Business Process Modelling

A White Paper & Development Methodology
And
An Introduction to Flowcraft ®

Version 2.3

By Bani Dom

Content Contributor

	Name	Role	Description
1	Bani Dom	Solution Architect	Designed the solution.
2	Mohammad Ikwan	Python Coder	Coded the Python APIs.
3	Luluk Ratri Ayu	Image Developer including Branding, UI/UX.	Developed the relevant image for this paper, and product designer
4	Alia Ali	UI/UX	Visual designer
5	Nur Halimartul Saadiah	Database Manager	Created all the SQL DLL script and loaded test data.
6	Haziq Irfan	Python Coder	Coded the Python independent Engines.
7	Adham	UI/UX and API Integration	Coded in React
8	Muhammad Zufar	Quality Assurance	Test & Test

Copyright Notice

For any queries regarding this document, please contact the following:

Detail	Description
Project Name	SAFWA E-Claims Business Process
Chief Technology Officer	Bani Dom banidom@schinkelsgroup.com.my banidom@ezdom.com.my Mobile: +6010 256 9368
Address	Schinkels Technik Sdn Bhd 21-1 & 21-2, Jalan Nova K U5/K, Subang Bistari 40150 Shah Alam, Selangor, Malaysia technik@schinkelsgroup.com.my http://schinkelsgroup.com.my/ http://www.technik.schinkelsgroup.com.my Telephone: +603 5885 3967 Fax: +603 5883 3964

Contents

1. Update Log

	Date	Description	Contributor	Version
1	19/04/2023	Start	Bani Dom	1.0
2	20/04/2023	Adding more contents	Bani Dom	1.0
3	21/04/2003	Adding content to Proposed Coding Approach.	Bani Dom	1.1
4	21/04/2023	Copyright notice	Bani Dom	1.1.1
5	02/05/2023	More contents	Bani Dom	1.2
6	07/05/2023	Change of addressing of child list	Bani Dom	1.4
7	08/05/2023	Addressing payment action list	Bani Dom	1.5
8	09/05/2023	Changes in detecting ‘create-payment’ key	Bani Dom	1.6
9	17/05/2023	Modify list designation	Bani Dom	1.7
10	05/07/2023	Added content on future requirements	Bani Dom	1.8
11	06/07/2023	Content on Interpretive Engine	Bani Dom	1.8
12	06/07/2023	Adding content to Conclusion	Bani Dom	1.9
13	16/07/2023	Adding content to Appendix	Bani Dom	1.9
14	10/05/2024	Adding new content - everywhere	Bani Dom	2.1
15	10/05/2024	Working with multiple cycles in memory	Bani Dom	2.2
16	11/05/2024	Adding content to Monitoring section	Bani Dom	2.3

2. Synopsis

This paper describes the methodology defined for the SAFWA E-Claims business process digital automation. It describes the 3 key important elements of making the business automation framework:

- Database definition
- Class type definition
- Method definition for each class
- Integration with external function through defined APIs.

As of May 10th, 2024, this is currently a live paper. We have made progress in the development of the application, albeit incomplete, but it is usable with limited functionalities.

3. Introduction

The SAFWA E-Claims application development necessitates automating I was looking high and low in trying to find an architect solution but find none.the business processes involved in the claims process. There also several other requirements to satisfy the paymaster.

This solution went through a 2-phase program. The first phase gets to the database DDL stage but never did get to the API development. I found it a little cumbersome in the definition of each process Stage. It is also prone to error during the definition stage.

I tested several discussions on the business process methodology for the team. It got me more blank faces than quick understanding. I hope this documentation helps in getting the understanding much quicker.

This second stage is more natural and allows easy addition to the Business Process Stage development. It is also facilitated for the need to have multiple Business Process in the same database storage.

The business process is designed to cater for the following eventualities:

1. The business process is broken down into several tasks called stages.
2. The business process will change, and each change is reflective of a business process cycle.
3. There will be several conditions and requirements for each task, including staff process restriction.

In addition, the automation of operational processes is designed with the following requirements:

4. A single active business process cycle, but the change history must be recorded.

5. Each claim will only go through a single cycle business process. Mid-cycle change of business process will take the claims to the beginning of the cycle.
6. Validation of completion for each stage before proceeding to another
7. Some change of stage may require secure identifiable user validation.
8. A stage may have multiple previous sources and multiple targets.
9. For the targets of the same stage, the first evaluated targets that satisfy all requirements will be selected.
10. For each stage, if there the value of no requirement or no condition is equated to None and return value of evaluating this None is True. (We will go into detail of this requirement later)
11. The business model has only 1 stage at the beginning.
12. The business model has only 1 stage at the end.

4. Development Approach

The digital business model will first be stored in a SQL-enabled database. The tables as-is will be constructed on a faster memory cache database, like REDIS. It is from this database that the in-memory business process model to be re-created at each app initialization. Information from the in-memory model can be retrieved using APIs.

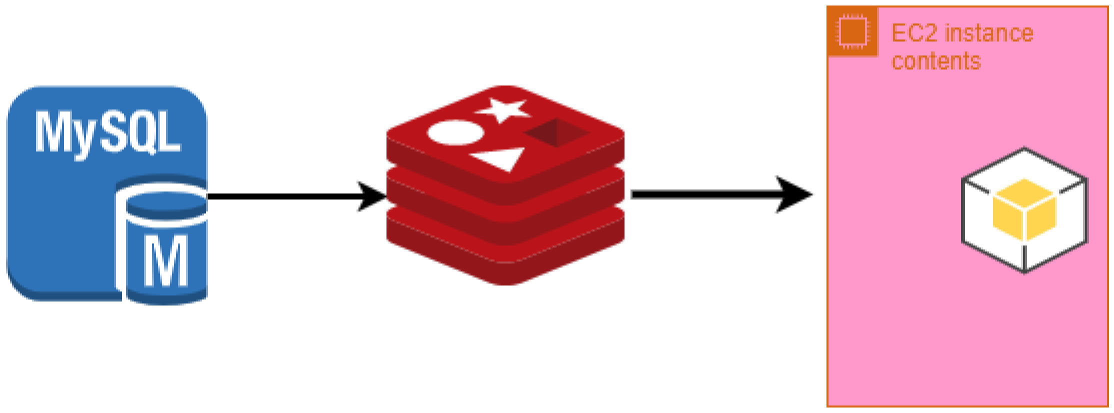


Figure 1: Building of Business Process

The following are scope of work for the digital modelling requirements:

	Description	Task Mode	Status
1.	Creation of the model dB structure	Direct	Done
2.	Insert of data to the DB structure	Direct	Done
3.	A migration app for dB structure from SQL-enabled database to REDIS	Independent Python App	Done
4.	Loading of Business process model in memory	Python app as part of APIs.	Done
5.	Develop API to validate process movement to NEXT stage	Python API.	Done
6.	Developed API to select NEXT stage	Python API	Done
7.	Developed API to move to NEXT stage	Python API	Done
8.	Develop APIs ft GET business process content	Python API	Done
9.	Develop test APIs	Python API	Ongoing
10.	Develop Model Management App	Independent Python App	Ongoing
11.	Develop a module to generate a graphical image of the model from REDIS dB.	React module	Done, but static representation
12.	Develop a brand for the Product	UI/UX	Flowcraft
13.	Develop GUI to manage stage content	React module & Python API	Done
14.	Develop stage content syntax validation engine	Python API	Done
15.	Develop stage content semantic validation engine	Python API	Always WIP
16.	Develop GUI to manage cycle	React module & Python API	At UI/UX

17.	Develop GUI to manage stage connectivity	React module & Python API	At UI/UX
18.	Integrating db model to graphical representation	React module & Python API	Future
19.	Graphical stage connectivity manipulation	React module, new graphics package & Python API	Future

As mentioned above, the delivery of these modules will be in python object-oriented programming. In Python, a class is a blueprint for creating objects with shared attributes and methods.

What is the methodology of object-oriented programming?

Object-oriented methodology is a way of viewing software components and their relationships. Object-oriented methodology relies on three characteristics that define object-oriented languages: encapsulation, polymorphism, and inheritance.

Object-oriented programming (OOP) is a method of structuring a program by bundling related properties and behaviors into individual objects. The objects in this case are:

1. Process Stage Object
2. Claim Object
3. Model Cycle Object

The detailed definition of the above shall be highlighted in the Python code.

5. Business Process

A business process is a set of activities or tasks that are performed by a company or organization to achieve a specific goal. It typically involves a series of steps or actions that are designed to create or deliver a product or service, manage resources, or achieve some other objective.

Business processes can be formal or informal, depending on the organization and the type of activity being performed. They may involve one or many different departments, teams, or individuals within an organization.

Examples of business processes include manufacturing processes, sales processes, customer service processes, human resources processes, and financial processes. These processes are often documented, analyzed, and optimized to improve efficiency, quality, and overall performance.

6. Business Process Mappings

Business Process Mapping details the steps that a business takes to complete a process, such as hiring an employee or ordering and shipping a product. They show the “who,” “what,” “when,” “where” and “how” to these steps and help to analyze the “why.” These maps are also called Business Process Diagrams and Business Flow Charts. Like other types of diagrams, these maps use defined symbols such as circles, rectangles, diamonds and arrows to depict the business activities.

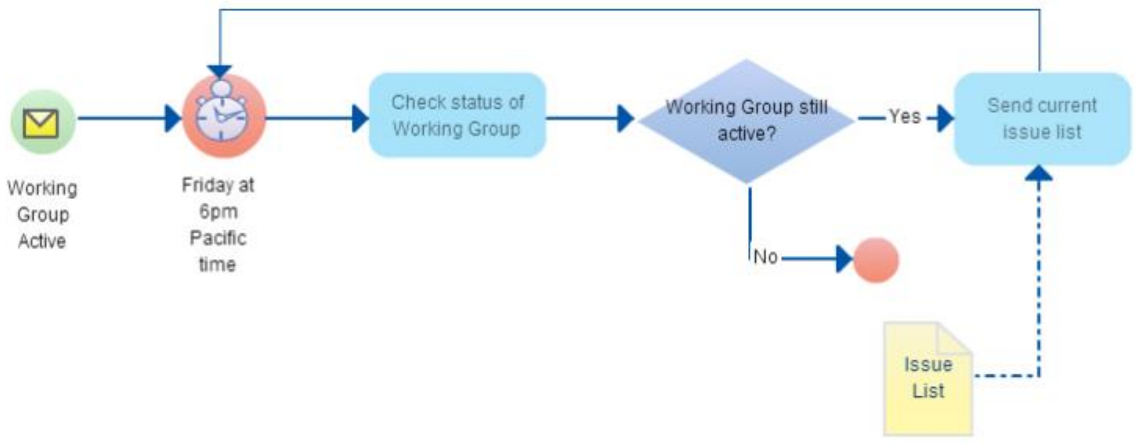

Working Group Active
Friday at 6pm Pacific time
Check status of Working Group
Working Group still active?
Yes
No
Send current issue list
Issue List

Figure 2: Am example of a Business Process Mappings

Business Process Mapping can be used to document a current process and to model a new one. Its purpose is to gain a detailed understanding of the process, people, inputs, controls and outputs, and then potentially to simplify it all, make it more efficient and/or improve the process results. It requires time and discipline to conduct this mapping, but the payoff can be significant over time. Mapping has become common in the business world to standardize procedures, become more efficient, meet audit requirements and gain competitive advantage.

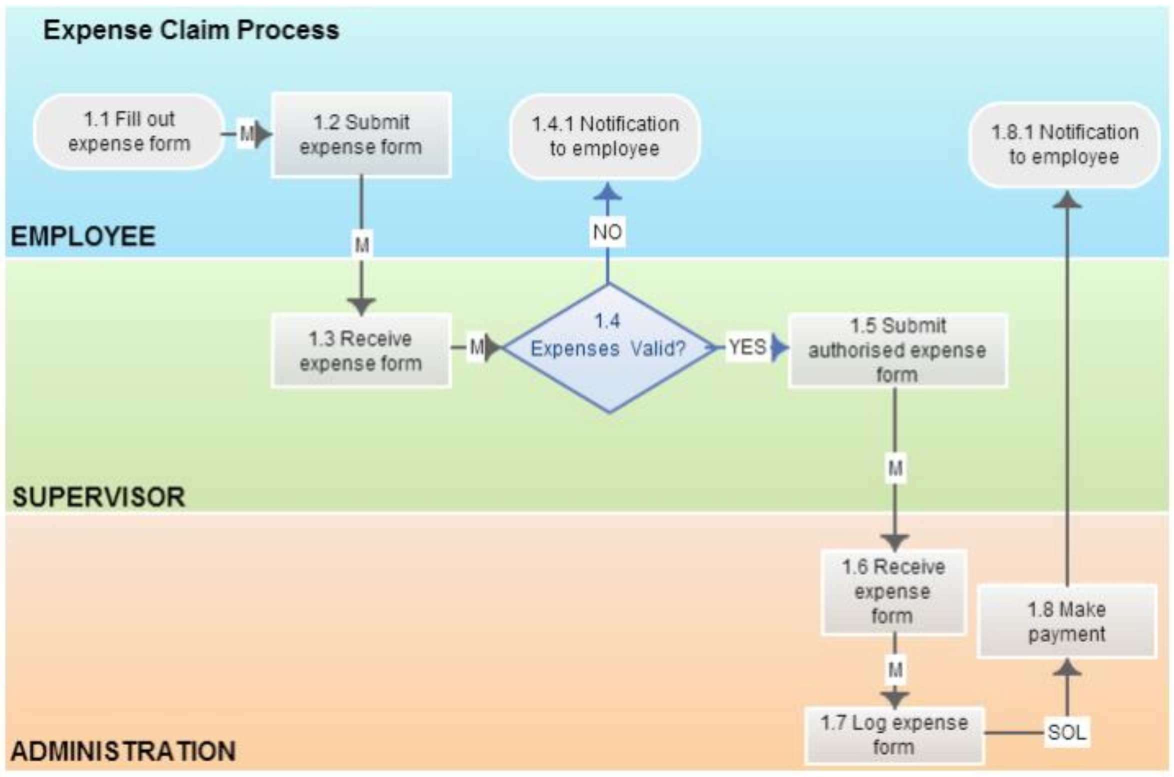


Figure 3: Another example of a Business Process Mappings

The M in the diagram above indicates Manual processes, while the A for Automation.

Business Process Mapping can be used to prepare for business audits or a sale, to reduce expenses, to plan for automation, to understand the impacts of pending changes, to realign related processes, and to measure and realign the efforts of people involved in the processes. Often, a business may think it understands its processes, but then discovers twists and turns during a mapping initiative. When modeling a new business process, the mapping is sometimes called Business Process Modeling, or BPM.

7. Business Process Modeling Notation (BPMN)

Simply put, BPMN is a graphical representation of your business process using standard objects. If you want to get more technical It can also be defined as a set of graphical objects and rules defining available connections between the objects.

BPMN consists of the following basic building blocks;

- Flow objects: events (circles), activities (rectangles with rounded corners), and gateways (diamonds)

- Connecting objects: mainly comprising arrows, these indicate sequence flow (filled arrows), the message flow (dashed arrows), and associations
- Swim lanes: pools (graphic container) and lanes (sub-partition of the pool)
- Artifacts: data objects, groups, and annotations

Flow Objects

A BPMN has a small set of (three) core elements, which are the Flow Objects so that modelers do not have to learn and recognize a large number of different shapes. The three Flow Objects are:

Event

An Event is represented by a circle and is something that "happens" during the course of a business process. These Events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of Events, based on when they affect the flow: Start, Intermediate, and End (see the figures to the right, respectively).

Activity

An Activity is represented by a rounded-corner rectangle (see the figure to the right) and is a generic term for work that the company performs. An Activity can be atomic or nonatomic (compound). The types of Activities are Task and Sub-Process. The Sub-Process is distinguished by a small plus sign in the bottom center of the shape.

Gateway

A Gateway is represented by the familiar diamond shape and is used to control the divergence and convergence of Sequence Flow. Thus, it will determine traditional decisions, as well as the forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control.

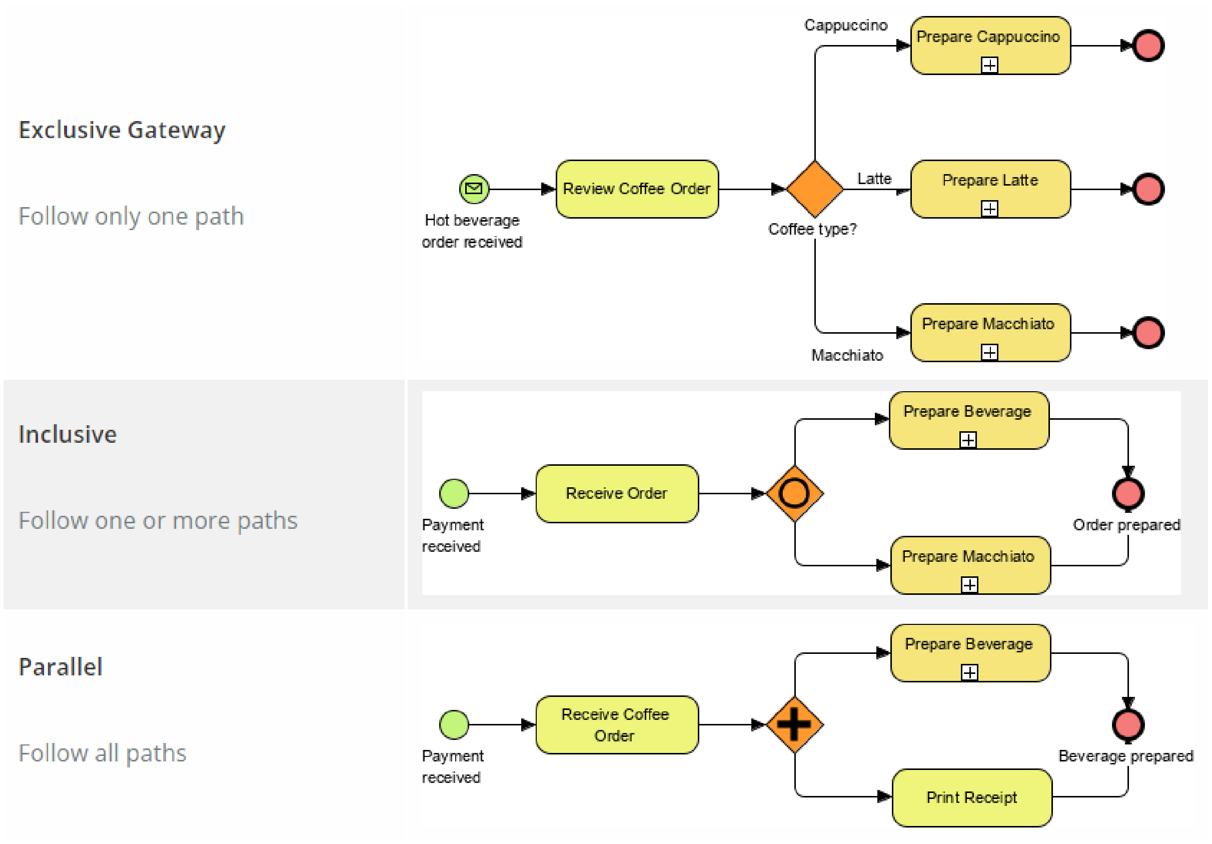


Figure 4: Uses of BPMN Building Blocks

Connecting Objects

Connecting objects illustrate how different pieces of the puzzle connect with one another. There are four types of connecting objects: sequence flows, message flows, and associations. They're represented by arrows.

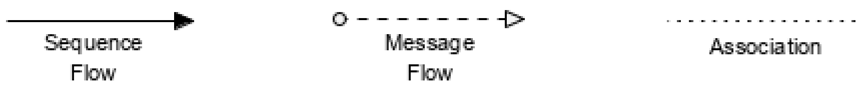


Figure 5: Illustration of connecting objects.

Sequence: Maps the sequential flow of objects.

Association: Depicts the relationship between different data and objects. Annotations allow additional information relevant in documenting the process to be shown on the diagram.

Message: Indicates a message sent between various participants in the workflow. A message Flow symbolizes the information flow across organizational boundaries. Message flow can be attached to pools, activities, or message events. The Message Flow can be decorated with an envelope depicting the content of the message.

Swimlanes

Swimlanes are rectangular boxes in BPMN that represent the participants of a lane's participant business process. Swimlanes can contain flow objects that are performed by that, except for the black boxes. Swimlanes can be arranged horizontally or vertically, and they are semantically the same but just different in representation. For horizontal swimlanes, the process flows from left to right, while in vertical swimlanes, the process flows from top to bottom. Examples of swimlanes include the customer, account department, payment gateway, and development team.

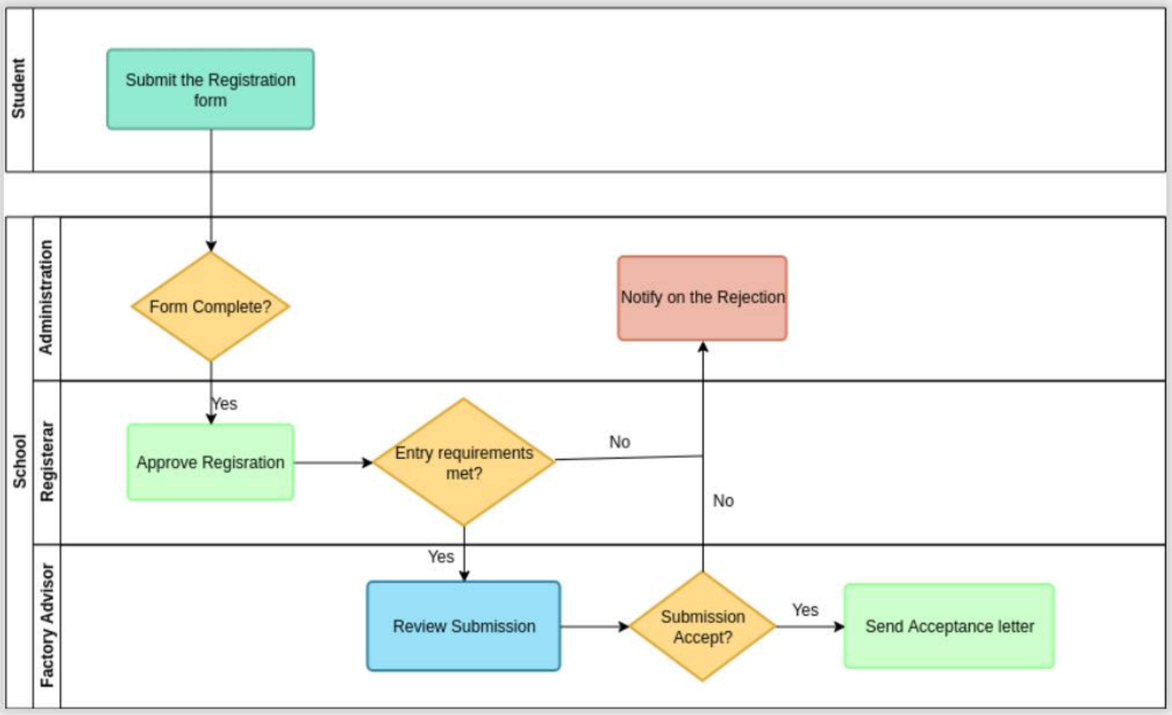


Figure 6: BPMN examples with swinlanes

8. Database Definition Approach

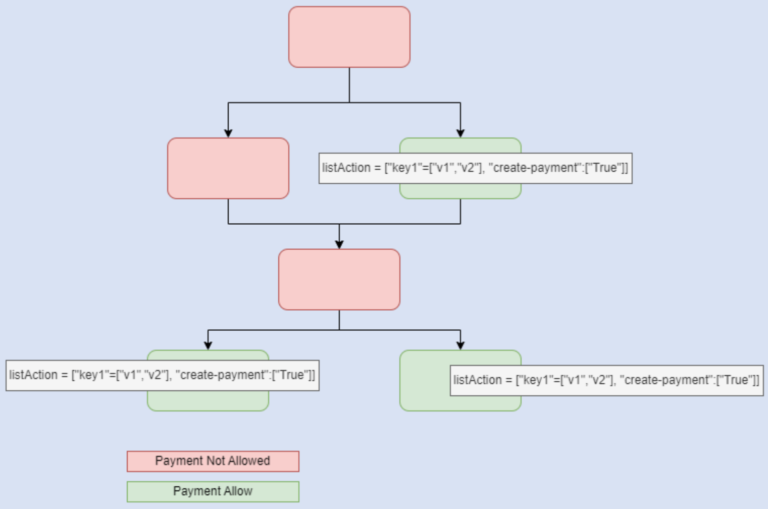
Following are table definitions require for the business process management:

1. tbl_claims_process
2. tbl_claims_stage
3. tbl_ref_process_cycle

```
CREATE TABLE dangabay.tbl_claims_process (
  uuid varchar(36) NOT NULL DEFAULT (UUID()),
  process_stage_name varchar(256) NOT NULL,
  list_entry_condition json NOT NULL,
  List_exit_condition_json NOT NULL,
  list_action json NOT NULL,
  list_requirement json NOT NULL,
  list_doc_json NOT NULL,
  list_user json NOT NULL,
  list_roles json NOT NULL,
  list_pbt json NOT NULL,
  list_pacate json NOT NULL,
  cycle_uuid varchar(36) NOT NULL,
  created_datetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_datetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (uuid)
)
ENGINE = INNODB,
AVG_ROW_LENGTH = 2340,
CHARACTER SET latin1,
COLLATE latin1_swedish_ci;
```

The content of each field are as follows:

	Field Name	Content Value
1	uuid	This process uuid.
2	process_stage_name	The process readable name. A name that is readable instead of a uuid.

3	list_entry_condition_json	List of condition to move to this process. This list contains all the requirement needed to move to this stage. Sample value can be *[‘category’:[‘routine’, ‘emergency’], ‘pbt’:[‘1’,’2’]]*
4	list_exit_condition_json	List of exit conditions to go to the next stage. This list contains all the exit condition action for this stage. This may include actions of the type ‘Approval’, ‘Check’, or ‘Verify’. AS part of our definition notation it can contain more than one.
4	list_action_json	List of action required in this process stage. This list contains actions to do while in this stage. This is used as a guide to the user. A request can be made through an API and the response will be what is in the list. This list has a unique key – “create-payment” = [“True”]. This action shall indicate that payment can be made at this current stage, even from a different leg. The following describes stages that allow payment.  listAction = ["key1"=["v1","v2"], "create-payment":["True"]] listAction = ["key1"=["v1","v2"], "create-payment":["True"]] listAction = ["key1"=["v1","v2"], "create-payment":["True"]] Payment Not Allowed Payment Allow **Figure 7: Applying action key "create-payment".**

To eliminate any inaccuracy, it is recommended to have a common action stage for payment like below.

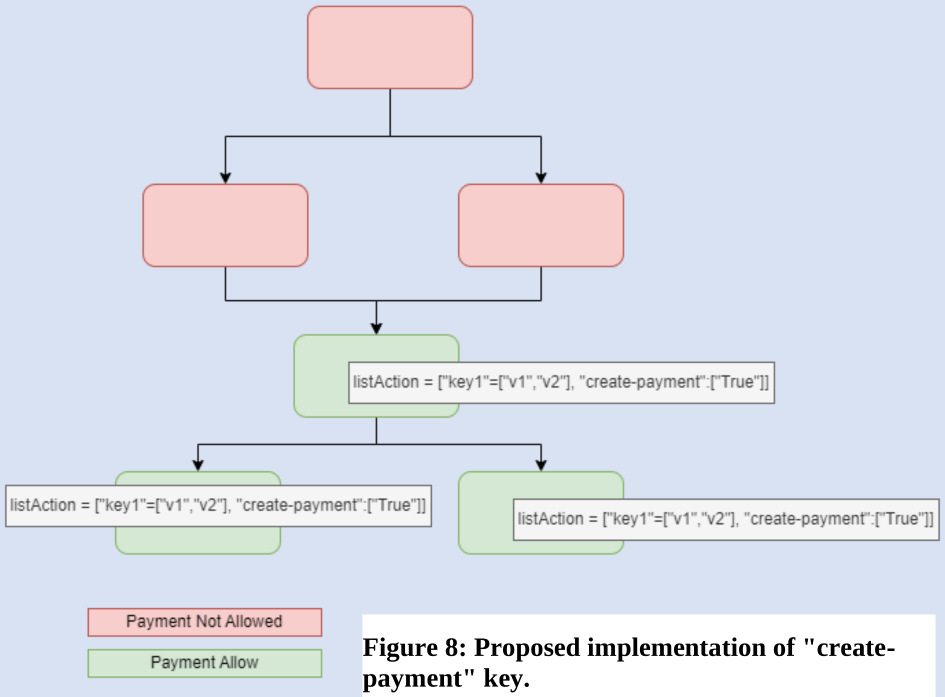
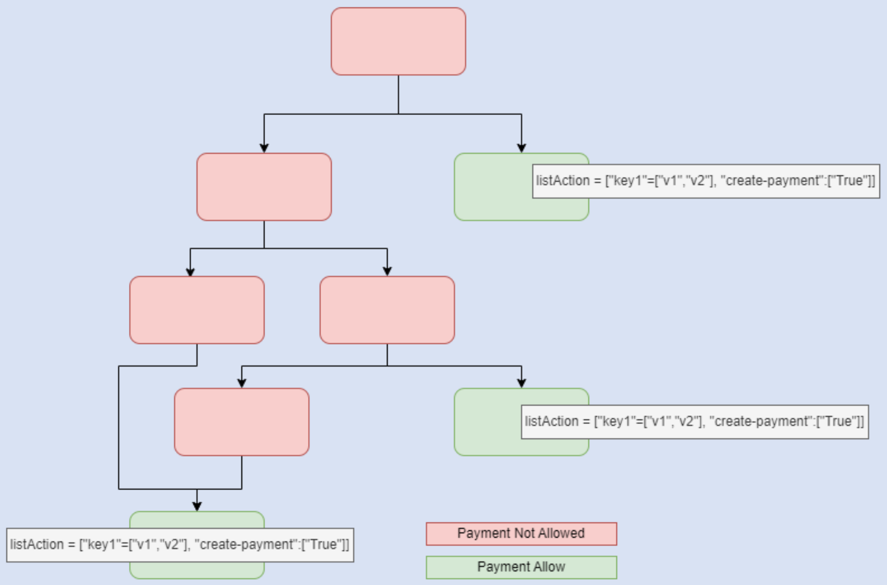

listAction = ["key1"=["v1","v2"], "create-payment":["True"]]
listAction = ["key1"=["v1","v2"], "create-payment":["True"]]
listAction = ["key1"=["v1","v2"], "create-payment":["True"]]
Payment Not Allowed
Payment Allow

Figure 8: Proposed implementation of "create-payment" key.

The strategy we use above allows better control of the business process for specific action. In this case the action key "create-payment".

		 listAction = ["key1"=["v1","v2"], "create-payment":["True"]] listAction = ["key1"=["v1","v2"], "create-payment":["True"]] listAction = ["key1"=["v1","v2"], "create-payment":["True"]] Payment Not Allowed Payment Allow **Figure 9: Better control of action key.**
5	list_requirement_json	List of content beside document needed for this stage. This list contains what needs to be done before the exit. Validation of the [Net] button is done against this list.
6	list_doc_json	List of documents to upload in this process stage. The value should be document type as provided in the lookup table.
7	list_user_json	List of users that can edit in this process stage. List of users than view, edit or add to this claim. This is for verification of credentials.
8	list_roles_json	List of user's roles that can edit/move this process stage. List of user's roles than can view, edit or add to this claim. This is for verification of credentials.
9	list_pbt_json	List of pbt for this stage. We added this for later usage. Only specify pbt can view this claim.

10	list_pacate_json	List of package category. Not used for now.
11	cycle_uuid	The cycle of this process. This refers to the cycle’s uuid in the cycle reference table.

The following are definition and uses of the table structure above:

	Description
1	Null list is defined as []
2	Content of list is always JSON format, and it is a dictionary, and the value of each key is a list. This is to cater for multiple possibilities of “key” value. For example: *thisdict = { “brand”: [“Ford”], “model”: [“Mustang”, “Rover”], “year”: [“1964”], “century”: [] }* *Or the value in the list can be another dict and looks like below.* *thisdict = { “brand”: [{“truck”: [“Ford”, “DMAX”]}], “model”: [“Mustang”, “Rover”], “year”: [“1964”], “century”: []*
3	Each list has its own “key” vocabulary, and this vocabulary defined the action on the key’s value.
4.	List of exit condition are actions required in this process before moving to the next stage. It may be any one of the following: 1. Checked 2. Approved

	It can be represented as *{‘Checked’: [{‘User’: [‘Name1’, ’Name2’]}, {‘Role’: []}],* *‘Approved’: [{’Role’: []}, {‘User’: [‘Name1’,’Name2’]}]* *}* *In BPMN, the gateways are embedded in each process as the list of exit condition.*
5	As stated earlier the empty list [] validation always returned True.
6	The key’s value “*” implies for all value. For example: *{‘Checked’: [{‘User’: [‘*’]}]}* implies that the Checked key can be done by any or all users.
7	The key’s value “None” implies not for any value. For example: *{‘Checked’: [{‘User’: [‘None’]}]}* implies that the Checked key cannot be done by any or all users.
8	Each processing of a value in a list return either True or False. For example for the processing of dict {‘key1’: [‘value1’, ‘value2’], ‘key2’:[‘value1’]}, ‘key1’ must returned a True value and ‘key2’ also for the dict to returned a True value. This is why {‘key’:[]} and {} returns True. Processing the full list within a ‘key’ is with an OR operator. Processing of the ‘action key’ is with an AND operator. For example: List_x = *{‘key1’: [‘value1’, ‘value2’], ‘key2’:[‘value1’]}* 1. *‘key1’* and *‘key2’* operator is AND. 2. *‘value1’* and *‘value2’* operator is OR. 3. The value *‘key1’* can appear multiples times for and AND operation.
9	Key ‘SQL’must contain a SQL stamen that return a Boolean value. For example, {‘sql’: [‘select count(*) from tbl1’]} returns 0 for False and count(*)>0 as True. The key ‘sql’ should not be in a list.

10	The list of entry conditions for each stage is in fact the selection criterion for proceeding to this stage. Without this, there is no possible to identify which stage to go to in the case of multiple child targets.

Line	**Code**
1	*CREATE TABLE dangabay.tbl_claims_stage (*
2	*uuid varchar(36) NOT NULL DEFAULT (UUID()),*
3	*prev_stage_uuid varchar(36) DEFAULT NULL, (now a JSON list)*
	curr_stage_uuid varchar(36) DEFAULT NULL,
4	*next_stage_uuid varchar(36) DEFAULT NULL, (now a JSON list)*
5	*cycle_uuid varchar(36) DEFAULT NULL,*
6	*created_datetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,*
7	*updated_datetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,*
8	*PRIMARY KEY (uuid)*
9	*)*
10	*ENGINE = INNODB,*
11	*AVG_ROW_LENGTH = 2340,*
12	*CHARACTER SET latin1,*
13	*COLLATE latin1_swedish_ci;*

The content of each field are as follows:

	Field Name	Content Value
1	uuid	uuid
2	prev_stage_uuid	The previous stage uuid. This is change to a list that contains all parent stage-id.
3	curr_stage_uuid	The current stage uuid
3	next_stage_uuid	Next stage uuid. This is change to alist that contains all children stage-id.
4	cycle_uuid	Cycle reference

The above table states the link of each process with the child process. This is the connectivity of each process with the previous stage and the next stage. It is possible to have multiple previous stages and naturally multiple next stages.

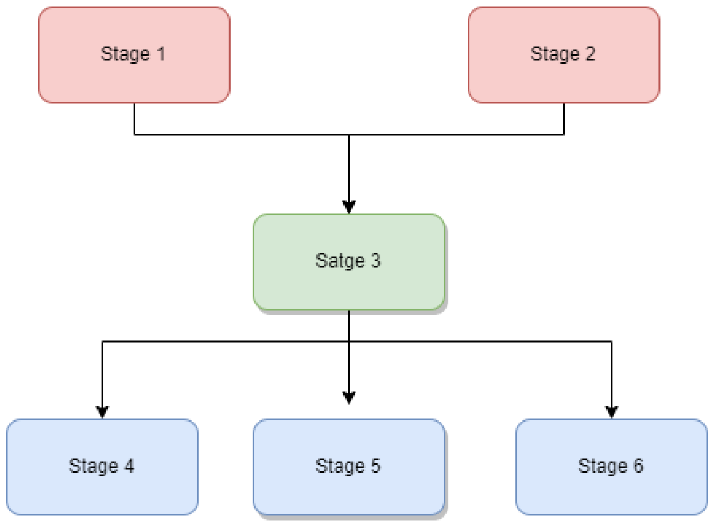


Figure 10: Previous and Next Stage Model

In the diagram above the previous stage of Stage 3 is Stage 1 and Stage 2. The next stage of Stage 3 is Stage 4, 5 and 6. This must be represented accordingly in the DB structure. Let's examine how the *dangabay.tbl_claims_stage* will be populated.

uuid	*prev_stage_uuid*	*curr_stage_uuid*	*next_stage_uuid*
1	*null*	*Stage1-uuid*	*Stage3-uuid*
2	*null*	*Stage2-uuid*	*Stage3-uuid*
3	*Stage1-uuid*	*Stage3-uuid*	*Multi*
4	*Stage2-uuid*	*Stage3-uuid*	*Multi*
5	*Stage3-uuid*	*Stage4-uuid*	*null*
6	*Stage3-uuid*	*Stage5-uuid*	*null*
7	*Stage3-uuid*	*Stage6-uuid*	*null*

Null in the prev_stage_uuid indicates at the top of the business process, and null at he next_stage_uuid indicates at the bottom or the end of the business process.

The term 'Multi' is only used for the next stage if the current stage has multiple next stages. Previous stage needs to identify all previous stages if it is multiple previous stages as stated by 'Multi' in the next_stage_uuid. The value of the next stage for Multi is determined by finding the value in the current stage for prev_stage_uuid = curr_stage_uuid.

Note: The term ‘Multi’ is used ONLY in the next_stage_uuid.

Line	Code
1	CREATE TABLE dangabay.tbl_ref_process_cycle (
2	uuid varchar(36) NOT NULL DEFAULT (UUID()),
3	cycle int NOT NULL,
4	active tinyint(1) NOT NULL,
5	descriptions varchar(150) DEFAULT NULL,
6	created_datetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
7	updated_datetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
8	PRIMARY KEY (uuid)
9	)
10	ENGINE = INNODB,
11	AVG_ROW_LENGTH = 16384,
12	CHARACTER SET latin1,
13	COLLATE latin1_swedish_ci;

The content of each field are as follows:

	Field Name	Content Value
1	uuid	uuid
2	cycle	Reference cycle id
3	active	Boolean value. Only one active cycle in the table.
4	description	Description for this cycle

The above table is a reference table for the cycle value. A cycle is an integer that is incremented each new cycle is added. It is important that for each cycle, a full process model is replicated in the database.

9. Class Definition Approach

As illustrated in the process definition above, we must simulate the class definition to accommodate the BPMN approach.

The following are class requirements.

	Class Name	Description
1	classStage	For each stage
2	classClaim	For every processed claim
3	classModel	For the model

(To be added)

10. Proposed Coding Approach

The proposed coding approach is valid during the initial development stage. As always, the actual approach and modules requirements change during the actual development time. However, this proposed approach remains the key methodology in building components using oop.

Modules Requirements

Following are module requirements:

	Description	Task Mode
1	Migration app for dB structure from sql-enabled database to REDIS	Independent Python App
2	Loading of Business process model on memory	Python app as part of APIs.
3	Develop API to validate process movement to NEXT stage. This requires the need for an API to locate the current process in the model.	Python API.
4	Developed API to select NEXT stage	Python API

5	Developed API to select CURRENT stage	Python API
6	Developed API to move to NEXT stage	Python API
7	Develop APIs to GET business process content	Python API
8	Develop test APIs	Python API
9	Develop Model Management App	Independent Python App

1. The Python independent app should only move the same dB structure into the JSON format REDIS content into the same dB(x) of the reference table.
2. The python app and it must be as part of the APIs will load the REDIS JSON structure into memory model. It will create objects of Process class with all the list appended with the details of the DB fields.
    a. There is one additional list of called *list_next_process* where this is appended with the child processes. This is modelled from the table *dangabay. tbl_claims_stage.*
    b. There is one additional list of called *list_previous_process* where this is appended with the child processes. This is modelled from the table *dangabay. tbl_claims_stage.*
    c. The first process has the *list_previous_process* of value [], while the last process has the *list_next_process* value [].

3. This API is ONLY to validate that the process is all good to go to the next stage. There returned of this API are error number, error message, and what to do next based on the exit condition. An approval may be needed and the returned to app must obtain the code when necessary.

    Before the validation begins, there is also a need to find the correct process stage. This is done by traversing the model from beginning to end. A 'None' value is returned if no stage is not found, or the stage object is returned. Validation is then done on that stage.

4. This API selects the next correct stage to move forward stage if any available.
5. This API will locate the current stage of the claims from the model.
6. This API moved the stage to the next stage forward. Testing is done by API above. It just moves from one stage to another.
7. More APIs are needed to peek at the model behavior.
8. These are Test and Quality Assurance APIs. It is most needed when a new cycle is implemented to assure completeness and accuracy of the business model.

9. This is for the future where a user can manage the business model of every cycle. New models can also be added using a defined format. The new business process can be loaded into the database and the REDIS memory DB cache.

I have not defined the parameters of each API function here, as the actual need should be clearer during the coding activities.

There are also additional activities that must be done during the processing of the following APIs.

1. Updating log
2. Updating timeline
3. Updating audit trail
4. Updating status of claims instance.

The above operation is not done by the Python APIs. It must be done from the GUI side or server activated from the GUI side.

What is in the GLOBAL environment?

The following are lists of variables and constants in the GLOBAL scope. The name indicated is just for reference. It is by no means set in stone. The list below is neither complete nor accurate. It is meant as representation of part of the variables in the GLOBAL environment scope.

	Description	Task Mode
1	listBizModelCycle	List of loaded business model. Currently it should be only 1 item in this list. This is in preparation of loading multiple business models on memory. The item contains the instant of the first process stage of the model.
2	instCurrStartStage	This points to the current Start stage.
3	instCurrLastStage	This points to the current Last stage.
4	listProcessClaim	List of claims that are being processed. This is a self-disposed list. The idea is to only allow a certain time limit for each claim object to exist for any validation before it is disposed from the list. In a multiuser

		environment value in the dB may change while waiting for next user input. Disposal timing uses the concept of ttl – time to live.

11. Integration Use Case

The following are use scenarios, and the algorithm presented is an example of a longer methodology. It is neither complete nor accurate. There should be far more subtleties in the final enduring algorithm. My objective is to initiate a possible thinking process for the described use cases.

What happens when the <NEXT> button is clicked.

The GUI must validate that the GUI content is saved. Claims info pulls from dB and not from the GUI content.

```
If GUI content is not saved
    Request for Saved user input.
    If Yes
        Saved the GUI content
        Proceed to processing of <Next>
    Else
        Explain why the <Next> is aborted
```

The processing of <Next> can be as follows. This is from React function.

```
Call API to initiate validate Process Movement
If Require Actions
    Get Action Result
    If Action Result is OK to moved
        Call API to move to next stage
    Else
        Explain why the <Next> is aborted
Else
    Call API to move to next stage
```

The processing of Get Action Result uses the queue, and the action can be done thru the consumer function of the queue.

Algorithm to initiate validate Process Movement. This is in Python and a method of the classModel

```
Create instClaims
Pull data from dB from claims (if necessary)
Append instClaims into listProcesClaims and set ttl
Call the API to find the Current Stage of the claims in the model
If Found
    Call API to validate the claims
Else
    Explain why Stage is not Found in the model
```

Algorithm to validate Process Movement. This is in Python and a method of the classStage.

12. Interpretive Engine

An interpreter produces a result from a program. The interpreter architecture then turns the resulting program into a process code.

In our case, we called the interpreter to evaluate keywords in the list of lists as Interpretive Engine – IE. IE will then process the evaluation of the keyword.

There are two different IEs in the module. The first is called IE1 and the second is called IE2.

IE1 produces True and False output with error number if there are any. Recipients need to call an API for the full error message. IT evaluates the logical truth of the list of assignment in various list. IE1 is a recursive module that extracts logical value. The process is done with an AND operative for each keyword in a given list, while the OR operative is apply each value in each keywork.

IE2, produces output in the body of the response with a logical value as True and False for success status for further action or actions.

IE never do any action. IE only evaluates the keyword and produces a logical value or a list of results. Recipient module must act on the result. IE also does not generate UI interface. Again, it is the responsibility of all recipient modules.

13. Syntax Validation Engine

Syntax validation is the process of checking whether the syntax of a given piece of code conforms to the rules and structure defined by the programming language. Every programming language has its own set of syntax rules, which dictate how statements, expressions, and other code constructs should be written.

Here's how it typically works:

1. Parsing: The code is analyzed to identify its structure. This involves breaking down the code into its constituent parts, such as keywords, operators, identifiers, etc.
2. Rule Check: Each part of the code is checked against the syntax rules of the programming language. This involves verifying that the code adheres to rules such as correct order of keywords, proper use of punctuation, and adherence to language-specific conventions.
3. Error Detection: If any part of the code violates the syntax rules, errors are flagged. These errors may include missing semicolons, mismatched parentheses, incorrect indentation, or the misuse of keywords.
4. Feedback: Feedback is provided to the developer about the syntax errors found in the code, typically in the form of error messages or warnings. These messages help developers identify and correct the errors in their code.

Syntax validation is a fundamental step in the software development process, as it ensures that the code is written correctly and can be understood by the compiler or interpreter of the programming language. It helps prevent common programming errors and ensures that the code functions as intended.

The following is the list of keywords we used and their purposes. This is valid as of May 10th, 2024.

Keywords	Role
*****	return true for all cases,
@	user can next stage claim that in their pending
verifyrole_by	verify claim to next stage by roles,
verifyuser_by	verify claim to next stage by users,
approverole_by	approve claim to next stage by roles,
approveuser_by	approve claim to next stage by users,
checkuser_by	checkuser_by - check claim to next stage by users,
checkrole_by	checkrole_by - check claim to next stage by roles,
certifyuser_by	certifyuser_by - ceritfy claim to next stage by user,
certifyrole_by	certifyrole_by - ceritfy claim to next stage by roles,

chk_doc	must complete all document before next stage,
cate_id	entry stage based on cate id,
pbt_id	entry stage based on pbt id,
sql	check status claim using sql statement,
create_payment	stage can make payment or not,
payment_user	make payment by users,
payment_roles	make payment by roles,
user_id	check claim by user id,
upload_doc	must upload all document before next stage,
approve_payment	stage can approve payment or not,
revert	revert - stage can revert or not,
insert_cop_date	stage can insert cop date or not,
can_view	permission to view claim,
can_edit	permission to edit claim,
generate_sfa	stage can generate sfa or not,
certified	stage can certified claim or not,
can_view_exception	permission view exception by roles,
can_view_exclusion	permission view exclusion by roles,
notify_role	list of roles to notify,
revert_stage	stage can revert or not,
revert_mode	revert origin or revert one stage,
select_role	select user to notify and do works in next stage,
prev_revert	revert to prev stage based on timeline
check_roles	check by roles
verify_roles	verify by roles
approve_roles	approve by roles
test_view	for testing purpose
revert_origin	revert to the prev stage based on timeline
percentage	calculate the percentage of completed the claim
delete_attachment_role	delete attachment based on role

14. Semantic Validation Engine

Semantic checking is a crucial stage in the compilation process where the meaning or semantics of a program is analyzed. While syntax validation focuses on the structure and grammar of the code, semantic checking delves deeper into understanding the intended logic and behavior of the code.

Here's how semantic checking typically works:

1. Type Checking: One of the primary tasks of semantic checking is to ensure that the types of operands in expressions and statements are compatible with the operations being performed on them. For example, adding a string to an integer might not be allowed in some programming languages.
2. Scope Analysis: Semantic checking verifies that variables and other identifiers are declared and used within the correct scope. This includes checking for redeclaration of variables within the same scope and ensuring that variables are initialized before use.
3. Function/Method Resolution: Semantic checking verifies that functions or methods called in the code exist and are being used correctly, including checking the number and types of parameters passed to them.
4. Control Flow Analysis: This involves analyzing the flow of control within the program to ensure that it follows valid paths and that constructs like loops, conditionals, and branches are used correctly.
5. Error Detection and Reporting: If semantic errors are detected, such as type mismatches or undeclared variables, appropriate error messages are generated to help the developer identify and fix the issues.

Semantic checking goes beyond syntax validation to ensure that the code not only follows the rules of the programming language but also behaves as intended by the programmer. It helps catch logical errors and inconsistencies that might not be apparent from the code's structure alone, ultimately improving the reliability and correctness of the software.

We divide the process of building the semantic engine into stages.

Stage	Description	Status
1	String validation where needed. 1. A stage name must have a valid string value. 2. A stage name must be unique in the cycle.	Done

2	Content validity against database data. For example, if a userid is checked against its existing ion the record.	WIP
3	Verification of permission for set Actor (userid).	Future
4	Detection of continuous Checked and Verified by same Actor. This is to protect legality.	Future
5	Detection of Reversing Stage. Stage that allows a never-ending business process.	Future.
6	Use of * and @ wrongly for example for Verify role.	Future
7	Detection of use role as invalid role.	Future

Semantic checking returns 3 results with error messages.

1. The first is Failed or False.
2. The second is Passed or True for current semantic checking. Do take note that not all semantic checking will be done. This is an ongoing development work.
3. The third is Conditional Passed with the user having option to continue or stop.

15. Implementation & Monitoring Tools

It is necessary to always monitor the operation of the business process engine. There are 2 sets of instances to have real-time monitoring. The first to create a model for the processing of business cycles in memory and its associated claims in action. The second task is to monitor the editing activity for business process cycles.

The Business Cycles are used in the following ways. Business Cycles are stored in the database, and then reload to the REDIS elastic cache for access speed and then loaded into memory.

Let's divide the two business processes into two modes: Runtime and Editing Mode. While the Runtime mode is setup during the initial build and Editing Mode is setup during each editing session.

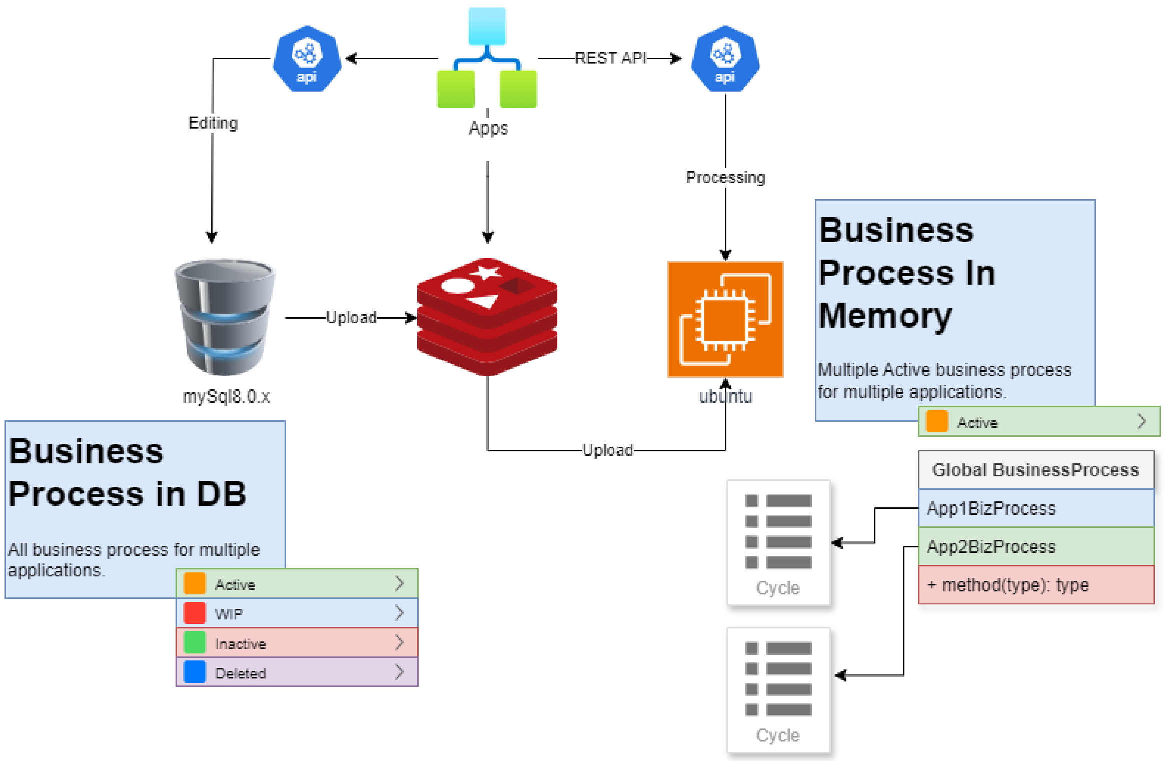


Figure 11: Implementation Model

The monitoring tools that have been suggested are as follows:

	Purpose	Status
1	Instances of Cycles at Runtime Mode.	
2	Activities of claims on each cycle at Runtime Mode	
3	Editing Mode status	
4	Activities of Editing Mode	
5	Startup of log trail for Runtime and Editing Modes	

16. Modification offstage in an WIP Cycle

Action taken on any stage in a cycle is limited to protect the integrity of the business process. Each business process has a set of related claims attached to it. Each cycle shall be in any of the following state: Active, Inactive, WIP, and Deleted.

	Action	Allowed when Cycle is in the State of
1	Edit	WIP
2	Duplicate	Active Inactive Deleted Once duplicated the status of the newly will be in WIP state.
3	Delete	Inactive WIP
4	Load Into REDIS	Active Only
5	Deactivate to Inactive	Active Only
6	Activate to Active	WIP Inactive
7	Pull and ReSynch claims from another Cycle	WIP Only The proposed mechanism to resynchronize from another cycle is to align the pulled claims to the top of the cycle. This also require to set new pendings status and timeline status to each claims.

It is crucial to note that if a cycle is transferred from a WIP state to another state, it cannot be returned to the same WIP state. Only the WIP state can pull stages from one cycle.

17. Changing the Cycle Stage Connectivity

We think that a detailed description is needed for any alteration to the connectivity of the cycle stages.

A few functionalities requirements is needed during the process of modifying connectivity of stages within a Cycle.

1. A visual interpretation of current work-in-progress connectivity.

2. A work area to do the following connectivity adjustment function:
    a. Break or Disjoint – Break the connectivity of a stage.
    b. Add – Add a new Stage to the Cycle.
    c. Moved or Arrange – Moved a Stage to a new connectivity.
    d. Delete – Delete a Stage from the Cycle.
    e. Duplicate – Making copies of one available Stage to connect to another.
3. The above functionalities do not replicate Stage's content. Stage content editing functionalities shall be done at another component of the system.
4. A proposed interpretation of the UI can be presented as follows:

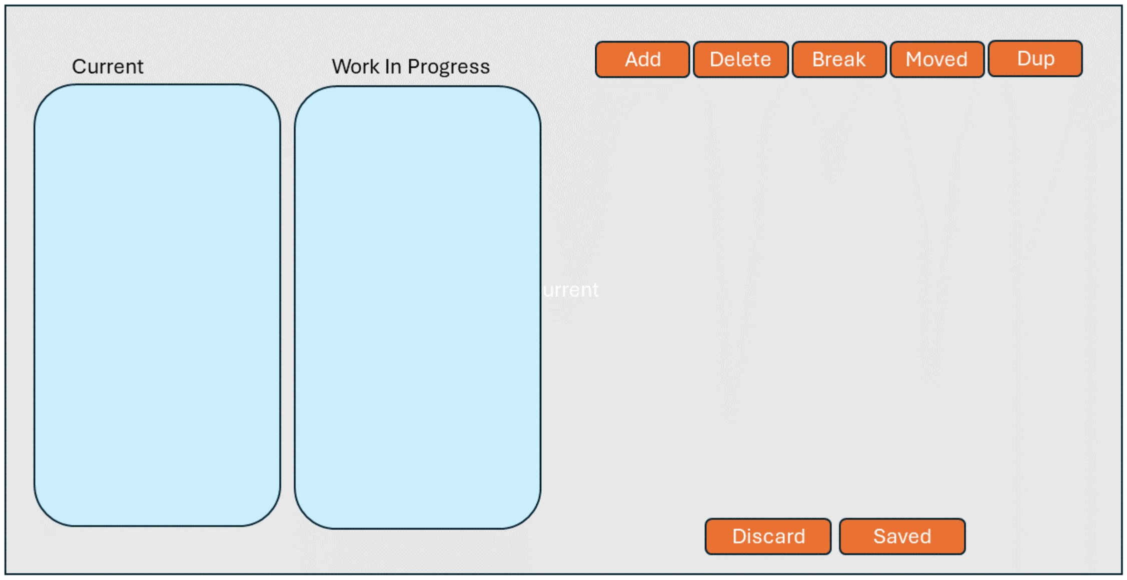


Figure 12: Stage Connectivity Proposed Work Area

The programming logic and datatypes requirement will involve setting up a class that contains at least the following:

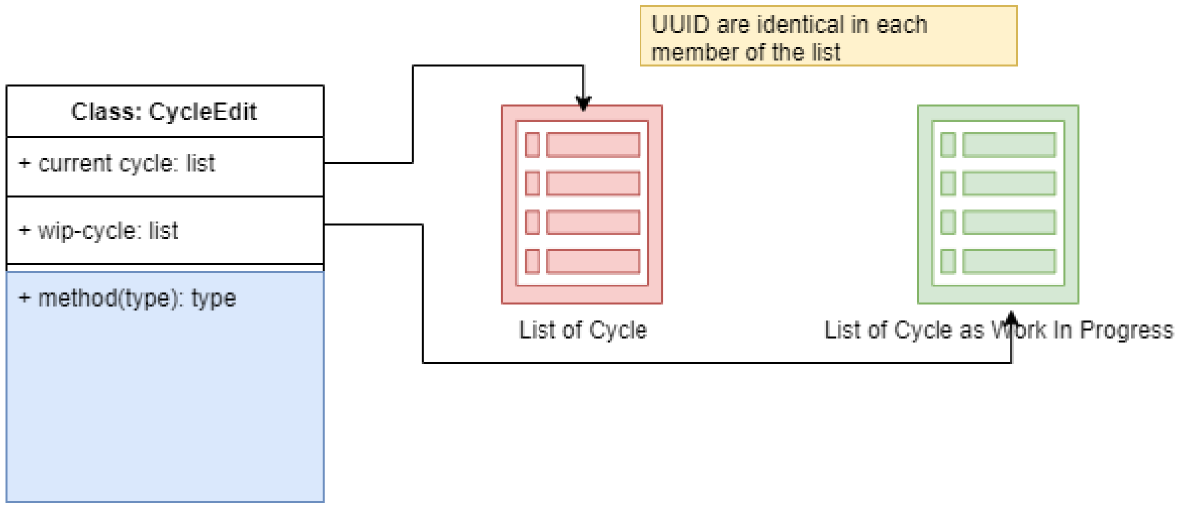


Figure 13: Class definition requirements for Edit Cycle

A list of records can be a tuple, which will have two components: a UUID and a pointer to the cycle. This list will display the current cycle that cannot be altered.

A list of records that will have two components: the same UUID as above and a pointer to the modifying cycle. This is the work that is currently in progress.

It's crucial to ensure that the 2 records point to different memory allocations. The references to the Records object are not the same.

18. Enabling multiple Active Cycles

At the moment, there is only one active cycle in memory. This implies that there is a single business process that is utilized for all claims. This may not be practical for cross-over FY or a new business process is needed for new claims, but older claims must align with the existing cycle.

We suggest the following approach for implementation.

a. It is necessary for the class instance in memory to have a list of Active cycles.
b. All Active cycles will be loaded by the REDIS storage.
c. Every claim must have a new field that is associated with their respective cycle.
d. Any resyncing during the WIP cycle process will modify the above field.
e. The processing cycle will now be in the database and not in the environment.

19. Future Direction / Long-Term Focus

There will be continuous addition to the execution list for the interpretive engine to process. This is also true for the syntax and semantic verification engine.

Another upgrade path is to design a graphical way of modifying the stage connectivity.

20. Conclusion

The business process modelling concept allows dynamic changes to the business process. Of course, as long as all the keywords are pre-defined. Addition to the keywork list requires additional coding for the IEs to process the keyword.

We are only implementing a single business process at any point in time. However, there is no limitation for that. The system can be enhanced to allow multi business process environment.

It is recommended for this system to mature before moving in that direction. Our advice to extend the keyword list for the Interpretive engine to process to the max. Nevertheless, we understand that this list will never be complete and exhaustive. It is best practice in an active business environment to keep adding the logic of the business process model.

21. Flowcraft

The Flowcraft ® brand was created by Schinkels's UI/UX team. The main asset is a web-browser application to manage the business process cycle and related stages. Without it, direct manipulation of business process content in the database is necessary. Going through the database is a complex task. Flowcraft ® app is an ongoing project.

Building the app has begun, but it will always be a work in progress.

Appendices

Appendix A – How is AI Used in Today's Business Model

Simple contained artificial intelligence (AI) based systems are increasingly becoming entrenched in our day to day lives. We adopt them and they disappear into the fabric of our daily lives, going unnoticed. AI in some form is encountered by most of us from the time we wake up in the morning until we go to bed at night. Just think of your conversations with Alexa, opening your phone with facial recognition, or the recommendations you receive when shopping online. When you use Amazon for example, AI algorithms go to work to understand your historical transactions, your preferences, and purchases made by people like you, and then it makes recommendations for items you will likely buy, all with a tremendous deal of accuracy. In fact, Amazon is so confident in its algorithms that it ships products towards your location in anticipation, even before you click buy.

In other more complex areas, AI and its connected technology is quickly evolving and working its way to the point of near autonomy. The top-of-mind example is of course the self-driving car. Today they are not fully autonomous but still semi-autonomous, requiring the person behind the wheel to oversee the system and remain responsible. The fundamental challenge with the development of autonomous cars is the issue of edge cases, which are the situations not typical of normal everyday driving. They are the circumstances which are new and consequently where the AI and technology has not been tested and therefore has not learned. The outcome in these cases is that the AI periodically fails, resulting in serious consequences and an appearance in the news cycle. However, this is changing and eventually the edge cases will be infrequent enough that trust can be placed in self-driving cars.

In business it is a different situation. Most of the edge cases are worked out in simulators beforehand. The ones that are encountered in the real world are not a matter of life and limb, but rather learning experiences that ultimately help the AI learn and adapt over the long-term. Today's best-in-class AI technology in this area typically makes successful choices at an extremely high rate and makes them thousands of times faster than even the most advanced traditional combination of people and analytics.

A great illustration of autonomous AI technology working in business today is software that provides recommendations and makes decisions for merchandise planning – decisions such as what items to promote, at what prices, and how much inventory to carry for each item in order to meet sales and financial targets. Here the best-in-class technology is making successful choices more than 98% of the time at lightning speed, with the Merchants only jumping into situations (edge cases) that are new to the technology or to shift the strategic direction. This is analogous to what drivers of semi-autonomous cars are required to do today, jumping in in new situations and setting the destination of the journey.

Autonomous self-driving cars are probably not right around the corner. However, in the business world, AI systems are improving on what people can do, saving labour, moving people away from mundane tasks, and allowing them to focus on the more strategic, high return activities.

Table of Figures