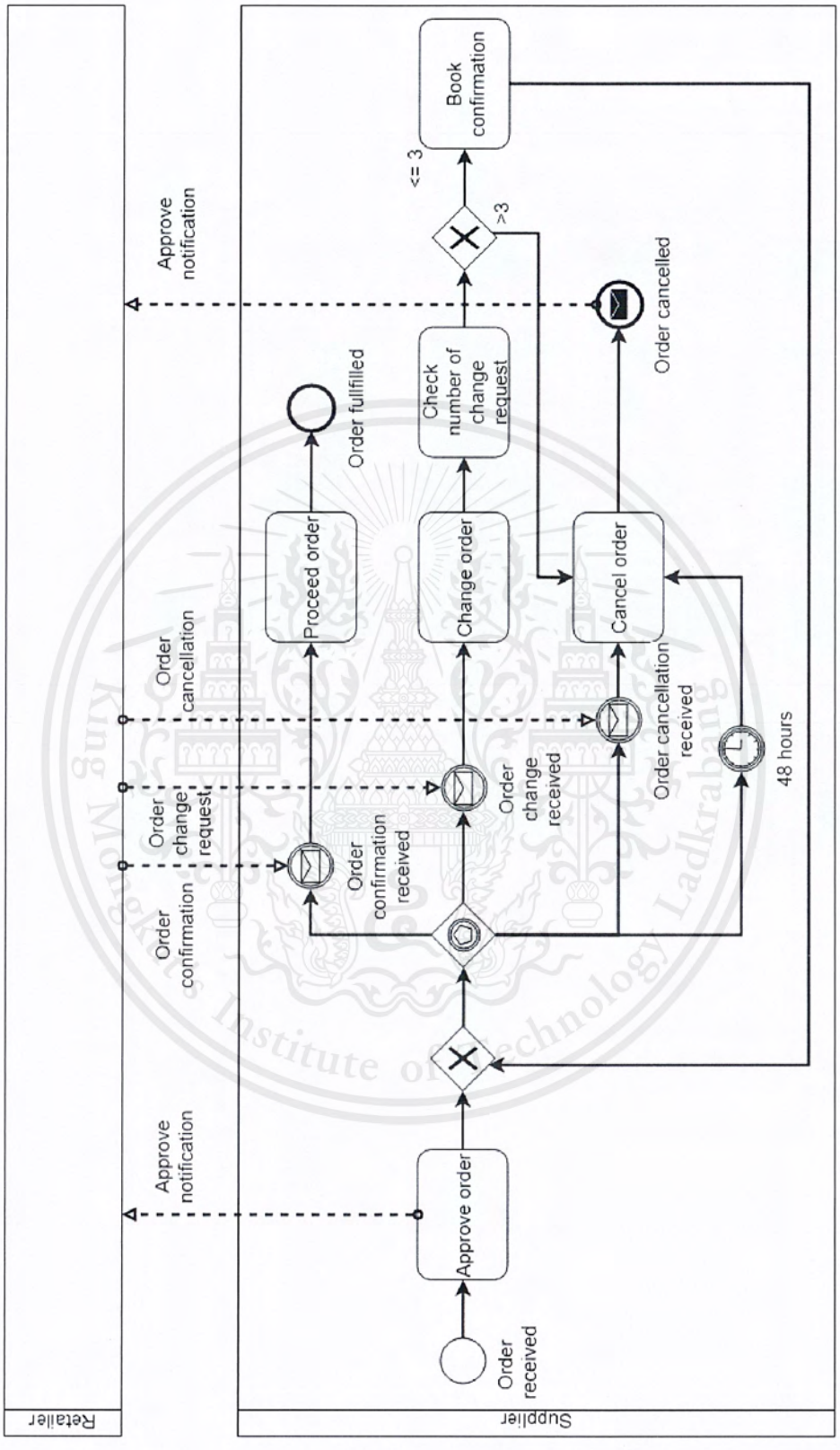


Figure 9: Example BPMN (Order placement)

This is a realistic example of an order placement process. This process contains two pools, a retailer and a supplier. Noticeably, the retailer pool is a black-box pool used to send and receive messages in this case, while the supplier process is wrapped in a white-box pool as the current process is focused on the supplier side.

There are several types of events used in this example, including an event-based gateway: which directs the flow to the first triggered event out of all outgoing flows. In this diagram, after the event-based gateway, a few intermediate message events are used in order to catch messages sent from the retailer. However, there is an intermediate timer event in the diagram as well. After the order has been approved, it starts counting down and triggers after the time reaches 48 hours as labeled. For events in the pool to connect to an external pool, it uses an association, for example, the approve notification is sent to the retailer from the supplier. To begin, it starts from the start event and eventually ends up at one of the end events. However, there are two end events in this case, order cancelled, and order fulfilled. The order fulfilled event can only be reached when the order confirmation is received, and the supplier proceeds the order. The order-cancelled event is reached when the order left untouched for 48 hours, the retailer sends a cancellation request, or the number of change requests are done over 3 times as shown above.

- **Online shopping**

This example of an online shopping process in Figure 10 has added the involvement of customers. For the pool of an online store, it is divided into 3 different roles; warehouse, finance, and sales management. The warehouse is responsible for checking the availability of the products in the stock. The finance department takes charge of checking the credit sufficiency. Sales management manages the order received from the customer. These white-box pools focus on the communication between pools as shown below.

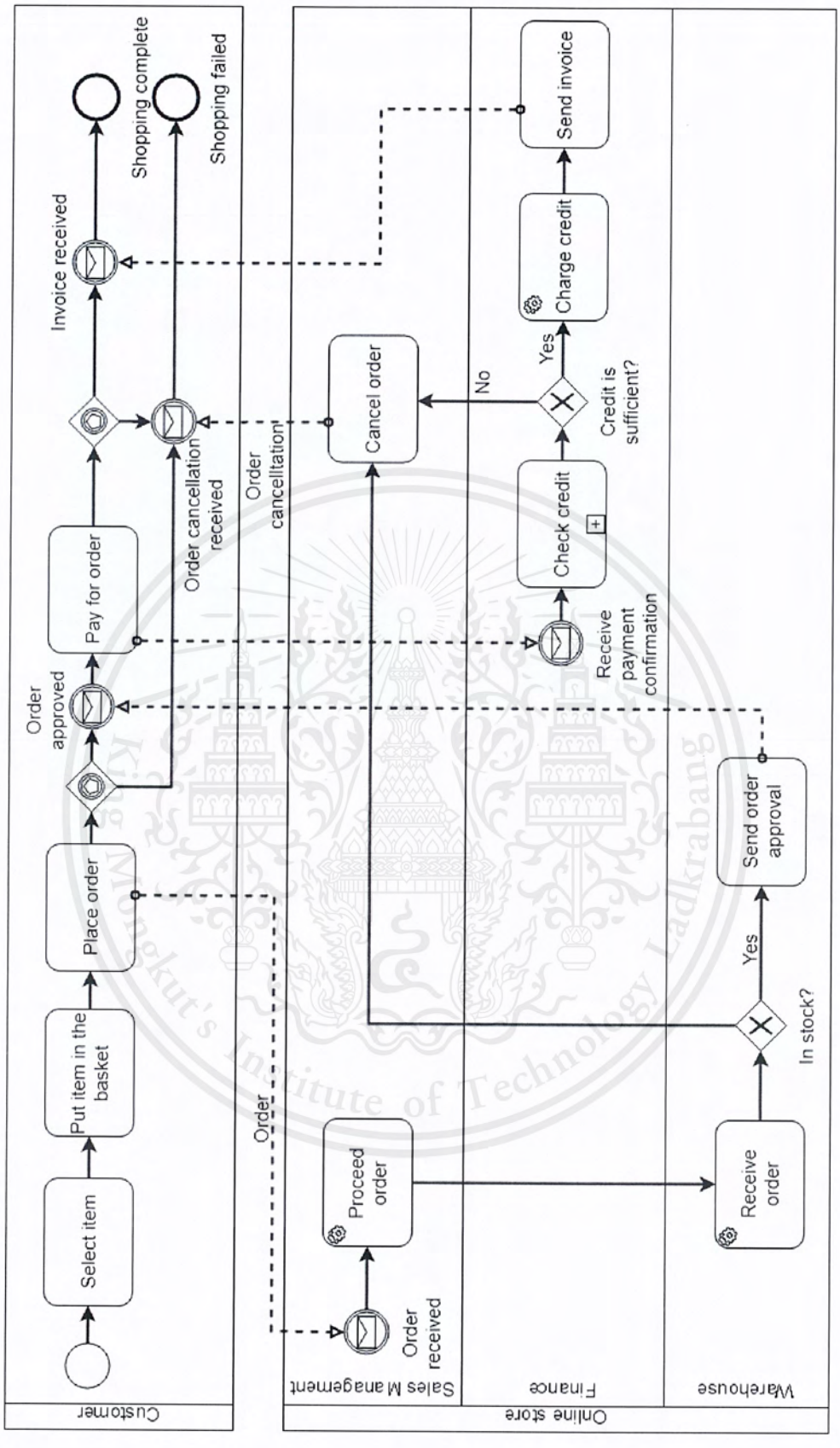


Figure 10: Example BPMN (Online shopping)

3.1.3 An execution model of a BPMN diagram

Generally, a BPMN diagram is created as an ordinary non-executable workflow. While a typical diagram describes the process logic in a human-readable way, for an executable BPMN diagram. However, there is a system that automates the workflow execution starting from process instantiation to completion. An executable process workflow must be machine-readable, which requires some specification of additional details in each BPMN's element; these details are:

- *Process variable* refers to BPMN task that is executed.
- *Task input interface and output interface* describe interfaces of input and output in order to pass the correct parameter for execution.
- *Task input and output data* stores a result of task execution.
- *Task user interface forms* refer to user interaction to an input execution's parameters.
- *Task performer assignment logic* describes execution instruction.
- *Conditional expressions* describe a flow condition that controls the execution path.
- *Event definitions* refers to event types, including time triggered events and message triggered events.

These details are not explicit in the diagram, however, for an executable BPMN diagram represented in XML, one can extra provides XML elements to specify the additional detail [5]. For AutoWeb, the execution of BPMN needs a business process to be in a BPMN model that requires technical detail needed for execution purpose. While running, task instances is generated for each process run. The system controls the flow and decides which task to be executed next. In addition, business workflow execution is not only sequential execution, but also parallel execution, which allows more than one task to run over a given time period. In other words, parallel execution

allows many tasks to be executed in an overlapping fashion. An asynchronous operation can manage the concurrent execution; which each process operates independently of the other processes.

In business process, a business workflow may require concurrent execution. Therefore, AutoWeb supports both sequential and concurrent execution in order to be able to automate the business workflow. Moreover, AutoWeb employs a finite state machine as an execution model.

3.2 Finite State Machine (FSM)

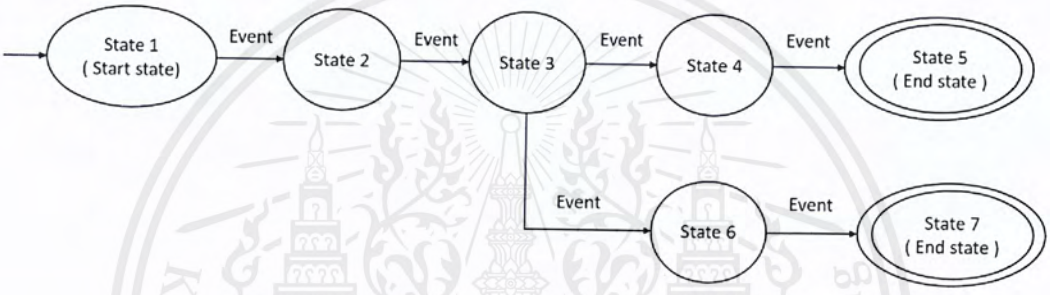


Figure 11: Finite state machine diagram

A finite state machine or a finite state automaton, is an execution model that operates on event or input to change the state and causes an action or output. The finite state machine can transition from one state to another state in response to some input; this is called a state transition.

The finite state machine can be defined using a set of input/output events, a set of states, a description of the initial state and a state transition function which given a state and an input event, will return the next state [6]. Finite state machine is described by a five-element tuple (Q, ∑, q0, F, δ).

- Q is a finite set of states
- ∑ is a possible set of events that cause state changes.
- q0 is an initial state.

- F is a set of accepting or final state.

- δ is a set of state transition function. From current state the finite state machine uses the transition function to determine the next state.

Finite state machine's behavior is analogous to business workflow since both finite state machine and business workflow consist of a finite set of states and state transitions. Hence, the model of a finite state machine can be used to implement executable workflow as a development tool for approaching and solving problems.

However, in order to apply finite state machine to develop executable workflow, it is necessary to apply the finite state machine's formal definitions. In the context of finite state machine, the finite state machine's formal definitions can be applied as follows:

- Q is a set of flow object that can be executed.

- C is a current execution of flow object in a workflow.

- F is the set of end event of the workflow.

- δ is a set of state-transition function that controls the flow of workflow. State transition function is a tuple that consists of current execution of flow object in workflow (C), action and next flow object (C, action, next flow object). When workflow executes, it looks up on the state-transition function to find the next flow object that should be executed. For example, (Start, start execution, Task1) means that when the current execution is "Start" node and "start execution" action has been triggered, the next node to be executed is "Task1". Moreover, current state (C) always starts at the first task.

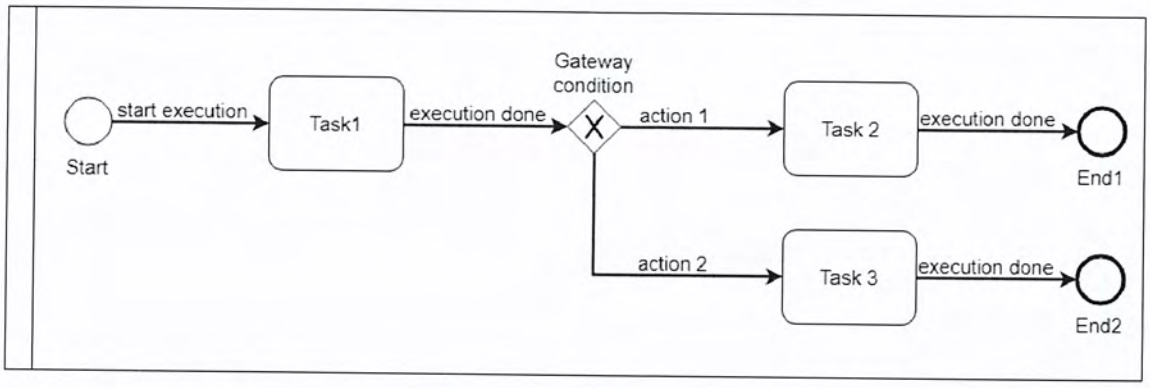


Figure 12: Example of workflow diagram that represents tasks and actions

For example, the workflow consists of tasks and transitions as shown in Figure 12. The workflow is described as the following:

- Q is {Start, Task1, Gateway condition, Task2, Task3, End1, End2}
- C is {Start}
- F is {End1, End2}
- δ is {(Start, start execution, Task1), (Task1, execution done, Gateway condition), (Gateway condition, action1, Task2), (Gateway condition, action2, Task3), (Task2, execution done, End1), (Task3, execution done, End2)}

Therefore, when the workflow is executed, it starts with the current execution point (C), then look up in the state-transition function to find the next flow object to be executed. However, transitions or flows are corresponding to user's action.

3.3 RESTful Web Service

Web service is a standardized to communicate between the client and server. Web services are web application components that can be published, found and used on the web. Web services can be searched over the network and can also be invoked. When invoked the web service will be able to provide functionality to the client which invokes that web service. Representation state transfer (REST) providing standards for web service, making systems to communicate between computers. A web service that

conform to REST, called a RESTful web service (RWS). In this project, a RESTful web service is used for data communication. RESTful web service is the architecture style that is driving modern web development. RESTful web services provide interoperability between computer systems on the internet. In fact, RESTful web services allow the requesting system to access or manipulate web resources by using predefined operations. Common operations available are GET, POST, PUT, DELETE. All these operators are used to manage data as follows:

- POST is utilized to create new resources.
- GET is used to read or retrieve a representation of a resource.
- PUT is utilized for update resource with the newly updated representation of the original resource.
- DELETE used to delete a resource.

Resources can be accessed using RESTful application program interface. Moreover, RESTful web service relies heavily on Hypertext Transfer Protocol (HTTP) [7]. RESTful web service is usually simple to build and adapt, also provides high performance.

3.4 Real-time Web Application

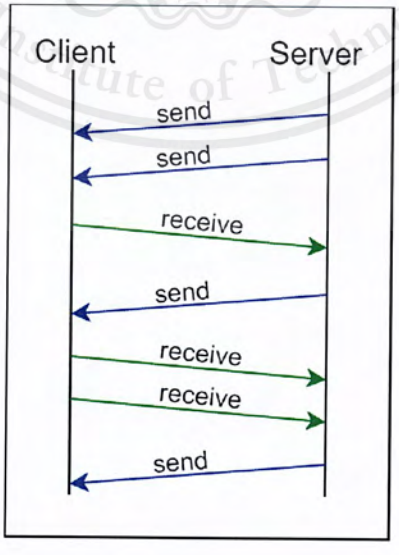


Figure 13: Real time web application communication

Generally, data communication for World Wide Web uses Hypertext Transfer Protocol (HTTP). It is a request-response based communication. For HTTP protocol, when a client requests for a web page, the web server receives an HTTP request and after it acknowledges this request, the web server sends back the HTTP response to client. However, the web server cannot response to client without request. Therefore, it is not a real-time web application.

However, web socket is a communication protocol, providing real time communication over a single connection. It is designed to be implemented on a web browser and a web server. It allows non-request communication between a browser and a web server. Web socket protocol enables interaction between a web client (web browser) and a web server. The communication between web server and web client can be occurred anytime without request from client. Moreover, web socket allows a system to support event-based communications [8].

Web socket is a representation of the new model of communications between client and server, with an operation of sockets by bidirectional communication channels over the web. Web socket provides a standard to be used to develop web applications with real-time ability and scalable. [8] The use of web socket has been applied to our application because real-time communication allows monitoring and tracking the flow of a workflow. Moreover, event-based communication allows the notification pushing from a web server to a web client easily. In addition, real-time web application is needed in order to be able to execute workflow in real-time with user interaction.

3.5 Microservices

Microservices also known as microservice architecture that structures a system as a collection of autonomous services, each of which does one thing well, that work together to perform more intricate operations [9]. Moreover, services in a collection of services are processes that communicate with each other over network. A microservice is an independent, standalone capability, designed as a process that communicates with other microservices through an inter-process communication such as RESTful API and messaging queues. Microservices are unique from a standard application because each microservice is developed, tested, deployed and scaled on demand and independent of

other microservices [10]. The microservice concept inherits all the best principles of software development, including being loosely couples, scalable on demand, and service oriented. A microservice-based application is a group of several independent standalone microservices, each offering specific, well-defined functionality, communicating through web services. Moreover, each microservice runs as a separate process and deployed independently.

There are characteristics of microservices that concerned when microservices are developed, each microservices is responsible for a single capability that can be business related and a microservices owns its data store to reduce coupling between services. From these characteristics of microservices, it suitable to apply to this project because there are various independent services that can be bind to tasks in workflow.

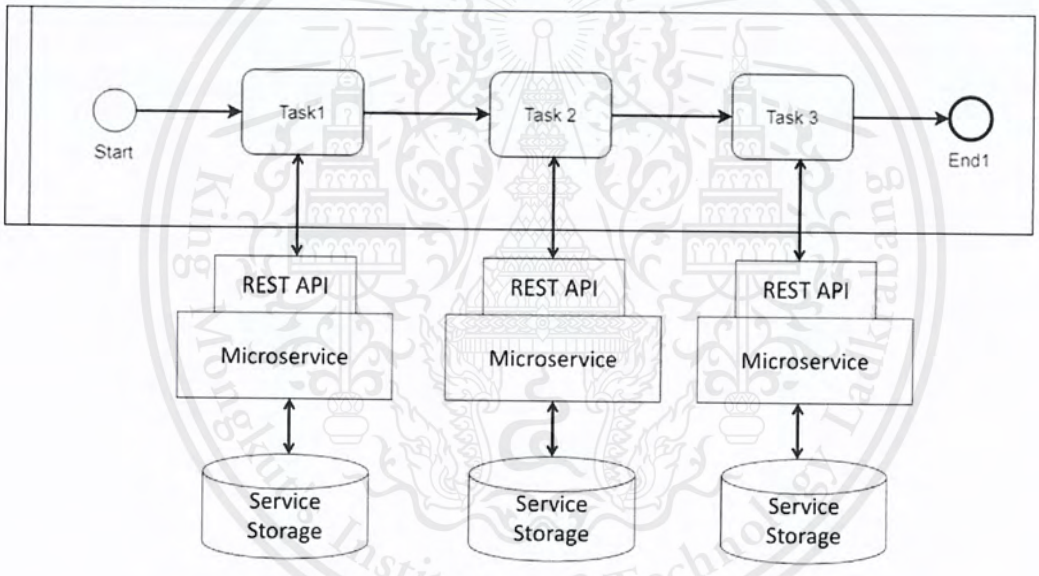


Figure 14: Business workflow diagram with microservices binding

Executable workflow used to control a business flow that each task in workflow. And each task in workflow are bound with microservices. Microservices is invoked by using RESTful web service corresponding to flow of execution. In order to reduce the complexity in develop a new independent service, microservice concept are applied in services development as a microservice instead of a single complex system that contain multiple services.

Chapter 4
Requirement Analysis / System Architecture / Design

4.1 Requirement Analysis

4.1.1 Functional Requirement

- The system provides user authentication system and registration system.
- The system allows user to edit their profile account.
- The system allows user to create a workflow.
- The system allows user to use an existing workflow template.
- The system provides a workflow composer to construct a workflow diagram.
- The system provides web services that can be bound to a task in a workflow.
- The system allows user to import their own web service.
- The system allow user to bind a provided web service and their own service into a task in a workflow.
- The system provides a form interface composer to compose a user interface, that is used to interact with a task in a workflow.
- The system allows user to customize or edit their own workflow

- The system allows workflow administrator to invite user to be a collaborator in a workflow.
- The system allows workflow administrator to pause and continue the execution of workflow.
- The system allows workflow administrator to delete the existing workflow.
- The system provides the workflow monitoring that use to trace the execution of workflow.
- The system allows user to see the execution history of workflow.
- The system has a searching system to search for the existing workflow templates.
- The system can push the notification to user.

4.1.2 Non-Functional Requirement

- The User Interface is easy to understand and user-friendly.
- The system shall not allow unauthorized users to execute a workflow.
- The system shall allow user to reset their password.
- The system has user guidance for supportability.
- The system shall display the information of the available services that could be bound into a task in a workflow.
- The system shall be a web application that runs on web browser.
- The system uses the microservice design to support maintainability.
- The system shall be a high reliable system due to microservices because services in the system are independent and collaborative.

- The workflow diagram composition is based on BPMN 2.0.
- The system can respond in real time via web socket for workflow interaction.
- The system has an event driven features using Django Channel.
- The information and data are managed by the Object-Relational Database Management System (ORDBMS) to support flexible data management.

4.2 Use Case Diagram

This section gives detail of functionality of web application in a use case diagram, where its use case description can be found in Appendix A.

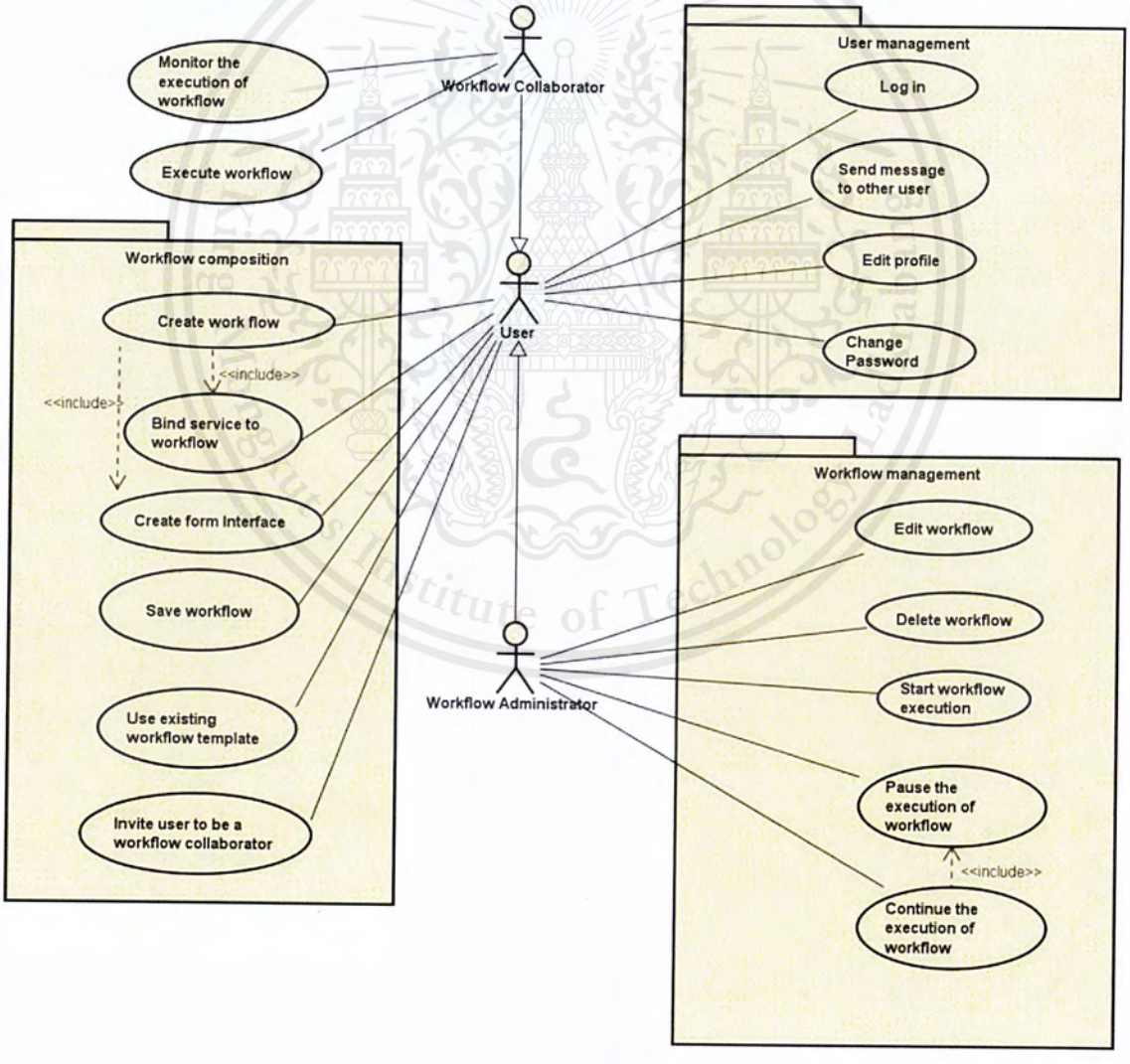


Figure 15: Use case diagram

4.3 System Architecture

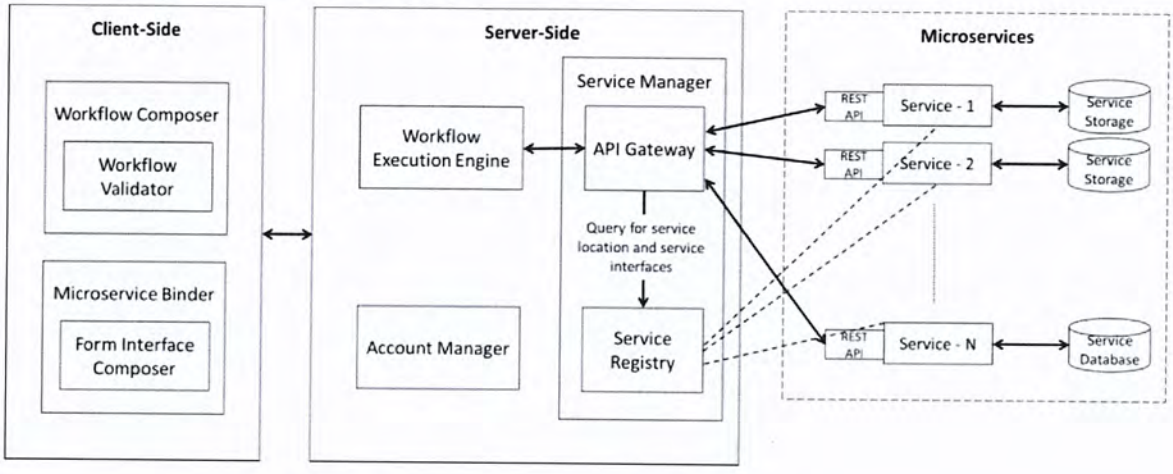


Figure 16: Architecture diagram

There are three main parts of the system, which are the client part, the server part, and the microservices part. The client-side consists of workflow composer component and microservice binder component. Therefore, the workflow diagram composition and service binding are organized on client-side. The server-side consists of workflow execution engine component, account manager component and service manager component. The server-side handles the execution of workflow by workflow execution engine component, manages user account by account manager components and manages microservices using service manager component. In addition, the communication between client and server is managed using RESTful web service to organize user data and workflow data. Web socket communication channel is used for real-time interaction between user and workflow while the workflow is executed.

4.3.1 Client-Side Component

The client-side manages user interface which interacts with user. The following components represent the roles of the client which are used to compose a workflow and bind microservices into each task in the workflow.

- Workflow composer component is a component that is used to compose a workflow diagram with a drag-and-drop feature. The workflow diagram is composed according to BPMN2.0 standards. Workflow validator is a sub-component of the workflow composer and is used to checks the syntax of the

workflow. The syntax of the workflows must comply BPMN2.0 standard. If the syntax violates the BPMN2.0 rules, the workflow composer component will not allow the user to save and execute a workflow in order to avoid execution error.

- Service Binder component is used to bind a microservice into each task in a workflow, also informs user about service input interface and service output interface, which are parameters of that service. When a workflow is executed and a service that bound into a task is invoked, the request and response parameters must follow input interface and output interface. Form interface composer is provided to create a form interface, which is a user interface of the task in the workflow. User can use drag and drop feature to construct the form interface with the provide user interface components such as a textbox, a button, a text input. However, the user needs to create a form interface according to the service input interface in order to execute service with the correct required parameters.

After a user uses workflow composer component to create a workflow of which each task in the workflow contains a service and interactive form interface, the client-side then stores the composed workflow in a form of JSON, so-called "JSON representation of a workflow". Moreover, the client will send the JSON representation of workflow to store on the server.

4.3.2 Server-Side Component

The Workflow Execution Engine is used to convert JSON representation of workflow to become a finite state machine, so-called "finite state machine representation of workflow". The finite state machine representation of workflow is an executable workflow which is Python objects. In addition, there is the account manager component which is used to organize user's data. Moreover, the Service Manager also works as API gateway that used to send a request to a microservice. The following components represent the roles of the server used to execute workflow and manipulate data.

- Workflow Execution Engine component is responsible for parsing the JSON representation of workflow to become a finite state machine object. Workflow

Execution Engine is also used to execute the finite state machine object, and maintain the state of the workflow during the execution. Moreover, Workflow Execution Engine handles both sequential execution and parallel execution.

- Account Manager component is a component that manages user account, such as user id, user password and user profile.

- Service Manager component is connected to the microservices part. It works as API gateway and service registry that contains service information, such as service location (URL), service input interface and service output interface.

4.3.3 Microservice Component

The microservice components consists of small independent services that are ready to be invoked by API gateway on the server-side. Each service has its own data storage and can be communicated using RESTful webservice for it to manipulate data. The provided services that allows user to run each task in workflow are distributed in form of microservices. The microservice architecture helps organizing service scalability and flexibility. Moreover, the number of provided services increases with less effect to other services. A server-side service discovery architecture pattern has been applied to our system in order to support increment of microservices.

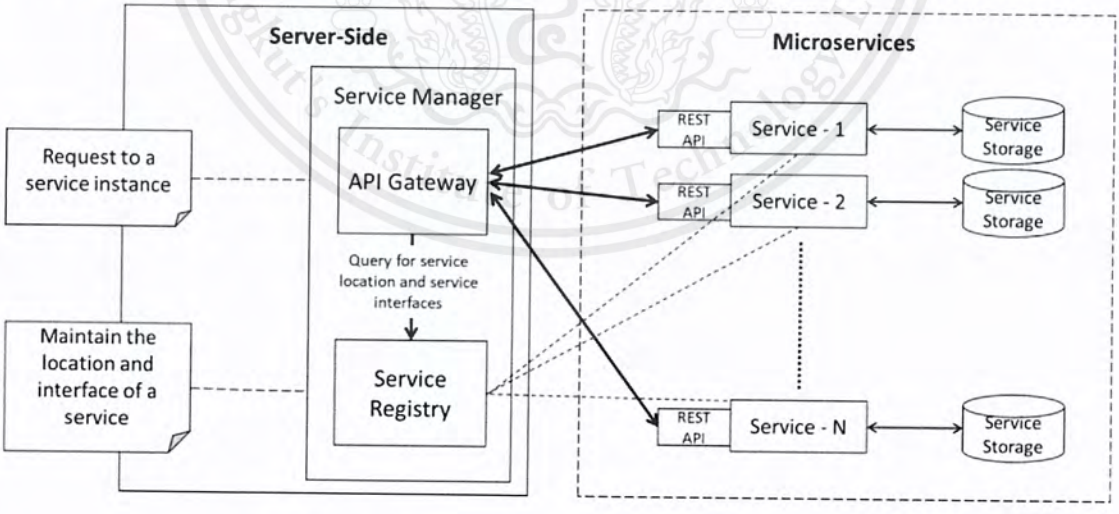


Figure 17: Server-side service discovery diagram

Service Discovery is a microservice architecture pattern. Normally, a service instance in microservice architecture can dynamically be changed because of

autoscaling, adding, or removing services [11]. Accordingly, API gateway needs to be able to determine a network locations of available service instances. Therefore, Service Discovery allows the API gateway to query service interfaces and service location from the service registry using service id.

In addition, service registry maintains the latest location and interfaces of service providers. Server-side discovery is a pattern that locates the service registry on the gateway API. Service manager component makes a request to a service via API gateway, queries the service registry to get a service interfaces and service location in order to request a service instance. However, the service registry is the key of service discovery. The service registry is a database that contains service locations and service interfaces. Thereby, the service registry needs to be highly available and up to date [11].

4.4 System Components Design

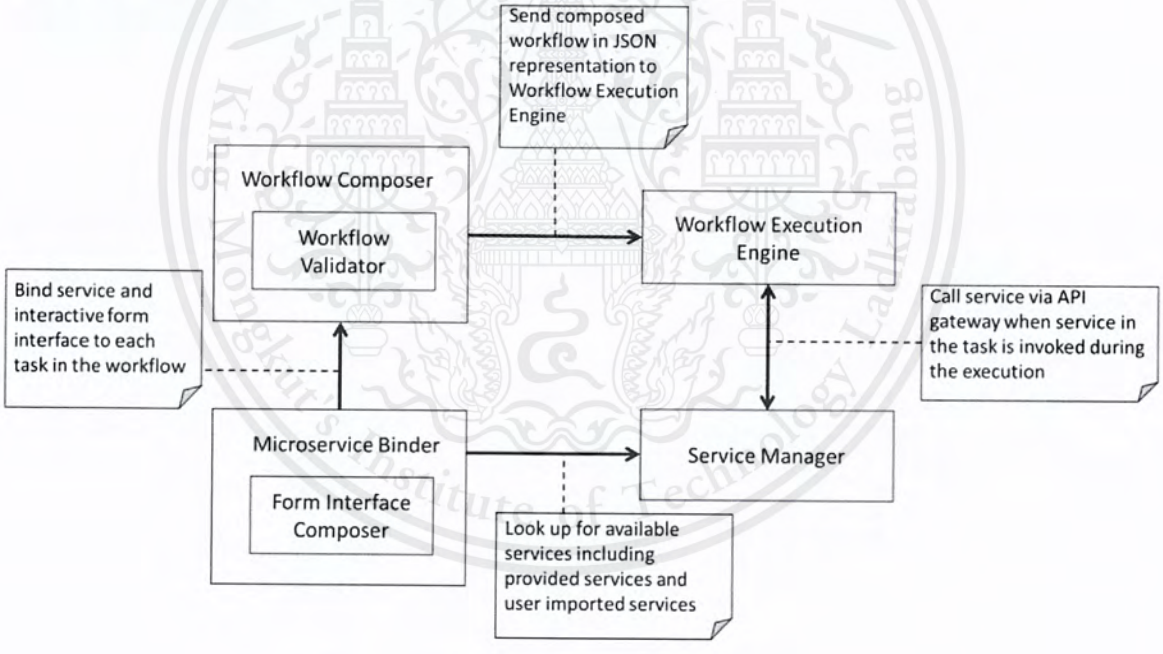


Figure 18: System Components Design

The overall components design of the system represents the collaboration of each component. These components work together in order to construct a web application from workflow and microservices. The collaboration of each component can be described as follows:

- Workflow Composer is responsible for creating a workflow diagram using BPMN2.0 standard. During the composition, workflow validator component is used for validating BPMN2.0 syntax. After a workflow is composed, a JSON representation of workflow is generated and send to workflow execution engine component.

- Service Binder is responsible for binding a microservice to a task during the workflow composition. However, the Service Binder gets the list of available services from the Service Manager. Form Interface Composer is responsible for creating input and output web form according to the service that is bound to each task.

- Workflow Execution Engine is responsible for converting JSON representation received from the Workflow Composer to become a finite state machine object. It is also, responsible for executing of finite state machine object. During the execution, service manager components is called to request a service in microservices part via the API gateway.

4.5 Overview Process of Workflow Creation

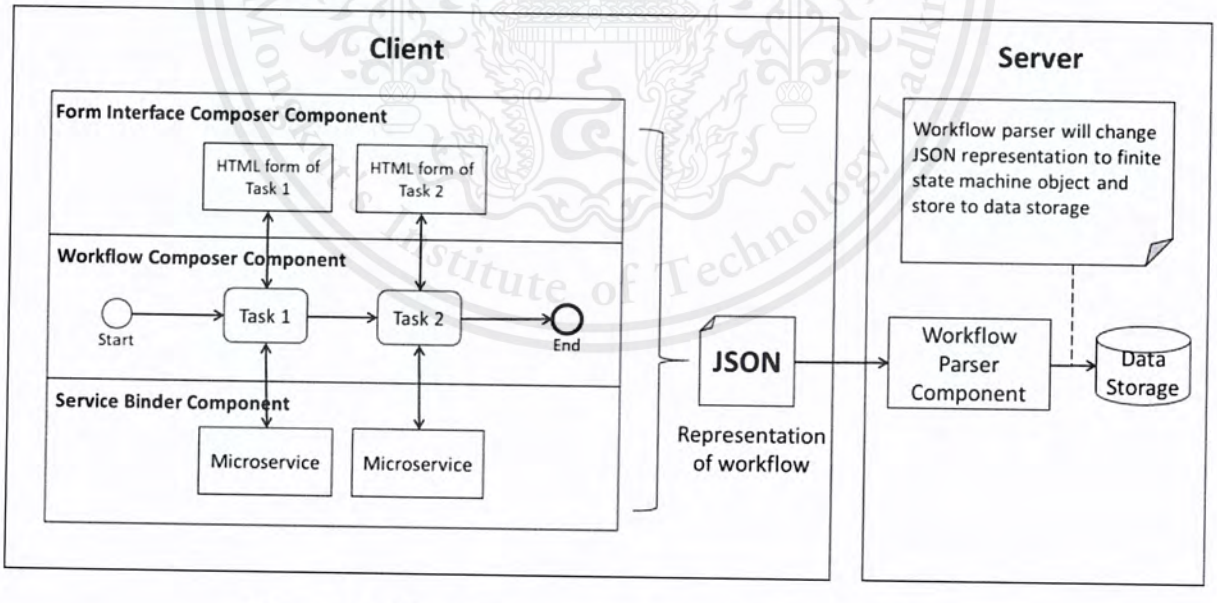


Figure 19: Overview process of workflow composition

Firstly, user creates a workflow diagram with workflow composer component. During the workflow composition, service binder component is used to binds a

microservice to each task in the workflow. Then, the user uses form interface composer component to create user interface of each task in the workflow. As a result, when the user saves the composed workflow, JSON file which represents the composed workflow information is generated; and, sends to the server. The Workflow Execution Engine parses the JSON file and constructs a finite state machine object from it. Finally, the Workflow Execution Engine serializes the finite state machine object to be a byte stream and stores to the data storage and the workflow is then ready to be executed in the future.

4.6 Overview Process of Workflow Execution

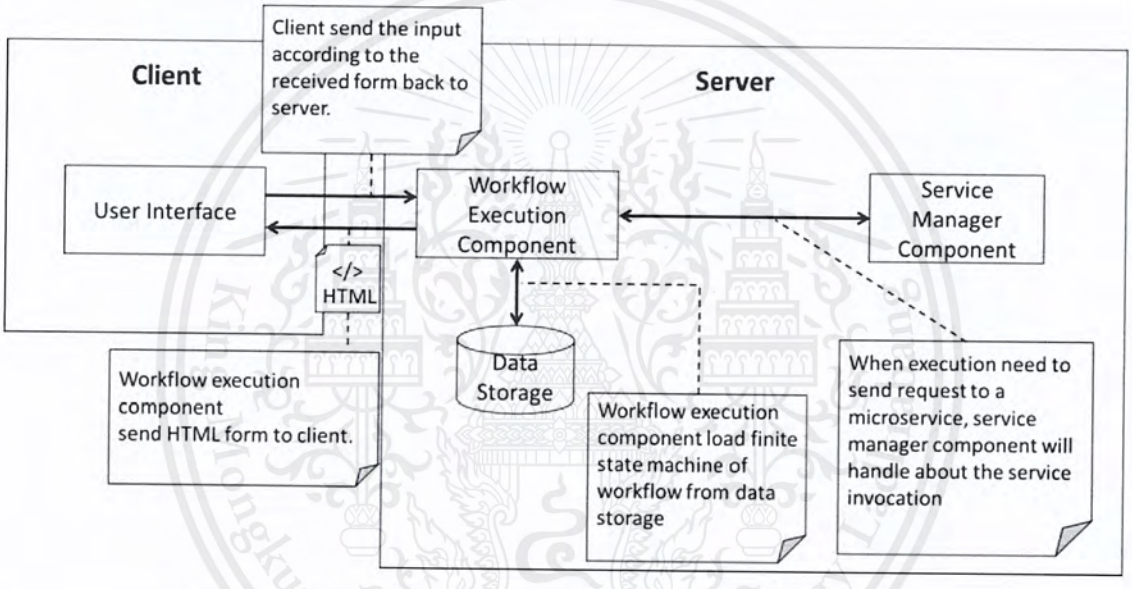


Figure 20: Overview process of workflow execution diagram

When the user requests to execute a stored executable workflow. The system loads the corresponding finite state machine object from the data storage and performs the execution using Workflow Execution Engine. Workflow Execution Engine then sends a form that has been created by the form interface composer to interact with the user. The user fills in the form and submit it to the server as an input; and, the execution performs until it reaches the end of state of the workflow. However, Workflow Execution Engine can maintain the state of the workflow. While the execution of workflow is being performed, the state of finite state machine object is updated.

Chapter 5
Software Development

This chapter describes the system development of AutoWeb. Additionally, this chapter also describes the development tools required to implement the system components. The development tools are divided into 3 parts, client-side, server-side, and microservices.

For the client-side, ReactJS, a JavaScript framework, is used to develop the user interfaces for a web application. In addition, BpmnJS is also use for drawing a BPMN 2.0 diagram and GrapesJS for generating HTML forms. For server-side, Python and Django framework are the fundamental language and web framework for implementing a RESTful web service that allows the client-side to communicate with the server. In addition, microservices are developed using Django framework and deployed using Docker container. Also, in this project we adopt the Django channel to develop real-time communication between client-side and server-side modules.

5.1 Client-Side Development

The client-side components are implemented as a web-based application since it does not require any special hardware on the device. As well as it can be defined as a cross-platform application. There are two main components that are developed on the client-side.

1. Workflow composer component
2. Microservice binder component

5.1.1 Workflow Composer

This component is used for composing and validating a workflow diagram created by a user. It can be divided into two major components, which are BpmnJS Modeler and the Workflow Validator.

- **BpmnJS modeler**

The BpmnJS Modeler is a component used for drawing BPMN diagram as shown in Figure 21. It uses BpmnJS, an open-source JavaScript library, for drawing BPMN 2.0 diagram and exporting in XML format. Additional parameters can be attached to each node in the diagram, and those parameters can be exported in XML as in Figure 22.

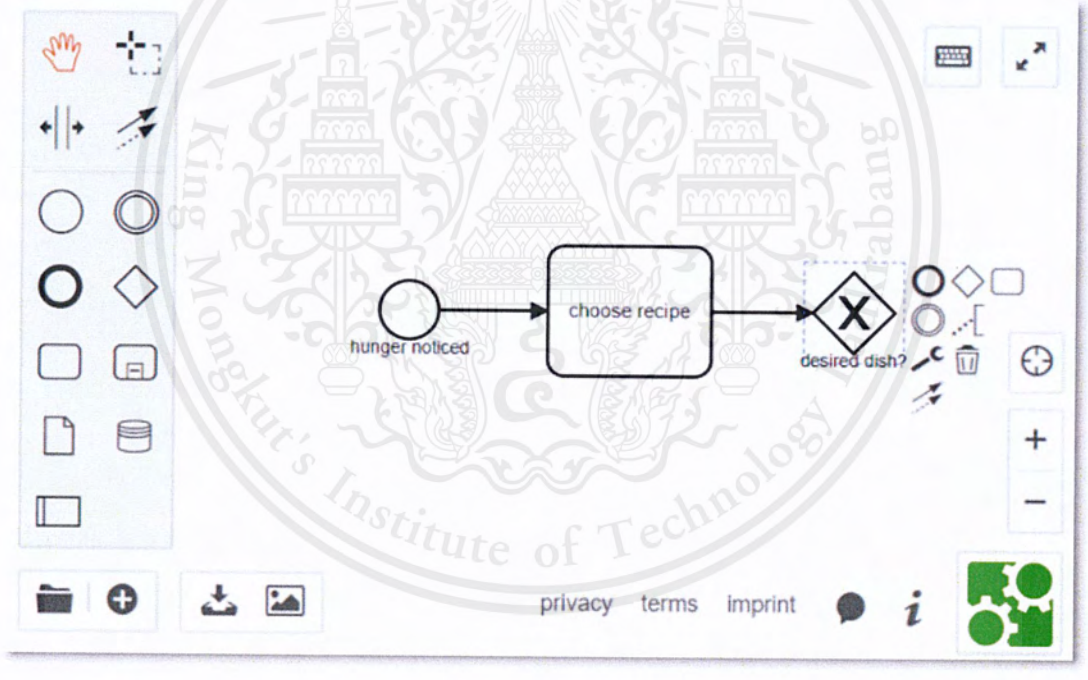


Figure 21: BpmnJS Modeler for drawing BPMN2.0 diagram

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bpmn2:definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
  xmlns:qa="http://some-company/schema/bpmn/qa"
  id="sample-diagram" targetNamespace="http://bpmn.io/schema/bpmn"
  xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL BPMN20.xsd">
  <bpmn2:process id="Process_1">
    <bpmn2:startEvent id="StartEvent_1">
      <bpmn2:extensionElements>
        <qa:analysisDetails lastChecked="2018/10/14" nextCheck="2018/10/28" comment="Hello,Test,{"a":100}" />
        <qa:analysisDetails lastChecked="2018/10/14" nextCheck="2018/10/28" comment="Hello,Test,{"a":100}" />
      </bpmn2:extensionElements>
    </bpmn2:startEvent>
    <bpmn2:task id="Task_0o739bw" />
  </bpmn2:process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Process_1">
      <bpmndi:BPMNShape id="_BPMNShape_StartEvent_2" bpmnElement="StartEvent_1">
        <dc:Bounds x="412" y="240" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Task_0o739bw_di" bpmnElement="Task_0o739bw">
        <dc:Bounds x="604" y="384" width="100" height="80" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</bpmn2:definitions>
```

Figure 22: An example XML from BpmnJS with additional parameters

To add the additional parameters, the data needs to be specified for the selected element (qa: analysisDetails in Figure 22) in JSON format in Figure 23.

```json
{
  "name": "QualityAssurance",
  "uri": "http://some-company/schema/bpmn/qa",
  "prefix": "qa",
  "xml": {"tagAlias": "lowerCase"},
  "types": [
    {
      "name": "AnalyzedNode",
      "extends": ["bpmn:StartEvent"],
      "properties": [{ "name": "suitable","isAttr": true,"type": "Float"}]
    },
    {
      "name": "AnalysisDetails",
      "superClass": ["Element"],
      "properties": [
        {"name": "lastChecked","isAttr": true,"type": "String"},
        {"name": "nextCheck","isAttr": true, "type": "String"},
        {"name": "comment","isAttr": true,"type": "String"}
      ]
    }
  ]
}
```

Figure 23: Declaration of custom element

To attach data to an element in BPMN diagram, it is necessary to define tag description by creating the declaration of the custom element as shown in Figure 23. Therefore, a single element contains loads of data, which results in an enormous size of data for a whole diagram. This approach accordingly is not the best way to handle data attachment. Another approach is splitting the data into two sets. The first set is an XML file that contains a graphical representation of a BPMN diagram, which is later converted to JSON because it is easier to extract values for further use. The other set also contains necessary attributes for workflow execution in JSON. At the end, the Workflow Composer sends both sets to the server for execution and monitoring.

- **Workflow Validator**

Another component of the Workflow Composer is the Workflow Validator. To verify the correctness of a workflow diagram, each element of a workflow should be labeled, as well as connected correctly to other elements. In other words, the elements should follow the rules of BPMN 2.0. Therefore, *bpmn-js-bpmnlint*; a plug-in for BpmnJS, serves as a validator. It provides recommended validation rules to be used. In addition we can add or customize validation rules as Figure 24.

```
{
  "extends": "bpmnlint:recommended",
  "rules": {
    "label-required": "off"
  }
}
```

Figure 24: Validation rules of bpmn-js-bpmnlint

In the modeling page example in Figure 25, errors and warnings are shown next to the elements that violate the rules, along with suggestions of what should be done to correct these errors and warnings.

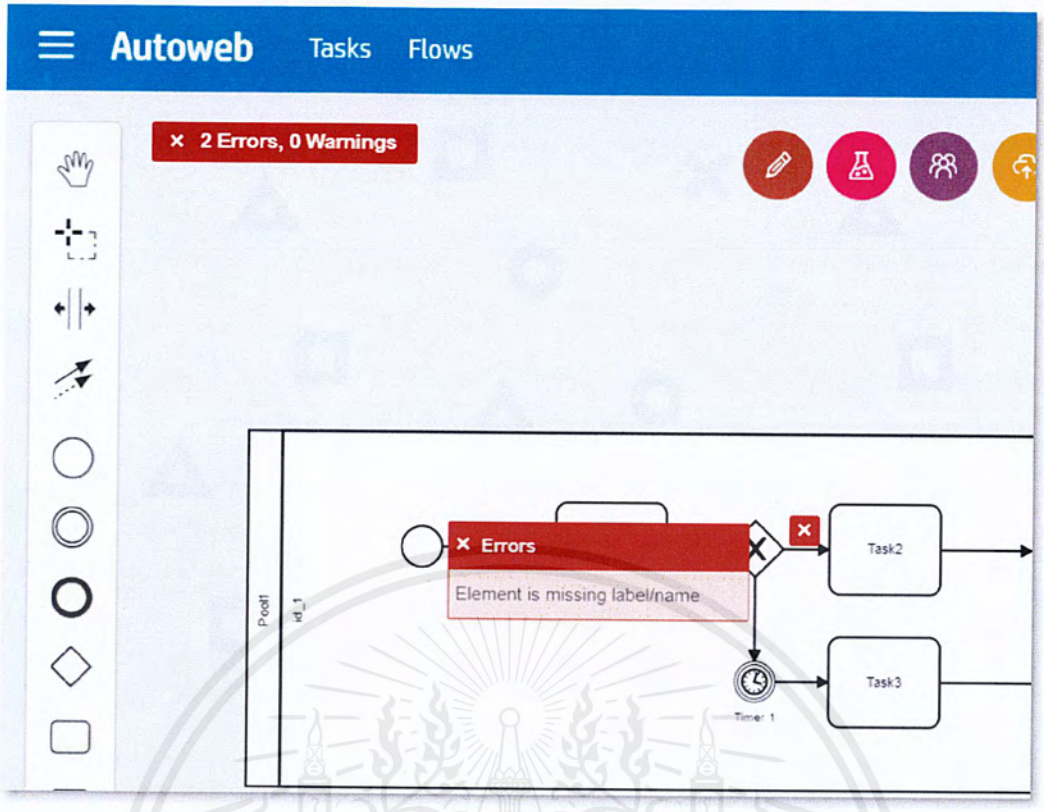


Figure 25: Example of workflow validation by bpmn-js-bpmnlint

5.1.2 Microservice Binder

Since there are several services that can be selected and bound to a task of workflow, this component is responsible for binding a service to a task in a BPMN diagram by storing an identity of the service (such as service id) into a task.

When the user clicks on a BPMN task element in the diagram, a list of available services will be displayed for mapping with the selected task on the property panel on the right-hand side as shown in Figure 26.

The identity of a service is the necessary data in JSON format required to point to the address of the service in API gateway. To be more specific, an ID of each task is used for this mapping.

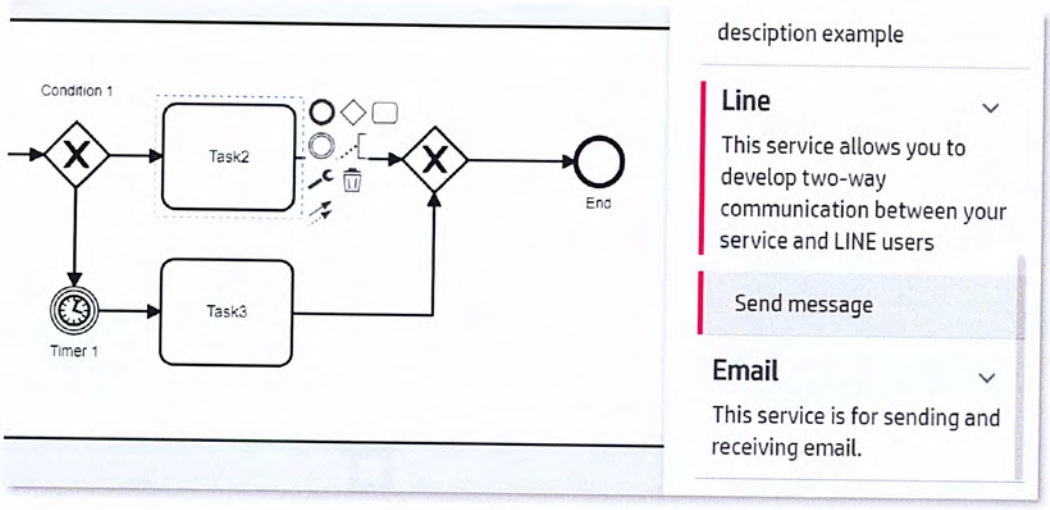


Figure 26: List of available services that can be bound into a selected task

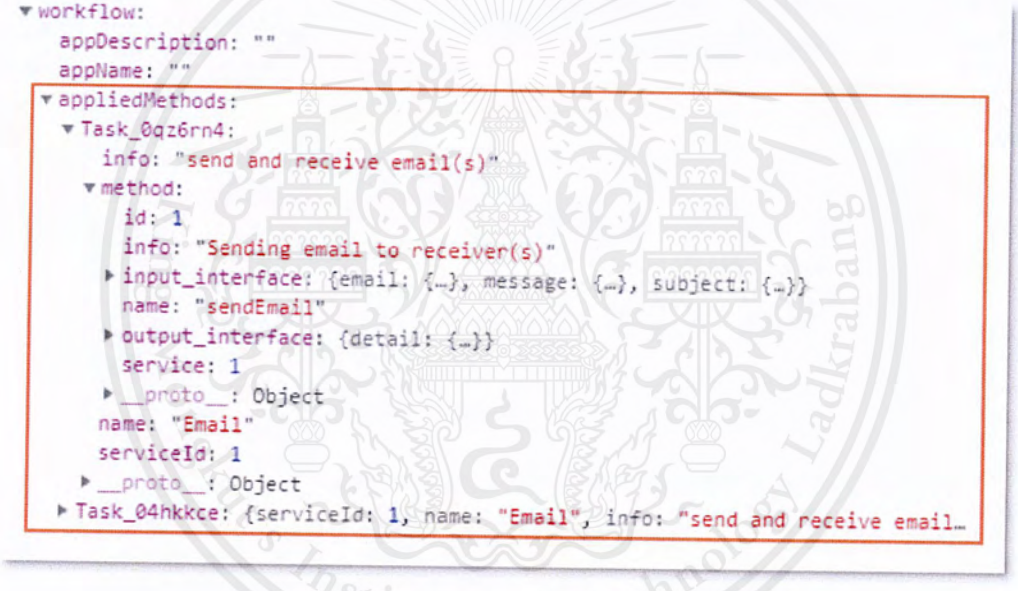


Figure 27: JSON format that store the relation between task and service

- **Form Interface Composer**

Since some service require interaction with user in order to be executed, there must be a component that allows the user to create a form based on service input interface. Therefore, the Form Interface Composer plays an important role. It is developed using GrapesJS, an open-source JavaScript library, which is used for generating HTML forms including CSS. It uses a drag-and-drop technique, so the user can focus on business logic rather than writing the whole HTML by themselves.

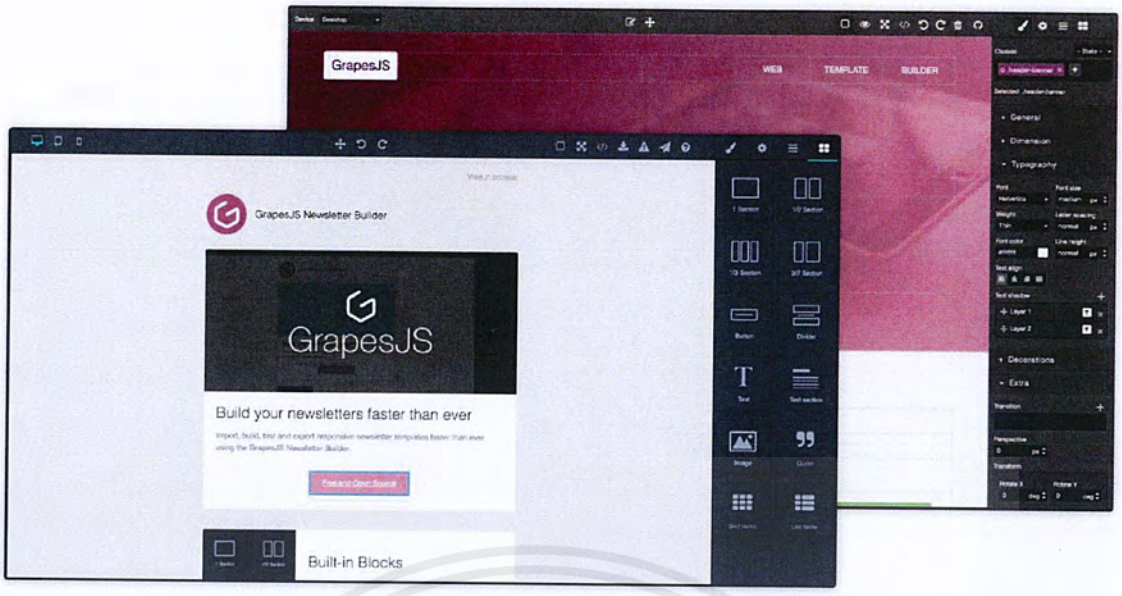


Figure 28: GrapesJS panels

5.2 Server-Side Development

For the server-side, Python and Django framework are used to develop the server-side components. Django framework is a Python web framework that handles a significant part of web development. We apply Django framework to develop the server-side application while using ReactJS for client-side development. In addition, Django framework allows us to develop RESTful web service with HTTP request and response. Moreover, Django Channels is a Python extension, which adds a new layer that allows WebSocket handling and background task running. However, there are three main components that are developed on the server-side using Django framework.

1. Workflow execution engine component
2. Account manager component
3. Service manager component

5.2.1 Workflow Execution Engine

The workflow execution engine is developed in Python with Django framework and activated when client requests to save a composed workflow or to execute a workflow.

Therefore, Workflow execution engine has two responsibility, first, converting JSON representation of workflow to finite state machine representation, and second, executing the finite state representation of a workflow.

For finite state machine representation construction, Workflow Execution Engine component is employed to parse the JSON file sent from client, and construct Python objects from workflow's information contained in JSON file. Then, Workflow Execution Engine uses the created objects to construct a finite state machine. Python objects will be created according to subclasses of BPMN elements in Figure 29. The created subclass elements are stored as the *WorkflowEngine* object, which is the finite state machine representation of workflow. Moreover, the parsing algorithm is illustrated in Algorithm 1.

Algorithm 1 Converting JSON representation of workflow to Finite state machine representation of workflow

Input: JSON representation of workflow

initilize set of state **Q** , cuurentState **C**, set of end state **F** and stateTransition δ
for each **BPMNelement** in JSON **do**
    Initialize Object **temp** according to BPMNelement type
    if BPMNelement is start event **then**
        Update current state **C** to be object **temp**
    else if BPMNelement is end event **then**
        Update set of end state **F** by object **temp**
    Update set of state **Q** by Object **temp**
    Update state transition δ by Object **temp**

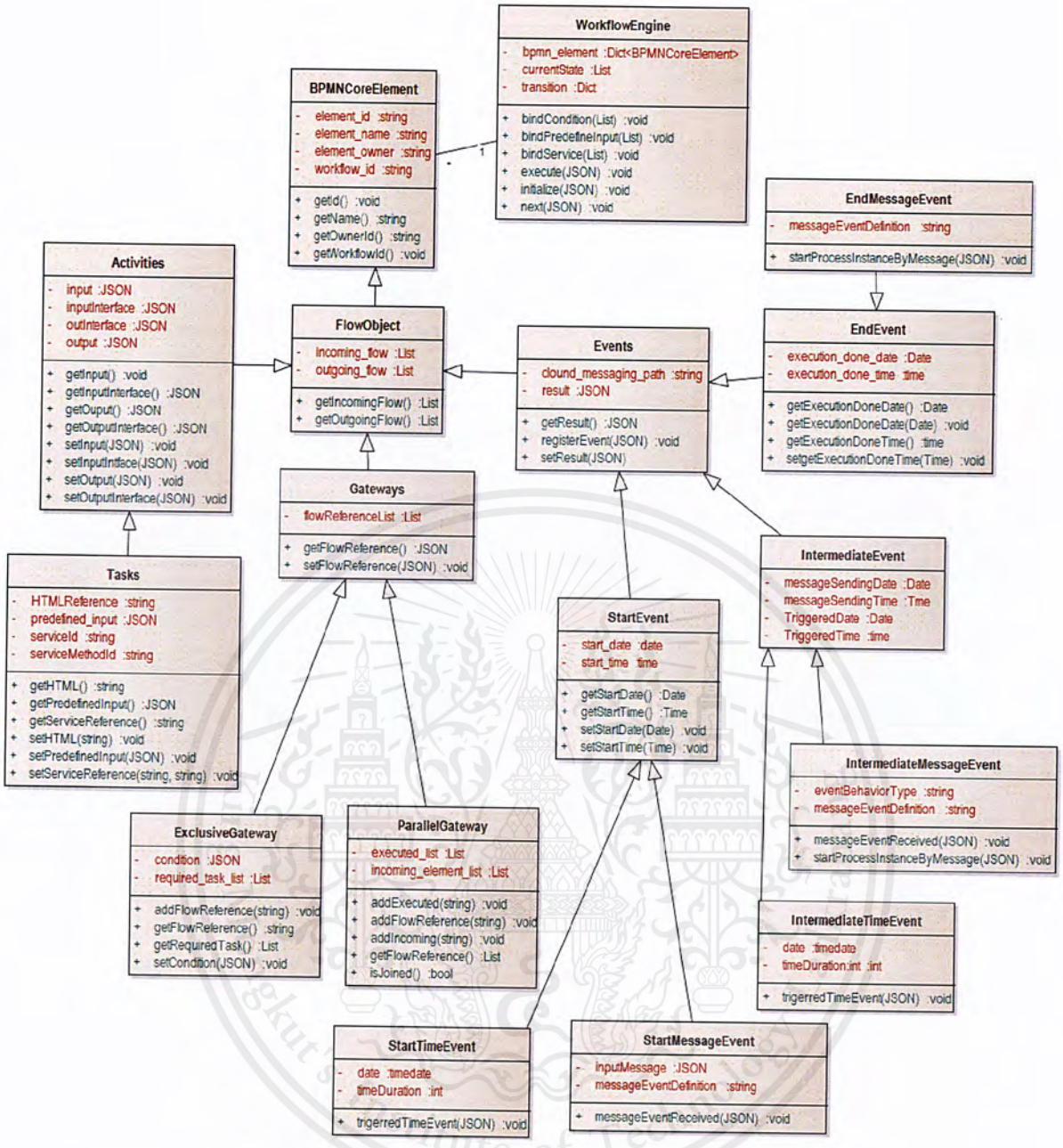


Figure 29: Class diagram of BPMN element

Concerning execution of a finite state machine representation, the execution is based on the current state of the finite state machine; which indicates by the type of its BPMN element. Each type of BPMN element object has to be executed in a different way. That is, the execution is sequential execution for task element object, exclusive gateway element object enables conditional execution. Moreover, parallel gateway element object enables parallel execution, and time event element object enables time event execution. In additional, end event element object indicates the end of the

workflow execution. The execution algorithm is shown in Algorithm 2. However, start event element object always be the first element that will be executed.

Algorithm 2 Workflow execution

```
1: function EXECUTE(C)                                   ▷ C is current state
2:     if C is start event then                          ▷ start workflow
3:         Update currentState C to next state using stateTransition δ
4:         EXECUTE(C)                                    ▷ recursive calls
5:     else if C is task element then                    ▷ task execution
6:         if C contains HTML form interface then
7:             return HTML form interface C to client
8:         else
9:             Update currentState C to next state using stateTransition δ
10:            EXECUTE(C)
11:    else if C is exclusive gateway then               ▷ conditional execution
12:        Get next state N from condition in C
13:        EXECUTE(N)
14:    else if C is parallel gateway then                ▷ parallel execution
15:        for each parallel path P in C do
16:            Create thread T
17:            T.run(EXECUTE (P))                        ▷ execute using multi-threading
18:    else if C is time event then                      ▷ time event execution
19:        if C already triggered then
20:            Remove C from event queue
21:            Update currentState C to next state using stateTransition δ
22:            EXECUTE(C)
23:        else if C not in event queue then
24:            Push C in to event queue                  ▷ push time event to event queue
25:        else C is pending                             ▷ time event is still pending
26:            return
27:    else if C is end event then                       ▷ reach end of workflow
28:        return
```

Timer event execution has a background process to manage time events. When the execution of workflow reaches timer event object, time event will be pushed to an event queue. The background process is used to fetch and check time events, also change the status of time event from pending to triggered. The time event management algorithm is shown in Algorithm 3.

Algorithm 3 Time event management

```
1: while true do
2:     for each time event E in event queue do
3:         if E.time > current time then
4:             status of E is changed to 'Triggered'
5:
```

5.2.2 User Account Manager

User Account manager component manages user account information and authentication. It is developed using Python and Django framework using RESTful style. The list of the services and their description are described in the following table.

Table 10: List of APIs in Account Manager

Path	HTTP Method	Action
api/register/	POST	Create an account
api/login/	POST	Perform authorization, if grated, a session for the user is created
api/logout/	POST	End the session of the user
api/change_password/	POST	Replace an old password with a new one
api/validate_token/	POST	Return a Boolean of the user's session active status
api/collaborator/{project_id}	GET	Return collaborators of the project based on its *project_id*
api/collaborator/	POST	Add a collaborator to the project
api/workflow/	GET	Return workflows of the user
api/workflow/	POST	Create a workflow under the user's id

5.2.3 Service Manager

Service manager component is a component that work as API gateway and stores all of the provided microservice information. This component is developed to be a web service using RESTful. It consists of web services that allows user to request for microservice detail. However, the list of available services is described in the following table.

Table 11: List of APIs in Service Manager

Path	HTTP Method	Action
api/all_service/{id}	GET	Returns every method of the service on the specific *id*
api/service/{id}	GET	Returns service information based on its *id*
api/service/	POST	Registers a service
api/connecting_service/{id}	GET	Returns methods that can be connected to the specific method
api/connecting_service/	POST	Creates a pair of connecting services
api/method/{id}	GET	Returns method of the specific *id* information
api/method/	POST	Creates a new method
api/lookup/	POST	Returns the path of the method

5.3 Microservices Development

Microservices provided by this project, are developed using Django framework. Microservices are registered in the service registry of the service manager component. In addition, third-parties RESTful microservices can be used to in AutoWeb.

5.3.1 Docker Container

Container technology addresses the deployment problems by separating applications from the infrastructure dependency. It addresses with containers; which allow packaging the application with all its dependencies, including directory structure. Moreover, the container allows packaged application to run the same way, across different machines and environments [10]. In this thesis, Docker container is used to

deploy several microservices. In a Docker container, there are application run on specific port with its own data storage as shown in Figure 30.

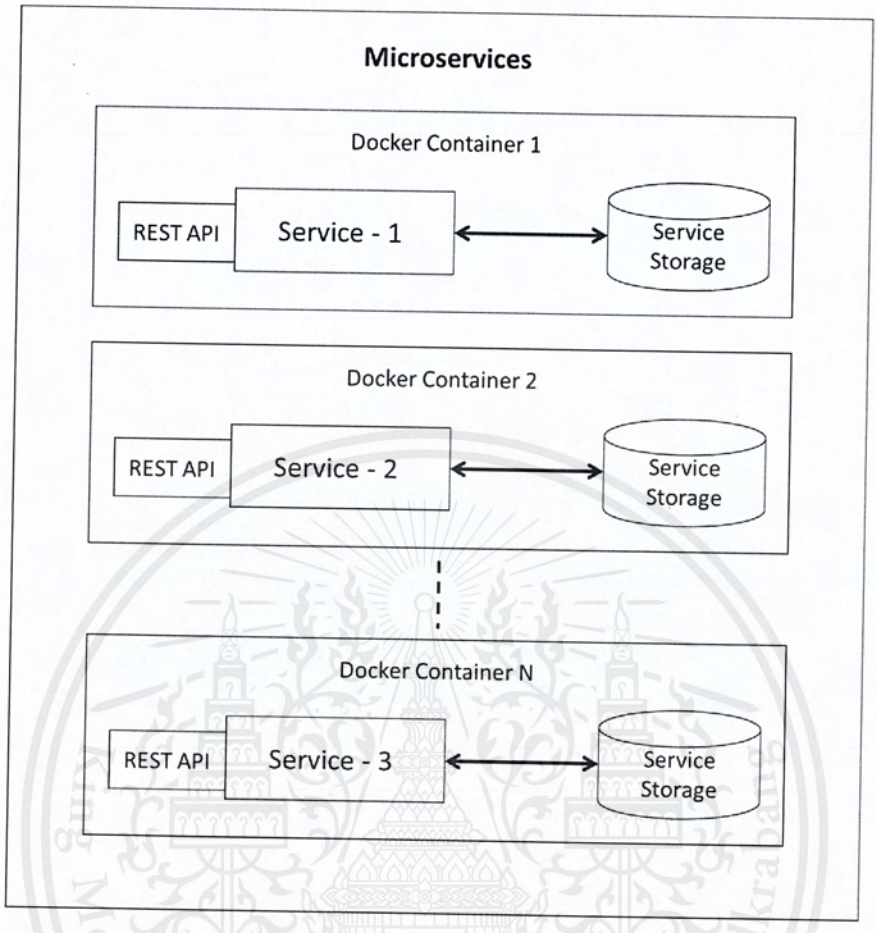


Figure 30: Microservices deployment using Docker

Chapter 6

Results

This chapter covers the results of our software development, which includes how the user interacts with a real-world application created from an executable workflow implemented by AutoWeb.

6.1 Web application

The results can be described as a group of functionalities of AutoWeb, i.e. workflow display, workflow composition, microservice binding, form interface composition, workflow execution, and workflow monitoring.

6.1.1 Workflow display

Figure 31 shows a screen of the home page of AutoWeb. The home page will be displayed once a user is logged into the web application. The main purpose of the home page is to display a list of workflows already created by the user and a list of workflows of which user is a collaborator; see Figure 31 for more details. The circle No.1 in Figure below indicates a list of workflows created by the user, and the circle No2. indicates a list of workflows which user is a collaborator. Moreover, the user can create a new workflow using the button at the circle No3, and the workflow composition page will appear when the user click "+ Flow" button.

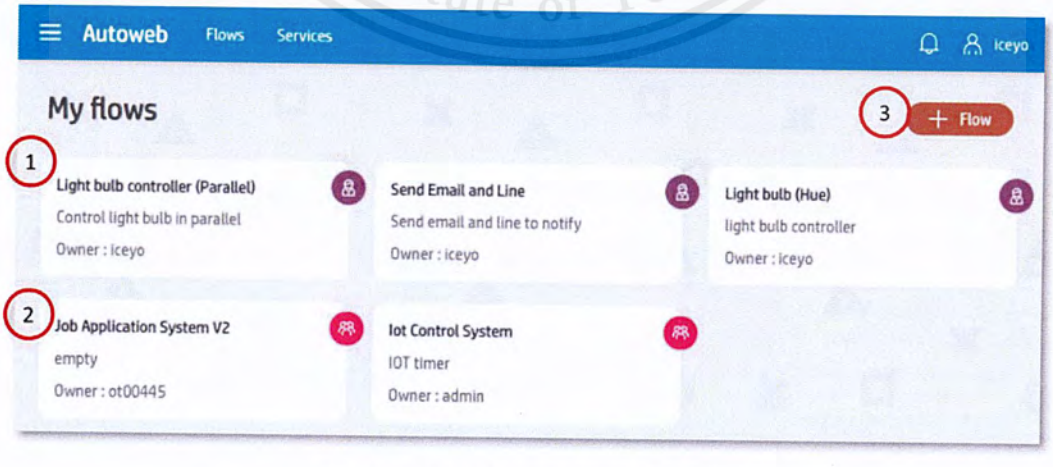


Figure 31: Homepage of AutoWeb

6.1.2 Workflow Composing Step

AutoWeb allows the user to compose a BPMN 2.0 diagram on a workflow composition page. Moreover, the workflow composition page allows user to bind a microservice and a form interface into each task of the workflow. In Figure 32, the user can compose a workflow diagram by dragging a BPMN element from the left-most-side panel which is indicated by the circle No.1 and drop it into a workflow composition canvas at circle No.2, then proceed to the microservice binding step which is described in the next section.

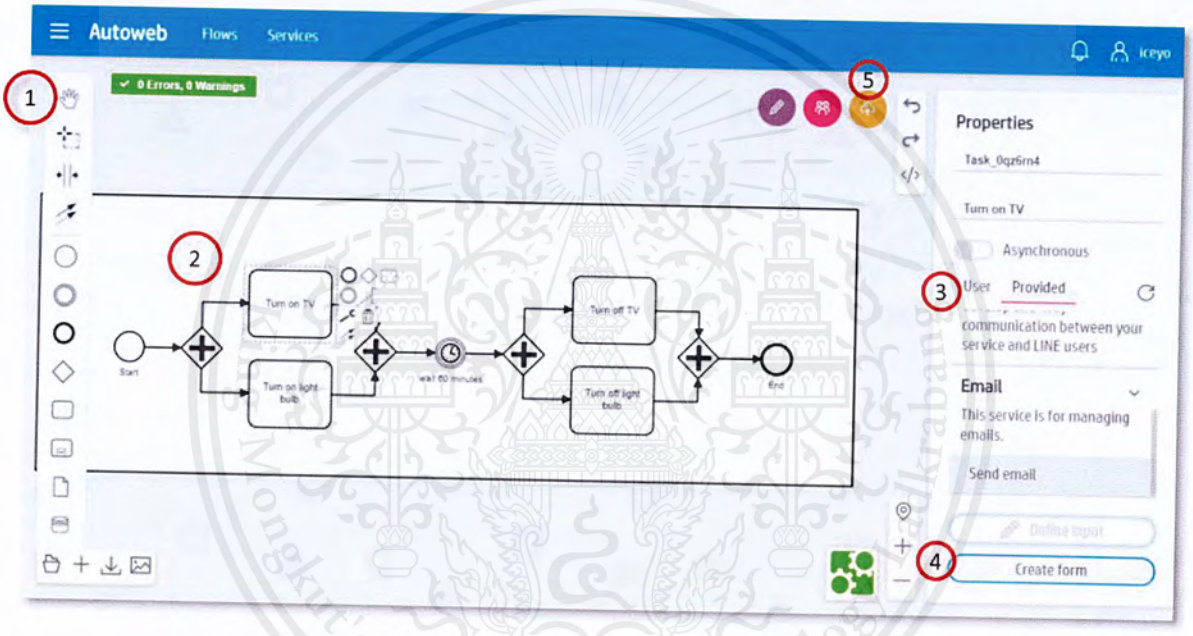


Figure 32: Workflow composing page of AutoWeb

6.1.3 Microservice Binding Step

After finishing the workflow composition step, the user can bind a microservice into each task of the composed workflow. Firstly, the user selects a task of workflow and then chooses a service from the right-most-side panel in Figure 32 which is indicated by the circle No.3 to bind to the selected task. When a microservice is bound to a task, a popup window will appear which displays input interfaces and output interfaces of the selected service as shown in Figure 33. However, there are two types of service, i.e. a system-provided service and a user defined service. The system-provided service is a

service which is provided by the system and this service's input interface and output interface cannot be modified.

The user-defined service is a service that user develops it by himself/herself and can later import it to bind to a task of a workflow. The user can modify all of the service information includes input interface, output interface and service location as shown in Figure 34.

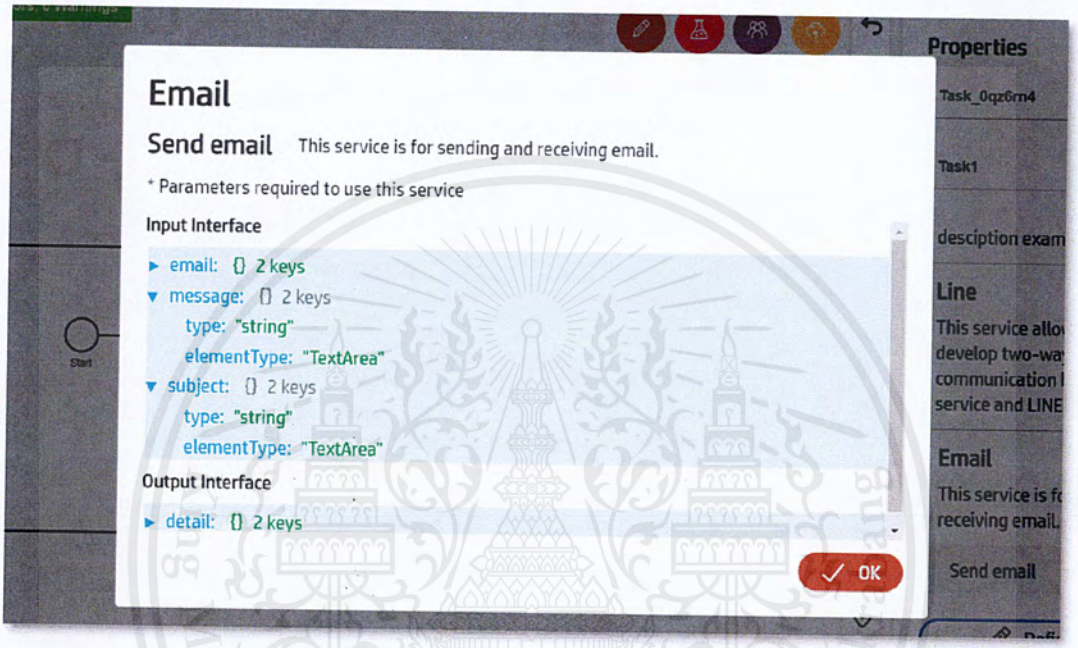


Figure 33: Input interface and output interface of a service

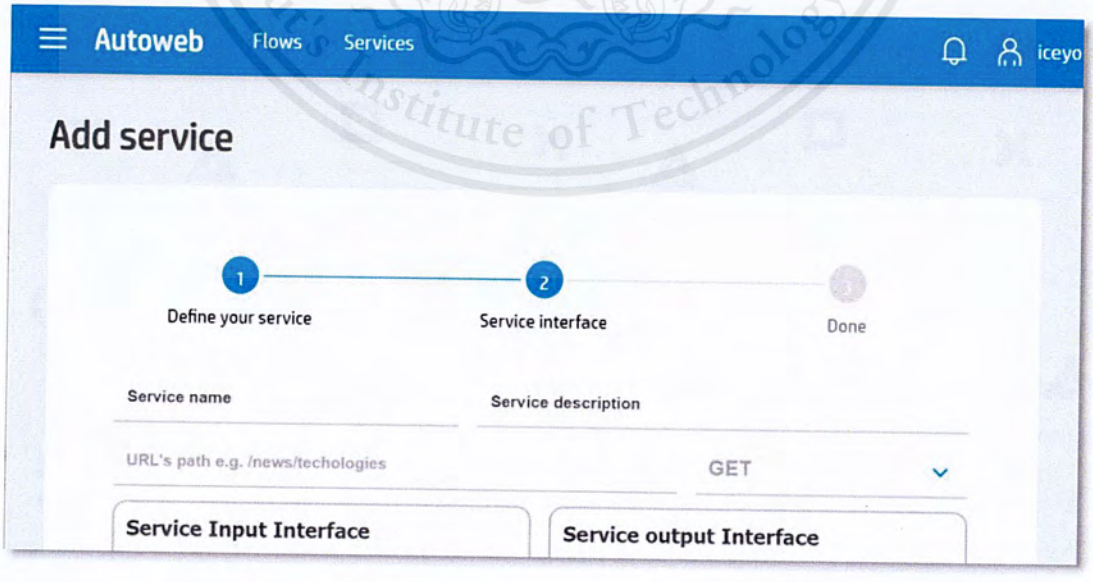


Figure 34 : A page for adding a new user service

6.1.4 Form Interface Binding Step

After the microservices binding step, the user can now attach a form interface to each task of workflow, which is an input web form for the service that is bound to each task. In Figure 35, A form composing page allows the user to design their own forms interface, where the user can drag user interface components such as textbox, button, text input from the right-most-side panel which is indicated by the circle No.1 and drop it to an area indicated by the circle No.2. However, the user needs to specify an ID to every user interface component to match the input interface of the service as shown in the circle No.3, the user can specify the ID of each user interface component element in the circle No.4. After the user binds a microservice to a task and binds service's form interface to that task, the user can save the entire composed workflow by click at the button that indicates by the circle No.5 in Figure 32.

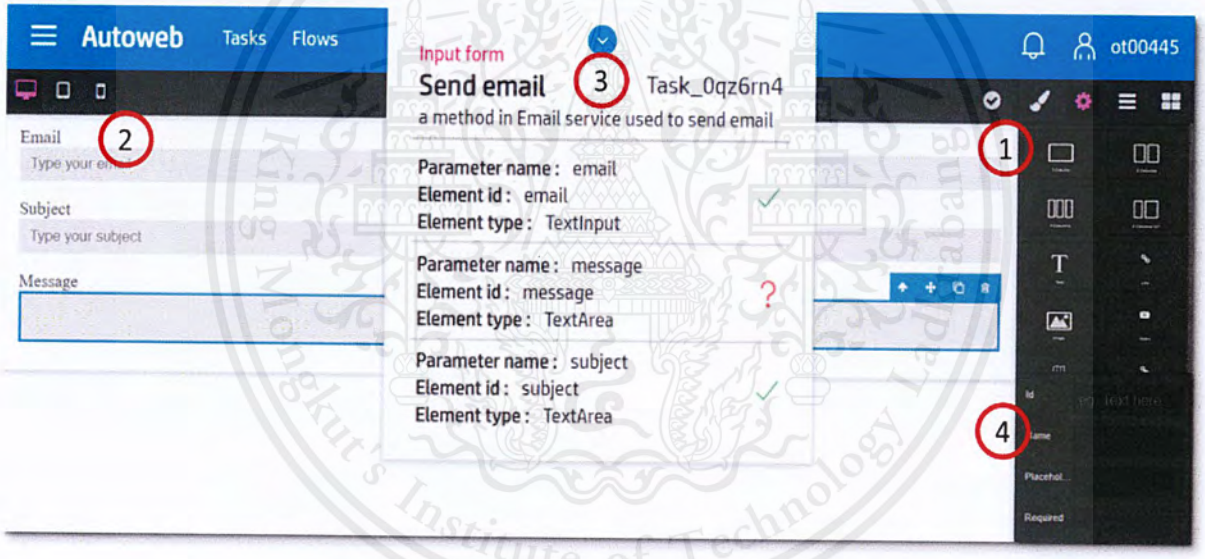


Figure 35: A form composing page of AutoWeb

6.1.5 Workflow Execution

After the composed workflow has been saved, the workflow will be listed in the workflow display page (See Figure 31) and now ready to be executed, the user can select a workflow to execute from the workflow display page in Figure 31. When the workflow is executed, the form interface will be popped up on the workflow execution

page as shown in Figure 36; this will allow the user to fill in the form in the area that indicates by the circle No.1, and submit the form using a button which is indicated by the circle No.2. After the user submits the form, the next form of the next task being executed may appear in the similar manner, the workflow will then be executed until the end of the process.

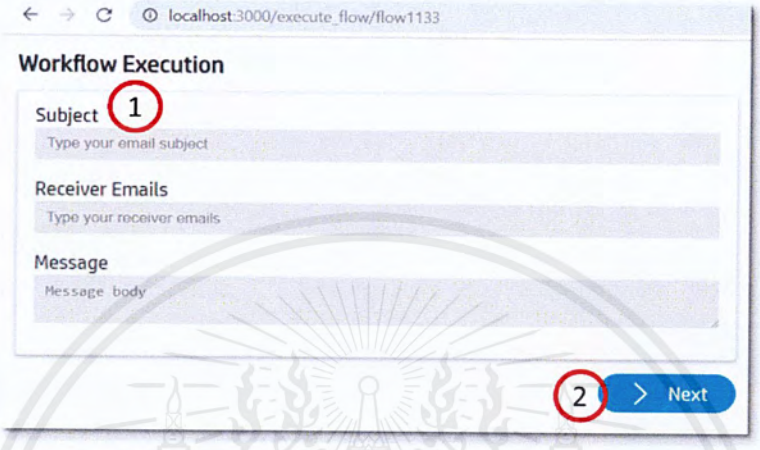


Figure 36: A workflow execution page of AutoWeb

6.1.6 Workflow Monitoring

AutoWeb also provides a workflow monitoring page to let a user trace an execution of a workflow by showing its current execution and history on a BPMN diagram of that workflow, as shown in Figure 37. The circle No.1 indicates the execution history includes the name of user who executed the tasks, execution date, and execution time. The circle No.2 indicates the graphical monitoring screen which show a BPMN diagram for tracking the progress of the execution. The components of the diagram highlighted with a grey color represents the task already executed, whilst the component with a green color represents the current point of execution. Moreover, the monitoring page is updated in real-time while a workflow is being executed.

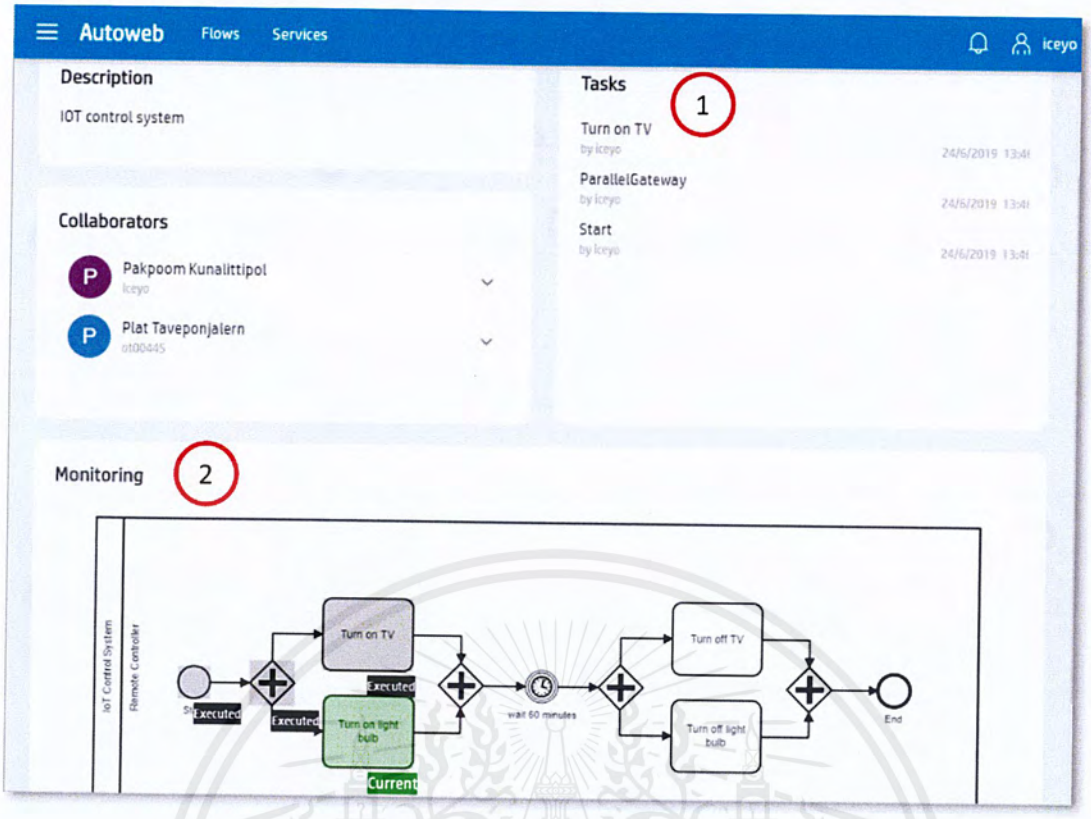


Figure 37: Workflow monitoring page of AutoWeb

6.2 AutoWeb Demonstration

AutoWeb is used to develop two real-world applications, which are a web application for applying jobs and a home automation.

One of our purposes is to demonstrate how we can use AutoWeb to create a real-world web application and we think that a job application system is a suitable example as it also involves several roles to perform different tasks. There are four roles: department manager, human resource (HR), recruiter, and applicant. An applicant is a candidate for a job position, and he or she needs to send his/her information to a company, while a recruiter is the first person who reviews the application form. A human resource officer is responsible for doing the first interview, while the second interview is done by a department manager; who wants to fill that position.

6.2.1 Job Application System

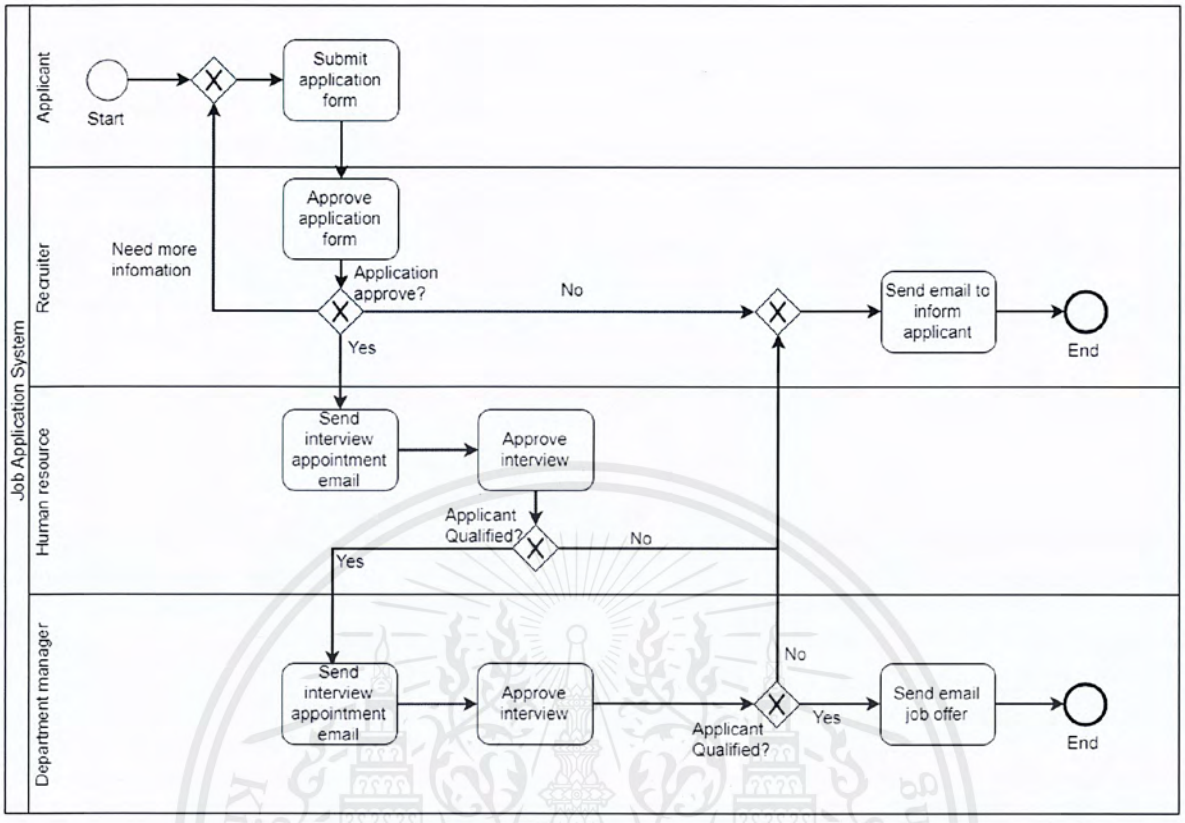


Figure 38: Job application system in BPMN

The BPMN diagram of the Job Application System is created as shown in Figure 38. After the system is launched, the current execution allows the applicant to fill the required information in the form created by the workflow owner as shown in Figure 39. This task is completed after the filled form is submitted.

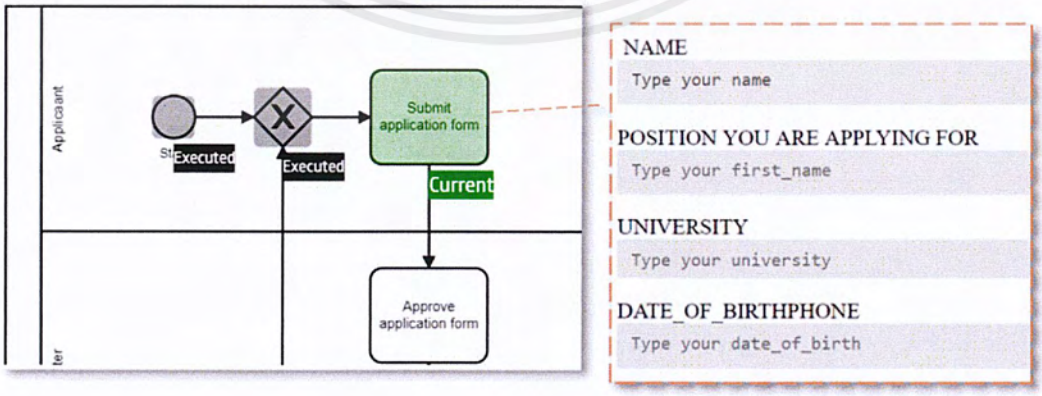


Figure 39: Current execution at Submit Form task

The recruiter then reviews the applicant's resume and decide whether to approve or reject the application. If the application is approved, the human resource then emails an interview arrangement and later conduct an interview and report a result. If the applicant has passed the first interview, the department manager then conducts another interview by emailing the interview appointment and report the result. Finally, the department manager sends a job offer if the applicant is accepted, which leads to the end of the execution.

6.2.2 Home Automation

Another application of AutoWeb is to automate an IoT control system. IoT (Internet of Things) is a network of physical devices that employ sensors and use APIs to exchange data over the internet. As the technology advances, IoT has become very trendy. Integrating our project with IoT create a new way of collaboration between human and machines.

For the demonstration purpose, AutoWeb is used to create a system that controls a television and a light bulb using a workflow. The IOT devices that we brought into this demonstration are Philips Hue bulbs and Zmote. Philips Hue bulbs are IOT light bulbs which can be controlled by APIs via Zigbee communication. To control a Hue bulb, a command sent from the APIs to a Hue bridge which is responsible for passing this command via Zigbee communication to the light bulb one wants to control. A microservice for Hue bulb control is developed using these APIs.

Zmote is another IOT device which allows one to control a home appliance which is operated by a remote control. A microservice can be developed from a Python Zmote APIs for controlling this home appliance. To control a home appliance, e.g. a TV, initially a Zmote must be connected via Wifi, then a microservice sends a remote-control code to Zmote which will then convert the control code into infrared signal in order to control that appliance.

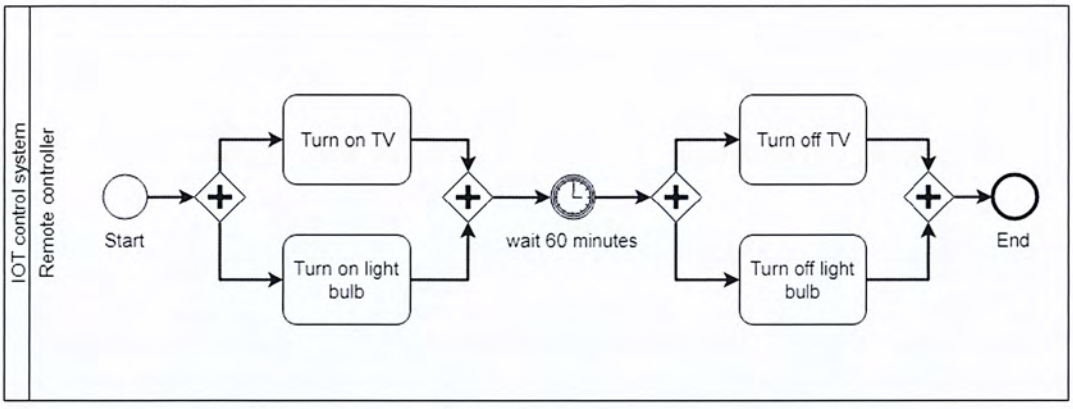


Figure 40: BPMN diagram of IoT Control System

In Figure 40 illustrates a workflow to automate a service of control of smart home devices. The procedure is to turn on the television and the light bulb; and after 60 minutes they will be turned off. The workflow first begins at the start event, then proceed to the parallel gateway where both outgoing flows are executed parallelly. This mean, the television and the light bulb are turned on simultaneously. After that, the execution is at the timer event as shown in Figure 41.

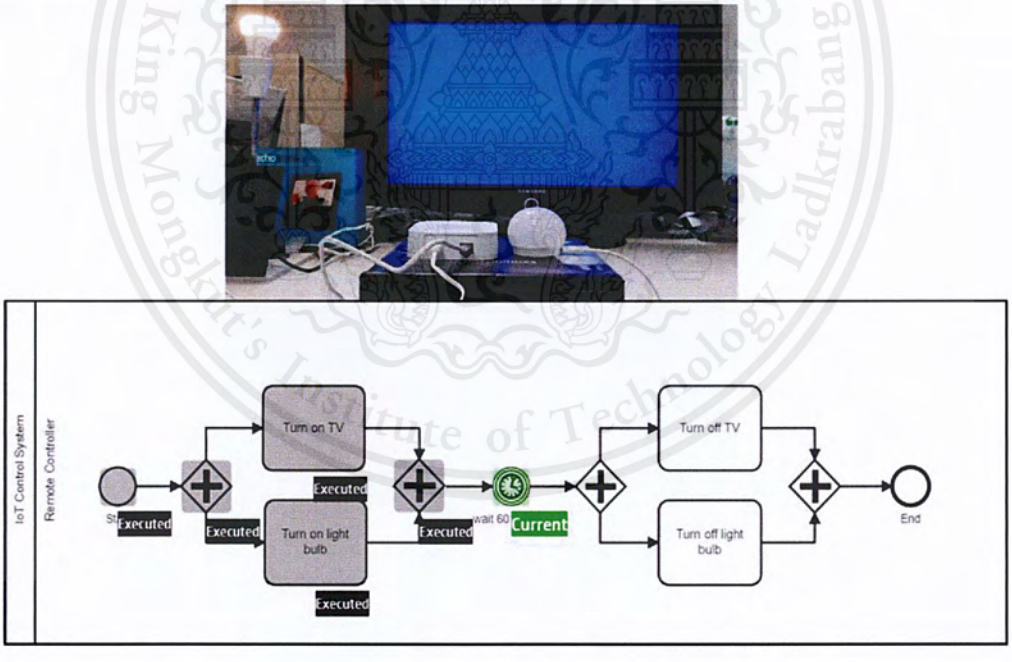


Figure 41: Current execution at timer event

At the timer event, the system waits for 60 minutes. Then the system proceeds to the parallel gateway, both outgoing flows are executed in parallel again. After the television and the light bulb are turned off, the system finally moves to the end event and to end the execution eventually as shown in Figure 42.

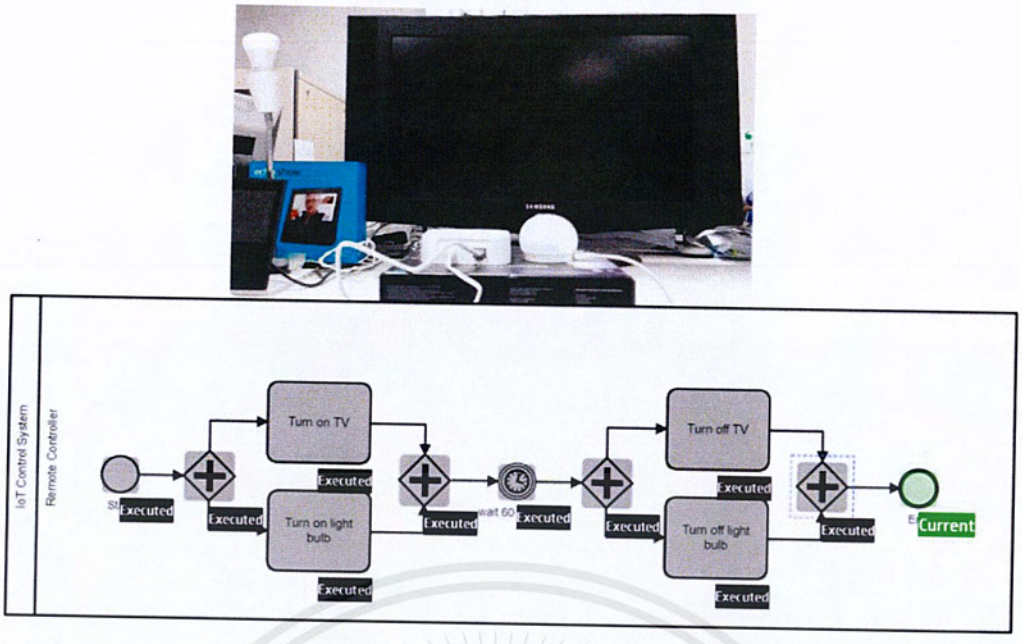


Figure 42: Current execution at end event

From the IOT case study, it demonstrates that an executable workflow can be used to automate a collaboration among machine and even together with human.

Chapter 7
Conclusion and Future Work

In this thesis, AutoWeb is proposed to be a developing environment for constructing a web application from an executable workflow. The aim of this project is to improve business efficiency and productivity as well as accelerate collaboration among people (as well as machines).

AutoWeb consists of three components: Workflow Composer, Microservice Binder, and Workflow Execution Engine. Workflow Composer is a tool for composing and validating a BPMN workflow diagram. This component also helps an organization visualize their business process with business workflow. The next step after a workflow is created is to bind an available microservice to each task of the workflow; this is done by the Microservice Binder. The last component is the Workflow Execution Engine, which is responsible for execution of a workflow already composed, and the outcome of such execution can be considered as the result of running a web application.

To prove the usefulness of AutoWeb, we employ AutoWeb to create two real-world applications, which are a web application for applying jobs and another for home automation. As a result, it vindicates that AutoWeb can be used to automate a collaboration among people and even among machines.

Due to the intensive scope of AutoWeb, it is hard to complete all the work require to develop a complete AutoWeb, what we have achieved is down to the software requirement in Chapter 4. Of course, there is still plenty of room for improvement and new emerging idea which can put into its future development. So far, the current Workflow Validator, which has been developed as a part of Workflow Composer component, can support for validation of only BPMN workflow diagram, under the

syntax of BPMN2.0, but still cannot validate the correctness of microservices bound to the workflow.

In addition, Workflow Execution Engine still has a limitation that it can execute only the basic BPMN notation such as start event element, end event element, task element, time event element, exclusive gateway element, and parallel gateway element. Therefore, Workflow Execution Engine can be further developed to support the execution of some other or even all other BPMN elements.

For our future work, to keep up with future development, the Workflow Validator should be able to validate that every task and its binding microservices of an entire workflow correctly. Moreover, it should be a rigorous study to prove that a finite state machine is a suitable model of any BPMN workflow execution.

To create an immersive world for an AutoWeb application, the technology like virtual reality (VR) and augmented reality (AR) can be deployed as the user interface for AutoWeb in order to create a more real-life interaction among people while collaborating when a business workflow is being executed.

References

[1] S. Siciliano, D. Herbert, P. Veenstra, M. Blythe. (2017, October 2017). *Microsoft Flow: Get start with Microsoft Flow* [Online]. Available: https://docs.microsoft.com/en-us/flow/getting-started.

[2] C. Nicolai. (2018, November 14). *Comuda: Platform for Workflow and Business Process* [Online]. Available: hhttps://camunda.com/products/bpmn-engine/.

[3] J. Lapanant, P. Watthanavarangkul, and C. Phongsuwan. (2016). *Workflow Engine: Web Application Development*. Available: International College King Mongkut's Institute of Technology Ladkrabang (KMITL-2017-IC-B-003-007).

[4] S. White. (2016, October 16). *IBM Corporation: Introduction to BPMN*. Available: https://www.omg.org/news/meetings/workshops/soa-bpm-mda-2006.

[5] B. Silver. (2012). BPMN implementer's Guild: BPMN method and style, 2nd Edition. Cody-Cassidy Press.

[6] M. Rouse. (2005, April). *TechTarget: Finite state machine* [Online]. Available: https://whatis.techtarget.com/definition/finite-state-machine.

[7] G. Hiller. *Building RESTful Python Web Services*. PACKT Publishing Limited.

[8] M. Ubl and E. Kitamura. (2010, October 20). *Maltentroducing WebSockets: Bringing Sockets to the Web*. Available: https://www.html5rocks.com/en/tutorials/websockets/basics.

[9] C. Richardson. (2018, August), *Microservice Architecture: What are microservices* [Online]. Available: https://microservices.io.

[10] P. Kocher. (2016). *Microservices and Container.* Addison Wesley, 2018.

[11] C. Richardson. (2015, October 12). *Service Discovery in a Microservices Architecture* [Online]. Available: https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/.

[12] J. Freund and B. Rucker. (2016). Real-life BPMN: using BPNM, CMMN and DMN to analyze, improve, and automate processes in your company. CreateSpace, pp. 23-116

[13] S. Ghezala. (2018, October 11). *Validate and Improve your BPMN Diagrams with bpmnlint* [Online]. Available: https://bpmn.io/blog/posts/2018-bpmnlint.html.

[14] A. Godwin. (2019, February 1). *Django Channels: Routing and Multiple Protocols* [Online]. Available: https://channels.readthedocs.io/en/latest/introduction.html.

[15] B. Selic and G. Gullekson. (1994). *Real-time Object-oriented Modeling.* Wiley.

Appendix A

Use Case Diagram

A.1 Use Case Diagram

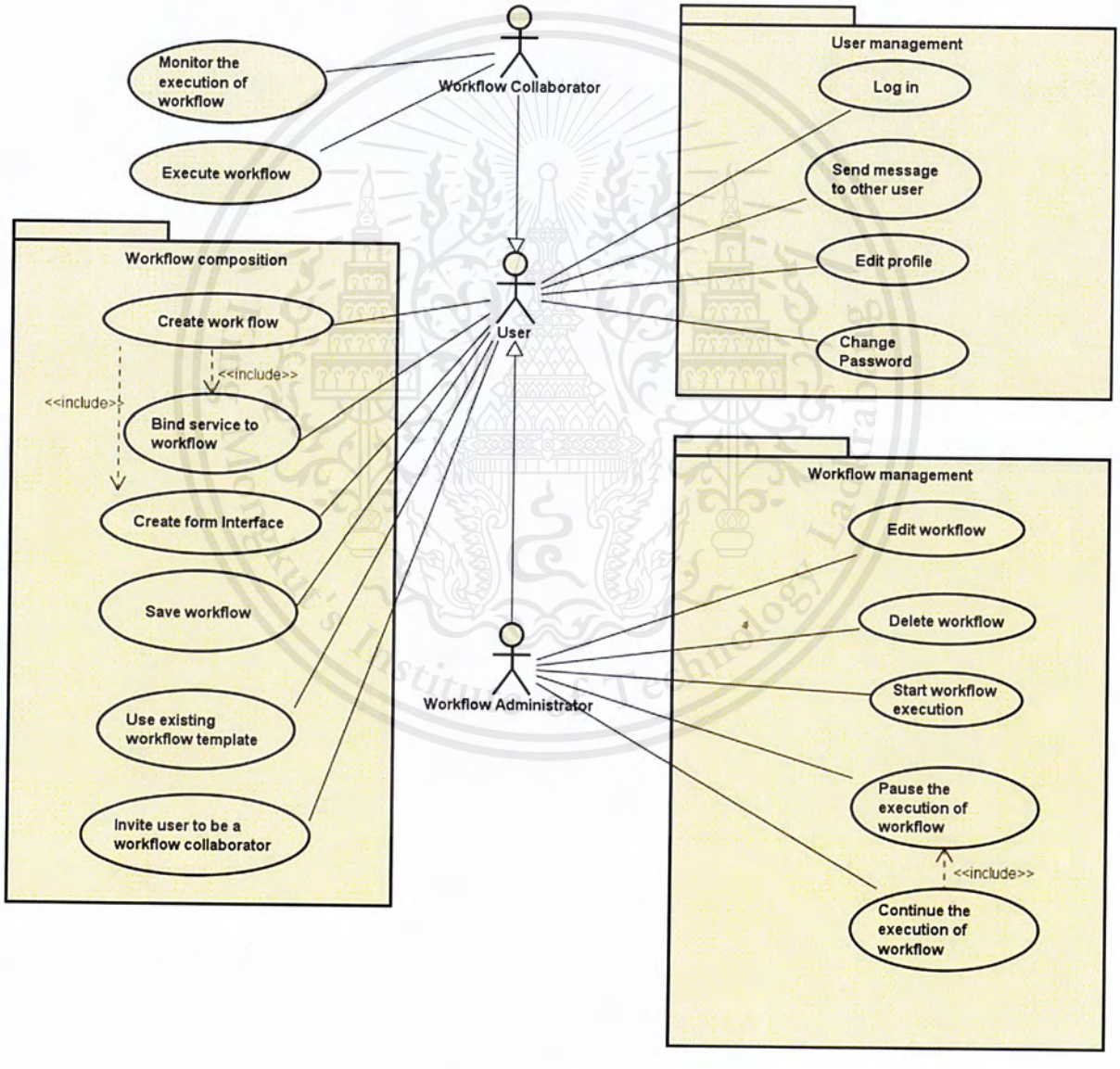


Figure A-1: Use Case Diagram

A.2 Use Case Description

Use Case	Create workflow
Primary Actor	User (Authorized user)
Main Success Scenario	A user selects to create a new workflow. The system lets the user compose a workflow diagram (BPMN) with workflow composer tools, then submit the created workflow diagram.
Alternate Scenarios 1	A user can choose to use existing business workflow diagram templates.
Exception scenario 1	A user draws the business workflow diagram (BPMN). The syntax errors in diagram occurred. The system notifies the user to reconstruct a diagram.

Use Case	Bind service to workflow.
Primary Actor	User (Authorized user)
Main Success Scenario	The user selects available service to bind service to task of workflow by using microservice binder.
Exception scenario 1	The system cannot connect to the selected service. The system does not allow user to add this service and notify to the user.

Use Case	Create form interface.
Primary Actor	User (Authorized user)
Main Success Scenario	After the user create a workflow diagram (BPMN) and bind service to task of workflow, the system allows the user to create web interface to interact with the task of workflow.

Use Case	Use existing workflow template
Primary Actor	User (Authorized user)
Main Success Scenario	A user selects to create a new flow using existing workflow template, instead of creates a new workflow. Workflow template provide a composed workflow to user to modify.

Use Case	Save workflow
Primary Actor	User (Authorized user)
Main Success Scenario	After the user create a workflow diagram (BPMN) with binding services and form interface, the system stores created workflow in data storage.

Use Case	Invite user to be a workflow collaborator.
Primary Actor	User (Authorized user)
Main Success Scenario	User invite another verified user to be a workflow collaborator. Workflow collaborators are assigned to each lane of the workflow.

Use Case	Log in
Primary Actor	User (Authorized user)
Main Success Scenario	The user fills in their username and password. Then, the server verifies information and notify to the user whether it succeeds or fails to log in.
Alternate Scenarios 1	Either the username or password or both is invalid. The system allows the user to fill in their username and password again.

Use Case	Edit profile.
Primary Actor	User (Authorized user)
Main Success Scenario	The user selects to edit their profile option. After editing their profile, the change of the profile is updated.

Use Case	Change password
Primary Actor	User (Authorized user)
Main Success Scenario	The user selects to reset password. The system allows the user to set a new password and send the verification to user registered email. After the user verifies the email, the system then updates a new password.
Alternate Scenarios 1	A new password is invalid. The system informs the user and allows the user to set up a new password again.

Use Case	Execute workflow
Primary Actor	Workflow collaborator (Authorized user)
Main Success Scenario	After the user visits the web application of the workflow, the user inputs information to the web application and submit to the server. And, the system sends the next form interface to user. The process is terminated when the system achieves end of the workflow.

Use Case	Monitor the execution workflow
Primary Actor	Workflow collaborator (Authorized user)
Main Success Scenario	The user selects workflow monitoring menu. The system shows log records of selected workflow and show the graphical workflow monitoring.

Use Case	Edit workflow
Primary Actor	Workflow administrator (Authorized user)
Main Success Scenario	An administrator or a workflow manager selects the edit workflow option. The system allows user to edit their created workflow.

Use Case	Delete Workflow
Primary Actor	Workflow administrator (Authorized user)
Main Success Scenario	The workflow administrator or a workflow manager selects to remove workflow application. The server deletes all the information of workflow from data storage.

Use Case	Start workflow execution
Primary Actor	Workflow administrator (Authorized user)
Main Success Scenario	The workflow administrator or a workflow manager enable workflow execution. The state of workflow changes to running.

Use Case	Pause workflow execution
Primary Actor	Workflow administrator (Authorized user)
Main Success Scenario	An administrator or a workflow manager disable workflow execution. The system blocks the execution of workflow. The state of workflow changes from running to waiting.

Use Case	Continue the execution of workflow
Primary Actor	Workflow administrator (Authorized user)
Main Success Scenario	An administrator or a workflow manager continue the execution of paused workflow. The state of workflow changes from waiting to running.