

# **A Novel Workflow Architecture based on Flow Mutation and Rendering As A Service**

*submitted in partial fulfillment of the requirements  
for the degree of*

**BACHELOR OF TECHNOLOGY**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

*by*

<b>KS KOUSHIK</b>	<b>CS17B013</b>
<b>BRIJESH VORA</b>	<b>CS17B031</b>
<b>GUDISEVA BALA SUSHMITHA</b>	<b>CS17B033</b>

**Supervisor(s)**

**Dr. Y. Kalidas**

भारतीय प्रौद्योगिकी संस्थान तिरुपति



**TIRUPATI**

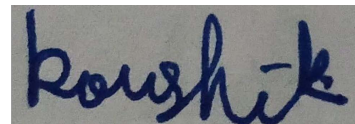
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI**

**MAY 2021**

## DECLARATION

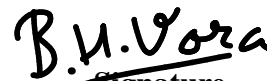
We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. we also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission to the best of my knowledge. we understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Place: Tirupati  
Date: 06-06-2021




**Signature**  
KS Koushik  
CS17B013

Place: Tirupati  
Date: 06-06-2021



**Signature**  
Brijesh Vora  
CS17B031

Place: Tirupati  
Date: 06-06-2021



**Signature**  
Gudiseva Bala Sushmitha  
CS17B033

## **BONA FIDE CERTIFICATE**

This is to certify that the report titled **A Novel Workflow Architecture based on Flow Mutation and Rendering As A Service**, submitted by **KS Koushik, Bri-jesh Vora** and **Gudiseva Sushmitha** to the Indian Institute of Technology, Tirupati, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the project work done by them under our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Tirupati  
Date: 06-06-2021



**Dr. Y. Kalidas**  
Guide  
Assistant Professor  
Department of Computer  
Science and Engineering  
IIT Tirupati - 517619

## **ACKNOWLEDGMENTS**

We want to thank our supervisor, Dr Y. Kalidas, for his invaluable supervision, expertise and support, who has guided us throughout our B.Tech Project. We extend our thanks to all the other faculty members in the Computer Science department and the members of Academics office who gave suggestions and critiques to improve our workflow system. Finally, we thank the M. Tech alumni Animesh Nanda, Vamsi Krishna and Kirtiman Mishra and MS Scholar Prashanth K, who provided us with the necessary starter code to continue their project.

# ABSTRACT

**KEYWORDS:** Workflow ; Microservices; Flow Mutation; Rendering as a service.

Workflow systems is still an open area of research due to diverse domain scenarios, response time, processing power, storage restrictions, domain specific scenario and security aspects. In this regard, 14 key features have been identified of any workflow system as domain agnosticism, content based and dynamic routing, message mutations, definition of nodes, workflows, their reuse and distribution, flexibility in rendering, user interface and computational processes, web adaptability, compatibility with low end devices and internet of things, security and ability to publish and discover functional capabilities. Formalism based reasoning to discover a fundamental issue in the state of the art for lack of customizability, as the orchestration logic which is a simple map from source to target nodes and forces nodes to couple with routing logic has been done. This issue is solved through a novel concept of *flow mutation* to expand the scope of routing logic to modify a message there by making a node more thin. In addition, for user interface nodes, a fast upcoming concept of *rendering as a service* for interface generation has been used. For computational nodes, a novel scheme of process-triad for data science workflows for model, data and control as services is introduced. The formalism can be realized in any implementation however a proof of concept is provided using Python/Flask for use cases for smart campus, computer vision and machine learning processes. In a nutshell, this report contains a novel workflow architecture backed by formalism for building of highly flexible, scalable, domain agnostic and light weight systems to provide plug-n-play nodes, workflows and rendering and security.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>LIST OF TABLES</b>	<b>x</b>
<b>ABBREVIATIONS</b>	<b>xi</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Motivations . . . . .	1
1.2 State of the Art . . . . .	2
1.3 Gap Area . . . . .	3
1.4 Our contributions . . . . .	4
1.5 Organization of the thesis . . . . .	4
<b>2 Background</b>	<b>6</b>
2.1 Microservices . . . . .	6
2.2 JSON . . . . .	6
2.3 Python-Flask . . . . .	7
2.3.1 WSGI . . . . .	8
2.3.2 Jinja2 . . . . .	8
2.4 Bcrypt . . . . .	9
2.5 MongoDB . . . . .	10
2.5.1 Pymongo . . . . .	10
<b>3 Proposed Workflow Engine</b>	<b>12</b>
3.1 Formal representation and reasoning . . . . .	12
3.2 Schematic of the system . . . . .	14
3.3 Features . . . . .	16

3.3.1	Domain Agnosticism . . . . .	17
3.3.2	Content-based Routing and Dynamic routing . . . . .	17
3.3.3	Flow Mutation . . . . .	17
3.3.4	Node Reuse . . . . .	18
3.3.5	Rendering as a Service . . . . .	18
3.3.6	Distributed Nature of Workflow System . . . . .	19
3.3.7	User-Interface Node . . . . .	19
3.3.8	Computational Node . . . . .	19
3.3.9	Web-API . . . . .	19
3.3.10	Light Weight . . . . .	19
3.3.11	IoT Enabled . . . . .	20
3.3.12	Communication Security . . . . .	20
3.3.13	Anonymity of Nodes . . . . .	20
3.4	Proof of concept using Python and Flask . . . . .	21
<b>4</b>	<b>Results - Course Registration/AddDrop Forms</b>	<b>23</b>
4.1	About this Chapter . . . . .	23
4.2	Course Registration . . . . .	23
4.3	Add/Drop Course . . . . .	26
<b>5</b>	<b>Results - HTTA/HTRA Forms</b>	<b>38</b>
5.1	HTTA/HTRA forms . . . . .	38
5.2	General . . . . .	41
<b>6</b>	<b>Results - Data Generator based Continual Learning Systems for Edge Devices</b>	<b>50</b>
6.1	About this Chapter . . . . .	50
6.2	Remote Data Generation (RDG) architecture . . . . .	50
6.2.1	Prediction-Service . . . . .	50
6.2.2	Training-Service . . . . .	51
6.2.3	Data-Service . . . . .	51
6.3	Genetic Algorithm (GA) as a data generator . . . . .	51
6.4	General Adversarial Networks (GAN) as a data generator . . . . .	54
6.5	Guassian Mixture Models (GMM) as Data Generator . . . . .	55

6.6	Original Data as a service . . . . .	56
6.7	Implementation Details . . . . .	57
<b>7</b>	<b>Conclusions and Future Directions</b>	<b>59</b>
<b>8</b>	<b>Contributions</b>	<b>61</b>
<b>A</b>	<b>Installation</b>	<b>62</b>
<b>B</b>	<b>Call Flow Graph</b>	<b>64</b>



## LIST OF FIGURES

- 3.1 Schematic diagram of workflow system is depicted in this figure. Green rectangles indicate key functional nodes for UI, Rendering, Computations, Workflow and Routing logic. There are 4 database each for node, workflow, routing logic and domain specific key-value information. Routing logic can be dynamically provided. The blue and orange small circles correspond to messages before and after modification respectively by the workflow module. . . . . 14
- 3.2 Schematic depiction of a user interface type node. The node processes any message characterized by user, role and job fields. The interface has interaction elements of diverse types including: file upload of diverse types, radio button, check boxes, text boxes and submit button. The execute button performs message modification and submission to the workflow. There are convenience features to do-undo and reset. . . . 15
- 3.3 This is a depiction of sequence of steps in a typical user interaction node. The green rectangles indicate action modules and the numbered circles denote the sequence of steps. The orange rectangles indicate background modules and their relationship to the foreground interface. The detailed steps are covered in the manuscript text. . . . . 16
- 3.4 A depiction of the **flow mutation** concept. The present workflow systems consider a router as a *postman* type there by leading to higher node complexity. The figure depicts a complex node coupling with routing logic and rendering. The figure depicts the effect of decoupling a node with routing logic and making it a *coordinator* type and allowing flow mutation. A user interface node is processed by the concept of *rendering as a service* where interface itself is dynamically generated. . . . 18
- 3.5 A highly distributed workflow system is depicted here. (A) Denotes the workflow system instance which as nodes of two types computational and interaction type, local workflows and a transfer workflow. It has databases for nodes and workflows. (B) Denotes one computer having multiple workflow system instanced deployed and running simultaneously. (C) Denotes a scenario of multiple computers, each with a multitude of workflow system instances and inter communicating via transfer workflows and receiver nodes. . . . . 21
- 3.6 This figure depicts modules in the proof of concept system built using python and flask libraries. The user interaction components are shown in green colour. The HTML box corresponds to user visualization and interaction with graphical elements. Databases for node, workflow and templates are shown as cylinders. Arrows indicate flow of data or next steps. The annotations on top of arrows denote specific relationships. 22

3.7	This figure depicts the process_wf.py module built using python. Databases for node and workflow are shown as cylinders . . . . .	22
4.1	The figure depicts flow of steps in the scenario for course registration in an academics scenario. . . . .	24
4.2	The figure depicts the login page with academics credentials. . . . .	24
4.3	The figure depicts the dashboard of academics. . . . .	25
4.4	The figure depicts the templates available for academics. . . . .	25
4.5	The figure depicts the rendered job for academics. Academics has entered the job name and uploaded the empty Course Registration form. . . . .	25
4.6	The figure depicts the rendered course registration job in the Student Dashboard. . . . .	26
4.7	The figure depicts the empty course registration form sent by academics to the student. . . . .	27
4.8	The figure depicts the rendered course registration job in the Faculty Advisor Dashboard, containing the form filled by student. . . . .	28
4.9	The figure depicts the rendered course registration job in the Academics Dashboard, containing the form filled by student and acknowledged(signed) by Faculty Advisor. . . . .	28
4.10	The figure depicts the directory where all the signed course registration form is downloaded automatically. . . . .	28
4.11	The figure depicts flow of steps in the scenario for course add and drop in an academics scenario. . . . .	29
4.12	Figure showing adding a new job for add/drop course template . . . . .	29
4.13	Figure showing academics entering the job name and uploading PDF form just before sending to students . . . . .	29
4.14	Figure showing a message after submit and job is processed . . . . .	29
4.15	Figure showing student's dashboard . . . . .	30
4.16	Figure showing empty add-drop course PDF form . . . . .	30
4.17	Figure showing student's dashboard with job just before sending to course instructor . . . . .	31
4.18	Figure showing course instructor's dashboard . . . . .	31
4.19	Figure showing PDF form sent by the student to the course instructor . . . . .	32
4.20	Figure showing course instructor's dashboard before sending to student . . . . .	32
4.21	Figure depicting student dashboard with job received from the first course instructor . . . . .	33
4.22	Figure showing PDF form filled by one course instructor . . . . .	33

4.23	Figure showing the student dashboard with job received from course instructor . . . . .	34
4.24	Figure depicting student dashboard with sending to faculty advisor radio button changed to 'YES' . . . . .	34
4.25	Figure showing faculty advisor's dashboard . . . . .	35
4.26	Figure showing PDF form filled by two course instructors . . . . .	35
4.27	Figure showing jobs received from students by academics . . . . .	36
4.28	Figure showing completely filled add/drop form . . . . .	37
4.29	Figure showing Student Notifications . . . . .	37
5.1	The figure depicts flow of steps in the scenario for HTRA form in an academics scenario. . . . .	38
5.2	The figure depicts the templates available for academics. . . . .	39
5.3	The figure depicts the rendered job for academics. Academics has entered the job name and uploaded the empty HTRA form. . . . .	39
5.4	The figure depicts the rendered HTRA job in the Student Dashboard. . . . .	39
5.5	The figure depicts the empty HTRA form sent by academics to the student. . . . .	40
5.6	The figure depicts the rendered HTRA job sent by student to the TA supervisor. The supervisor can click on the PDF file to view the form. . . . .	41
5.7	The figure depicts the notifications of the student. . . . .	41
5.8	The figure depicts the rendered HTRA job sent by TA Supervisor to the guide. The guide can click on the PDF file to view the form. . . . .	42
5.9	The figure depicts the job containing the completely filled HTRA Form. . . . .	42
5.10	Figure showing dashboard of <i>nx</i> and adding a general template . . . . .	42
5.11	Figure showing entering job name and role name in general template . . . . .	43
5.12	Figure depicting different fields of data that can be added . . . . .	43
5.13	Figure depicting adding text field . . . . .	44
5.14	Figure depicting adding drop down field . . . . .	45
5.15	Figure showing job after adding text and drop down fields . . . . .	45
5.16	Figure depicting adding checkbox field . . . . .	46
5.17	Figure depicting adding radio button field . . . . .	47
5.18	Figure depicting adding file upload field . . . . .	48
5.19	Figure showing job after adding all the possible fields . . . . .	48
5.20	Figure showing job after filling in the data in all the fields added . . . . .	49

5.21	Figure showing <i>nz</i> dashboard and the job received from <i>ny</i> . . . . .	49
6.1	Figure shows the interaction between controller and the database via workflow system when generating synthetic data using Genetic Algorithms. . . . .	52
6.2	Figure shows the shows the retraining process of the system after synthetic data has been generated. . . . .	52
6.3	the block diagram shows the interactions between various services when either GMM or GAN are used as Data services. . . . .	54
6.4	the figure shows the concept of pseudo rehearsal. the synthetic data of previous batch, generated using GAN or GMM are interleaved with the newly arrived training data. . . . .	55
6.5	Figure shows the interactions between the Controller and the database containing the original data. . . . .	56
B.1	The figure depicts the HTML file inheritance with layout.html as base HTML file. . . . .	64
B.2	The figure depicts routing for basic HTML files . . . . .	65
B.3	The figure depicts routing for adding templates and job rendering. . . . .	65
B.4	The figure depicts routing for notifications. . . . .	66
B.5	The figure depicts flow of workflow engine for academic use cases as an example. . . . .	66

## LIST OF TABLES

1.1	14 important features (section 3.3) in any workflow system have been identified that are critical to its wide spread use, scalability and flexibility which are depicted in the tabulation here. . . . .	4
1.2	A comparison of state of the art algorithms with respect to the 14 aspects of any workflow system. Here 0 means, the feature is not addressed and 1 means it is addressed by the corresponding method. .	5

## **ABBREVIATIONS**

<b>AWS</b>	Amazon Web Services
<b>DAG</b>	Directed Acyclic Graph
<b>GA</b>	Genetic Algorithm
<b>GAN</b>	General Adversarial Networks
<b>GMM</b>	Gaussian Mixture Models
<b>GUI</b>	Graphical User Interface
<b>PoC</b>	Proof of Concept
<b>REST</b>	Representational State Transfer
<b>UI</b>	User Interface

# CHAPTER 1

## INTRODUCTION

Workflow management is a very standard requirement in any organization involving administration, business or scientific processes. A workflow is theoretically equivalent to a program involving shared functions across diverse programs. A workflow involves several steps, where each step itself may be a complex business process. These steps or individual processes are shared among several dozens of workflows and their instances. The flow of a message or a snapshot of several related messages among these processes constitutes a workflow.

The application domain itself may be varied with different requirements and service level agreements. The processes range from slow and time consuming steps to real time processes, less data to several tera-bytes of storage requirements, novice to scientific processes and human-in-loop systems across diverse domain in e-commerce, business to business, defense, bioinformatics, administration and several other organization where processes are involved. There have been several dozens of workflows systems worldwide over more than last two decades and still newer systems are evolving.

### 1.1 Motivations

There is a need to understand what is common across all these systems, what is missing in common amongst the state of the art, why is it that several systems are still evolving and is there a way to define and address in a formal and theoretical setting and demonstrate by a proof of concept of the ideas developed. There are hundreds of workflow systems worldwide. The main domains are business, administration and scientific. There is a need to address why there are so many systems and is there a way to address at the fundamental level. There is also a need to address the lack of customization in the orchestration.

## 1.2 State of the Art

Various state of the art workflow systems have been explored and studied. Most of the literature talked about features like security, scalability and distributed node objects. Many papers discuss domain specific scenarios like document management systems [Joseph and Mosweu](#), life sciences (agriculture [Liu et al. \(2016\)](#) and health [Yang et al. \(2019\)](#)), scientific workflows ( [Chen et al. \(2015\)](#), [Rynge et al. \(2012\)](#), [Mandal et al. \(2015\)](#), [Tovar et al. \(2017\)](#), [Ludäscher et al. \(2006\)](#), [Mandal et al. \(2017\)](#) and so on), sensors in defense scenarios [Qiu et al. \(2019\)](#) and education technology ([Yang et al. \(2018\)](#), [Pap et al. \(2020\)](#)).

Some state of the art systems talks about job scheduling. For example, [Mujezinović and Ljubović \(2019\)](#) schedules workflow in a serverless architecture using producer-consumer architectural patterns and AWS lambda functions. [Cao et al. \(2003\)](#) discusses workflow for grid computing and the Fuzzy Time technique for workflow scheduling and conflict management. [Li et al. \(2016\)](#) talks scheduling in big data analytics and reuses node objects. Another feature that some systems discuss is the allocation of resources for better performance. [Chen et al. \(2015\)](#) talks about dynamic task clustering strategies to merge several short tasks into a single job and to improve the runtime performance of workflow executions in faulty execution environments. [Tovar et al. \(2017\)](#) termed this as the job sizing problem and recommends a resource feedback loop that uses historical information to compute a recommended first allocation for the job sizing problem. Further, [Król et al. \(2016\)](#) talks about workflow performance profiles, which analyzes workflow profiles based on time series data collected from real workflow executions. [Nawaz et al. \(2016\)](#), [Deelman et al. \(2019a\)](#), [da Silva et al. \(2017\)](#), [Ceri et al. \(1997\)](#) and [Amin et al. \(2018\)](#) discuss high performance computer simulations in large-scale applications. [Nawaz et al. \(2016\)](#), [Sánchez-Gallegos et al. \(2019\)](#) include support for IoT and IAAS. Cloud execution specific scenarios are discussed in [Mandal et al. \(2015\)](#), [Nawaz et al. \(2016\)](#), [Filgueira et al. \(2016\)](#), [Li et al. \(2016\)](#) and [Hoffa et al. \(2008\)](#) which include I/O and data-intensive workflows.

When it comes to representing a workflow, many state-of-the-art systems use DAG as the primary medium. [Rynge et al. \(2012\)](#), [Yang et al. \(2018\)](#), [Deelman et al. \(2020\)](#), [Mandal et al. \(2017\)](#) and [Linke et al. \(2011\)](#) talks using loosely-coupled workflows as



independent DAGs with each node as task and edge as dependencies. Definite interfaces for workflows are discussed in some of the literature. [Li et al. \(2016\)](#) uses textual and GUI interface for entering the requirements. Some workflow systems use user nodes, computational nodes or both. [Joseph and Mosweu](#) and [Ludäscher et al. \(2006\)](#) have only user nodes and no computational nodes. [Alonso et al. \(1995\)](#) uses only computational nodes. [Liu et al. \(2016\)](#) uses both. [Rynge et al. \(2012\)](#) and [Betancourt et al. \(2019\)](#) have an interface for passing messages between the jobs. [Alonso et al. \(1995\)](#) uses a persistent message passing system instead of a centralized database.

Exploring the key features in the literature, [Rynge et al. \(2012\)](#) uses content-based routing, but the node decides what to do upon receiving a message and not the routing logic. [Liu et al. \(2016\)](#), [Wolstencroft et al. \(2013\)](#) and [da Silva et al. \(2017\)](#) discusses using nodes as microservices. [Brzeziński et al. \(2011\)](#) and [Liu et al. \(2016\)](#) use workflows where nodes are communicating via REST.

### 1.3 Gap Area

After analysing the various state of the art systems, the common concepts amongst all the workflow systems have been identified. Further features are also proposed. All the features are discussed in detail in the section 3.3 and described in table 1.1. Many of them not address at the fundamental level the need for node independence from the knowledge of many a workflow in which it participates while still retaining ability to take part in decision branches. The literature mainly focuses on distributed node objects, execution, inter-node communication mechanisms, workflow abstraction as a graph and user interfaces. The existing methods are compared and contrasted against these features and critical gap areas are summarized (Table 1.2).

*Flow mutation* is not implemented in any of the examined systems. The concept of *rendering as a service* which use introduced in our system, is a recent phenomenon in the state of the art and only seen in about 6 systems out of 42 systems studied including our proposed system.

Feature number	Feature name	Abbreviation
1	Domain Agnosticism	DA
2	Content based Routing	CR
3	Dynamic Routing	DR
4	Flow Mutation	FM
5	Node Reuse	NR
6	Rendering as a Service	RS
7	Distributed/Scalable Systems	DS
8	User interface Node	UN
9	Computational Node	CN
10	Web API	WA
11	Light Weight	LW
12	Internet of things Enabled	IE
13	Communication Security	CS
14	Node Publication	NP

Table 1.1: 14 important features (section 3.3) in any workflow system have been identified that are critical to its wide spread use, scalability and flexibility which are depicted in the tabulation here.

## 1.4 Our contributions

A novel concept of *flow mutation* combined with the concept of *rendering as a service* to result a formalism enabling design of highly flexible, scalable and domain agnostic workflow systems have been proposed. The formalism is generic and any contemporary technology can be chosen to implement. A proof of concept implementation of the workflow architecture in Python environment and Flask libraries have been presented. The PoC is evaluated on a set of 5 scenarios for a smart campus use case, a mimic smart city use case, machine learning and computer vision use cases and IoT device connectivity use cases.

## 1.5 Organization of the thesis

Chapter 1 contains introduction to the workflow systems, why it is needed and motivation for building such systems. Also, it contains state of the art, gap areas and our contribution and literature review. Chapter 2 includes the background information of the technologies used - Flask, MongoDB, Python. Chapter 3 contains the core architecture, Mathematical formulation and salient features of our workflow systems. Chapter 4 contains Use cases pertaining to our workflow system - Course Registration and Add

S.No	Reference	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Feature coverage
1.	<a href="#">Alonso et al. (1995)</a>	1	0	1	0	0	0	1	0	1	0	1	0	1	1	50%
2.	<a href="#">Ceri et al. (1997)</a>	0	0	0	0	0	0	1	0	0	1	1	0	1	0	29%
3.	<a href="#">Grefen et al. (2000)</a>	0	0	1	0	0	1	1	0	0	0	1	0	1	1	43%
4.	<a href="#">Kappel et al. (2000)</a>	1	0	1	0	0	0	0	0	0	0	1	0	1	0	29%
5.	<a href="#">Rinderle et al. (2003)</a>	0	0	1	0	0	1	1	0	0	0	1	0	1	1	43%
6.	<a href="#">Cao et al. (2003)</a>	0	0	0	0	0	1	1	1	0	0	1	0	1	1	43%
7.	<a href="#">Heinis et al. (2005)</a>	0	1	0	0	0	0	1	0	0	0	1	0	1	1	36%
8.	<a href="#">Ludäscher et al. (2006)</a>	0	0	0	0	1	1	1	1	1	0	1	0	1	1	57%
9.	<a href="#">Barker and Van Hemert (2007)</a>	0	0	0	0	0	0	1	1	1	0	0	0	1	0	21%
10.	<a href="#">Missier et al. (2008)</a>	0	0	0	0	0	1	0	1	0	1	1	1	1	0	36%
11.	<a href="#">Hoffa et al. (2008)</a>	0	0	0	0	0	1	0	1	0	0	0	0	1	1	29%
12.	<a href="#">Gil et al. (2010)</a>	1	0	0	0	0	1	0	1	0	0	0	0	1	0	29%
13.	<a href="#">Neophytou et al. (2011)</a>	0	0	0	0	0	1	0	0	1	0	0	0	1	1	36%
14.	<a href="#">Linke et al. (2011)</a>	0	0	0	0	0	1	0	0	1	0	0	0	0	0	14%
15.	<a href="#">Brzeziński et al. (2011)</a>	1	0	0	0	1	0	1	0	0	1	0	0	1	0	36%
16.	<a href="#">Rynga et al. (2012)</a>	0	0	0	0	0	1	0	1	0	0	0	0	1	0	21%
17.	<a href="#">Islam et al. (2012)</a>	0	0	0	0	0	1	0	1	0	0	0	0	1	0	21%
18.	<a href="#">Wolstencroft et al. (2013)</a>	0	0	0	0	1	0	1	0	1	1	0	0	0	1	36%
19.	<a href="#">Pradhan and Joshi (2014)</a>	0	0	0	0	0	1	0	0	0	1	0	1	0	0	21%
20.	<a href="#">Mandal et al. (2015)</a>	0	0	1	0	0	1	0	1	0	0	0	0	0	0	21%
21.	<a href="#">Chen et al. (2015)</a>	0	0	0	0	0	1	0	1	0	0	0	0	0	0	14%
22.	<a href="#">Filgueira et al. (2016)</a>	0	0	0	0	0	1	0	1	0	0	0	0	0	0	14%
23.	<a href="#">Nawaz et al. (2016)</a>	0	0	0	0	0	1	0	1	0	0	1	1	1	0	29%
24.	<a href="#">Liu et al. (2016)</a>	0	0	0	0	0	1	0	1	1	0	0	0	1	1	36%
25.	<a href="#">Li et al. (2016)</a>	1	0	0	0	0	1	0	1	1	1	0	0	0	0	29%
26.	<a href="#">Król et al. (2016)</a>	1	0	0	0	0	1	0	0	0	0	0	0	1	0	21%
27.	<a href="#">da Silva et al. (2017)</a>	0	0	0	0	0	1	1	1	1	0	0	0	0	1	29%
28.	<a href="#">Tovar et al. (2017)</a>	1	0	0	0	0	1	0	0	0	1	0	0	0	0	21%
29.	<a href="#">Mandal et al. (2017)</a>	1	0	1	0	0	0	0	0	0	0	0	0	0	0	14%
30.	<a href="#">Simpkin et al. (2018)</a>	0	0	0	0	0	1	1	1	1	0	0	0	1	1	36%
31.	<a href="#">da Silva et al. (2018)</a>	0	0	0	0	0	1	0	0	0	0	0	0	1	0	14%
32.	<a href="#">Yang et al. (2019)</a>	0	0	0	0	1	0	1	0	0	0	0	0	0	0	14%
33.	<a href="#">Tomsett et al. (2019)</a>	0	0	1	0	0	1	1	1	1	0	0	1	1	1	50%
34.	<a href="#">Deelman et al. (2019b)</a>	0	0	0	0	0	1	0	0	0	0	0	0	1	0	14%
35.	<a href="#">Sánchez-Gallegos et al. (2019)</a>	0	0	0	0	0	1	0	0	0	0	0	0	1	0	14%
36.	<a href="#">Deelman et al. (2019a)</a>	0	0	0	0	0	1	0	0	0	0	0	0	1	0	14%
37.	<a href="#">Mujezinović and Ljubović (2019)</a>	0	0	0	0	0	0	0	0	0	0	0	0	1	0	7%
38.	<a href="#">Altintas et al. (2019)</a>	0	0	0	0	0	0	0	1	1	0	0	0	1	1	29%
39.	<a href="#">Deelman et al. (2020)</a>	0	0	0	0	0	1	0	0	0	0	0	0	0	0	7%
40.	<a href="#">Joseph and Mosweu</a>	0	0	0	0	0	1	0	0	0	1	0	0	0	0	14%
41.	<a href="#">Apache (2014)</a>	1	0	1	0	0	1	1	0	0	0	0	0	1	0	36%
42.	<b>Proposed Workflow</b>	1	1	1	1	1	1	1	1	1	1	1	1	1	1	100%

Table 1.2: A comparison of state of the art algorithms with respect to the 14 aspects of any workflow system. Here 0 means, the feature is not addressed and 1 means it is addressed by the corresponding method.

Drop scenario. Chapter 5 contains use cases of workflow systems HTTA/HTRA and general. Chapter 6 contains use cases of the data generator in a continual learning scenario. Chapter 7 is the final chapter with conclusion and future directions. Chapter 8 contains the people who contributed to this project and made it successful.

# CHAPTER 2

## Background

Technologies used are python-flask, MongoDB, pymongo, bcrypt. Data is represented as JSON objects. Our architecture is based on Microservices.

### 2.1 Microservices

A microservice is an independent part of a usually large application, communicating with others over HTTP via the published API. They are independently deployable and maintainable. They communicate via the REST API they expose, most often in JSON, which is light-weight to process and transfer.

Benefits of Microservices are:

1. Continuous Integration and Continuous Deployment (CI/CD).
2. Containerization
3. Programming Language and Framework Independent
4. High Scalability
5. High Availability
6. High Resilience

### 2.2 JSON

JSON stands for JavaScript Object Notation. It is a light weight data interchange format.

JSON Syntax Rules:

1. Data is in name/value pairs.
2. Data is separated by commas.
3. Curly braces hold objects.
4. Square brackets hold arrays.

JSON Example:

```
{  
    "rollno" : "B123" ,
```

```
    "student_name": "Abc",
    "year": "2017",
    "programme": "B.tech",
    "specialization": "Computer Science and Engineering",
    "faculty_advisor": "advisor1"
}
```

## 2.3 Python-Flask

Flask is a web application framework written in Python. A collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols and thread management is called Web Application Framework. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine.

Example code of python flask is shown below.

```
from flask import Flask
from flask_cors import CORS
app = Flask(__name__)
app.secret_key = "workflowsystem"
CORS(app)

@app.route('/dashboard', methods = ['GET', 'POST'])
def dashboard():
    return "Hello World!"

if __name__ == '__main__':
    app.run(port='5000', debug=True)
```

### 2.3.1 WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. Werkzeug is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

Example code of flask request is shown below.

```
from flask import Flask, request
from flask_cors import CORS
app = Flask(__name__)
app.secret_key = "workflowsystem"
CORS(app)

@app.route('/dashboard', methods = ['GET', 'POST'])
def dashboard():
    if request.method == 'POST':
        f = request.form.get('id')
        return render_template('dashboard.html')

if __name__ == '__main__':
    app.run(port='5000', debug=True)
```

### 2.3.2 Jinja2

Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages. It provides a Django-inspired non-XML syntax but supports inline expressions and an optional sandboxed environment.

Example code of Jinja rendering template is shown below.

```
from flask import Flask, render_template
```

```

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(port='5000', debug=True)

```

## 2.4 Bcrypt

The bcrypt is a password hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher. The bcrypt function is the default password hash algorithm for OpenBSD. There are implementations of bcrypt for C, C++, C#, Java, JavaScript, PHP, Python and other languages.

Example code of creating a hashed password is shown below.

```

import bcrypt
passwd = b's$cret12'
salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(passwd, salt)
print(salt)
print(hashed)

```

Example code of password matching is shown below.

```

import bcrypt
passwd = b's$cret12'
salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(passwd, salt)
if bcrypt.checkpw(passwd, hashed):
    print("match")
else:
    print("does not match")

```

## 2.5 MongoDB

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. MongoDB uses JSON-like documents with optional schema. MongoDB works on concept of collection and document.

**Database:** Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

**Collection:** Collection is a group of MongoDB documents. It is similar to Relational Database Management System (RDBMS).

**Document:** A document is a set of key-value pairs. Documents have dynamic schema.

### 2.5.1 Pymongo

Python needs a MongoDB driver to access the MongoDB database. That MongoDB driver is pymongo. PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.

```
from pymongo import MongoClient

#Creating a pymongo client
client = MongoClient('localhost', 27017)

records = client['kali_db']['users']

#check if userID exists in database
userID_found = records.find_one({"userID": "xyz"})

# To insert data in database
records.insert_one({"userID": "abc"})
```



```
#To delete data in database
records.delete_one({"userID": "pqr"})
```

```
# To update one record in databse
records.update_one({'userID':userID},
{"$set":{"password":new_password}})
```

# CHAPTER 3

## Proposed Workflow Engine

A formal representation of the system and interpretation is provided for deeper understanding of any workflow system and our specific modifications. The representation is agnostic of technology and any state of the art mechanisms may be used to deploy. For the proof of concept and initial working model, the formalism in Python using Flask micro-services framework have been implemented.

### 3.1 Formal representation and reasoning

Here a formal representation of the system is presented and its capability in terms of flexibility and scalability.

1. Let  $M$  denote set of all messages,  $2^{V \times D}$
2. Here  $V$  is domain specific vocabulary,  $D$  is data and  $M$  is in key-value format
3. Let  $N$  denote set of node identifiers
4. Let  $W$  denote set of workflow identifiers
5. Let  $B_N : N \times M \rightarrow [N \times M \times A]$  denote set of behaviours
6. Here one message can trigger several actions, therefore list abstraction is used
7. The symbol  $A$  denotes an action,
8. Let,  $A : N \times M \rightarrow W \times M$  denote purpose of the action
9. Let,  $R : M \rightarrow M$  for changing contents of a message using rendering as a service
10. **Rendering as a service:** corresponds to modification of a message upon user interaction,  $m' = R(m)$
11. Let,  $B_W : W \times M \rightarrow [N \times M]$  denote workflow behaviour, to map a message to a list of nodes
12. **Flow mutation:**  $(\exists w \in W, m \in M), \exists (m' \neq m) : (n', m') \in L, L = B_W(w, m)$
13. The key point here is presence of  $W \times M \rightarrow N \times M$  instead of  $W \times M \rightarrow N$
14. Let  $E_N = \{(n, m) | n \in N, m \in M\}$  denote node event store
15. Let  $E_W = \{(w, m) | w \in W, m \in M\}$  denote workflow event store
16. Let  $n_\phi \in N$ ,  $w_\phi \in W$  and  $m_\phi \in M$  denote no operation node and workflow and empty message respectively
17. For  $n_\phi$ , the functionality is  $(w_\phi, m) = B_N(n_\phi, m)$
18. For  $w_\phi$ , the functionality is  $(n_\phi, m) = B_W(w_\phi, m)[0]$
19. (Note here that  $n_\phi, w_\phi$  are only for theoretical completeness, there are not actual function calls in any implementation)

The workflow and node daemons are shown in (Algorithm 1) and (Algorithm 2) respectively. Some of the key inferences from the **node process** are as here.

- Node behaviour is plug and play, i.e.  $B_N$  can be dynamically configured
- The actions upon a given message,  $A$  can be of two types - computational or user interaction
- **Rendering as a Service:** If it is user interaction type,  $R(m)$  can be used to render a message  $m \in M$  and generate modified content

Some of the key inferences from the **workflow process** are here.

- Workflow behaviour is plug and play, i.e.  $B_W$  can be dynamically configured
- Content drives the routing, i.e.  $B_W(w, m)$ ,  $m \in M$  becomes critical
- **Flow mutation:** The workflow can modify the message, i.e. in  $[(n', m') \dots] = B_W(w, m)$ ,  $\exists m' \neq m$  can be true

---

#### Algorithm 1: Node Process

---

```

1 Workflow Daemon:
2  $E_W = E_W + \langle w_\phi, m_\phi \rangle$ 
3 //Infinite iteration
4 while  $|E_W| > 0$  do
5   if  $\exists e \in E_W : e[0] \neq w_\phi$  then
6      $E_W = E_W - e$ 
7      $w = e[0]$ 
8      $m = e[1]$ 
9      $NL = B_W(w, m)$  //get list of nodes to which this message is mapped
10    while  $(\forall (n', m') \in NL)$  do
11       $E_N = E_N + (n', m')$  //goes to node event store

```

---



---

#### Algorithm 2: Workflow Process

---

```

1 Node Daemon:
2  $E_N = E_N + \langle n_\phi, m_\phi \rangle$ 
3 //Infinite iteration
4 while  $|E_N| > 0$  do
5   if  $\exists \eta \in E_N : \eta[0] \neq n_\phi$  then
6      $E_N = E_N - \eta$ 
7      $n = \eta[0]$ 
8      $m = \eta[1]$ 
9      $AL = B_N(n, m)$  //obtain a list of actions
10    while  $(\forall (n', m', \alpha) \in AL)$  do
11      //applying action  $\alpha(\cdot, \cdot)$ 
12      //  $\alpha$  can be user interaction or computational type
13      if  $\alpha$  is user interaction then
14         $m'' = R(m')$  //using rendering as a service
15      else
16        //when  $\alpha$  is computational type
17         $m'' = \chi(n', m')$  //where  $\chi(\cdot, \cdot)$  is a computational node
18         $(w', m'') = \alpha(n', m')$ 
19         $E_W = E_W + (w', m'')$  //goes to workflow event store

```

---

## 3.2 Schematic of the system

The formalism may be realized in diverse platforms and application technologies. A software design perspective of the formalism is presented in the schematic (Figure 3.1). The workflow system focuses on flexibility and scalability are addressed at fundamental level of message routing and processing. In this architecture there are two types of nodes - (i) user interaction processes and (ii) computational processes.

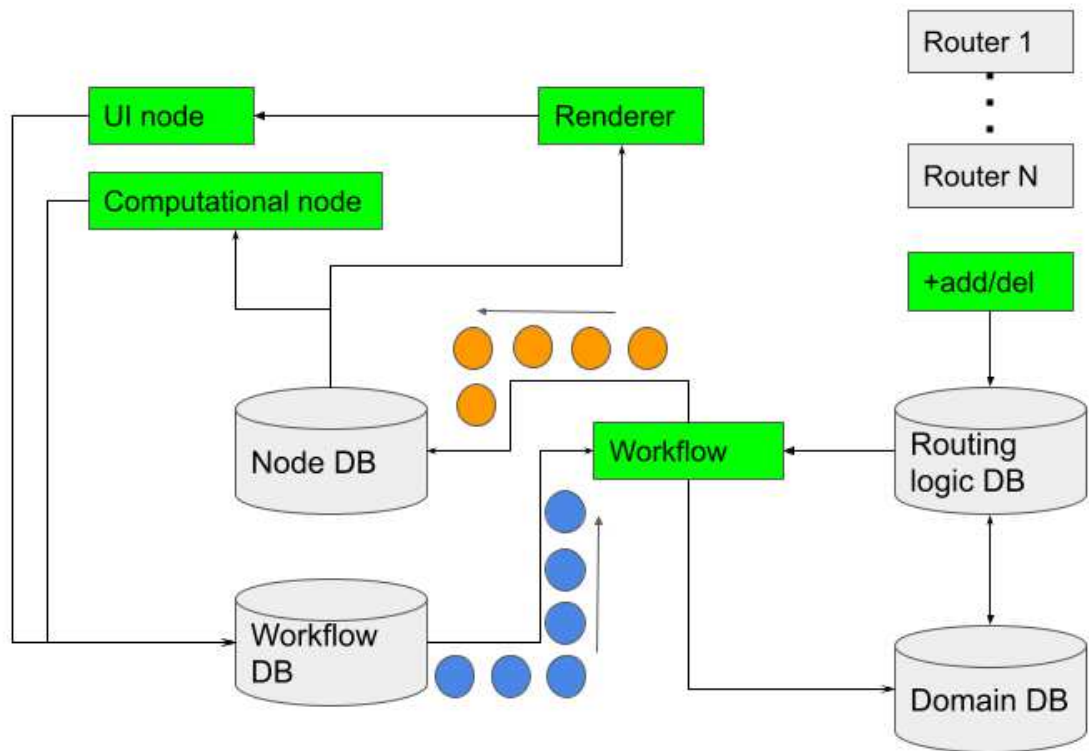


Figure 3.1: Schematic diagram of workflow system is depicted in this figure. Green rectangles indicate key functional nodes for UI, Rendering, Computations, Workflow and Routing logic. There are 4 database each for node, workflow, routing logic and domain specific key-value information. Routing logic can be dynamically provided. The blue and orange small circles correspond to messages before and after modification respectively by the workflow module.

The workflows dynamically load routing logistics. This is the plug-n-play mechanism that brings in enormous flexibility to customize for diverse domains and at scale. A reload of the routing logic does not require the restart of a system, it can be done on the fly.

The routing logic modifies the message content as well, which is called as *flow mutation*. This novel concept of *flow mutation* offers flexibility to control workflows

across scenarios. This also decouples a node from the knowledge of workflows in which it is participating.

A user interface node offers a friendly graphical interface to the end user. The interaction elements are customizable and can be dynamically added or deleted. A schematic of the user interface node is shown in (Figure 3.2).

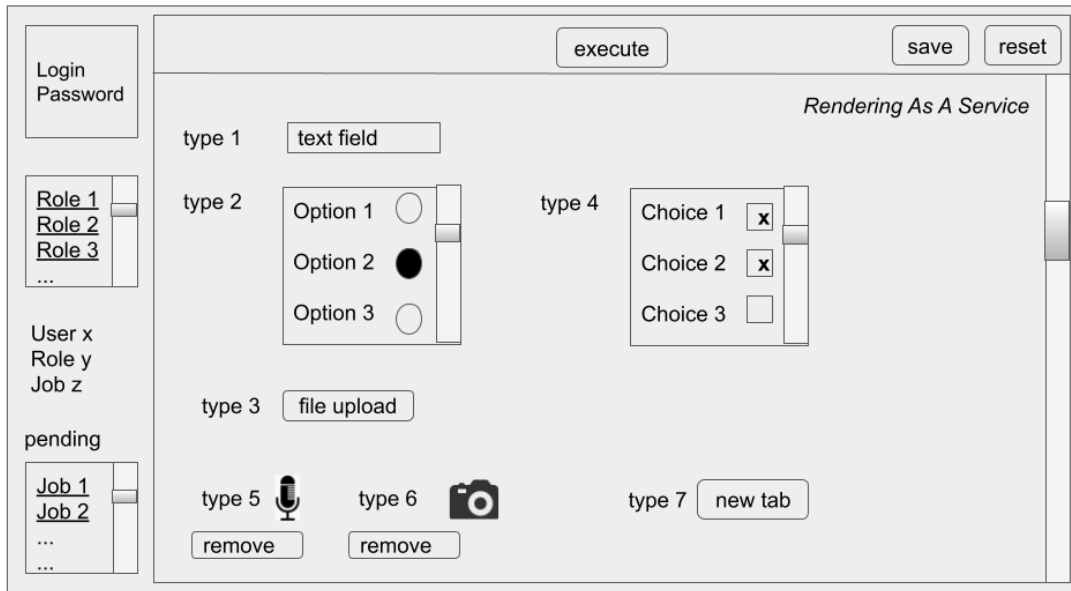


Figure 3.2: Schematic depiction of a user interface type node. The node processes any message characterized by user, role and job fields. The interface has interaction elements of diverse types including: file upload of diverse types, radio button, check boxes, text boxes and submit button. The execute button performs message modification and submission to the workflow. There are convenience features to do-undo and reset.

The user interface node has the following type of interactions, and can be extended based on implementation of the formalism in any specific platform.

- Text input
- File upload
- Radio buttons
- Check boxes
- Submit buttons
- Other may be added as required in any specific implementation.

The node execute button, confirms the data entered by the user and send the *modified message* to the workflow. The back-end workflow system then processes the message and routes the messages with or without modification to subsequent nodes and the system continues.

A user interface node interacts with a number of back-end modules. As the node

becomes agnostic of the workflows, it becomes thin. A thin node still requires visualization of content and interaction with user. This visualization as well can be off-loaded as this is a common requirement across nodes. *Rendering as a service* is made. The sequence of interactions with backend system are shown in (Figure 3.3) in a *dashboard*.

- STEP 1: Login. Authentication happens in this step.
- STEP 2: Selection of role and job.
- STEP 3: Visualization of the rendered message by the Rendering Service
- STEP 4: User interaction via graphical elements
- STEP 5: Node execution

In addition, there are further options available to the user such as,

- STEP 6: Add new/Delete/Modify jobs
- STEP 7: Add new/Delete/Modify templates
- STEP 8: Add new /Delete/Modify routing logistics

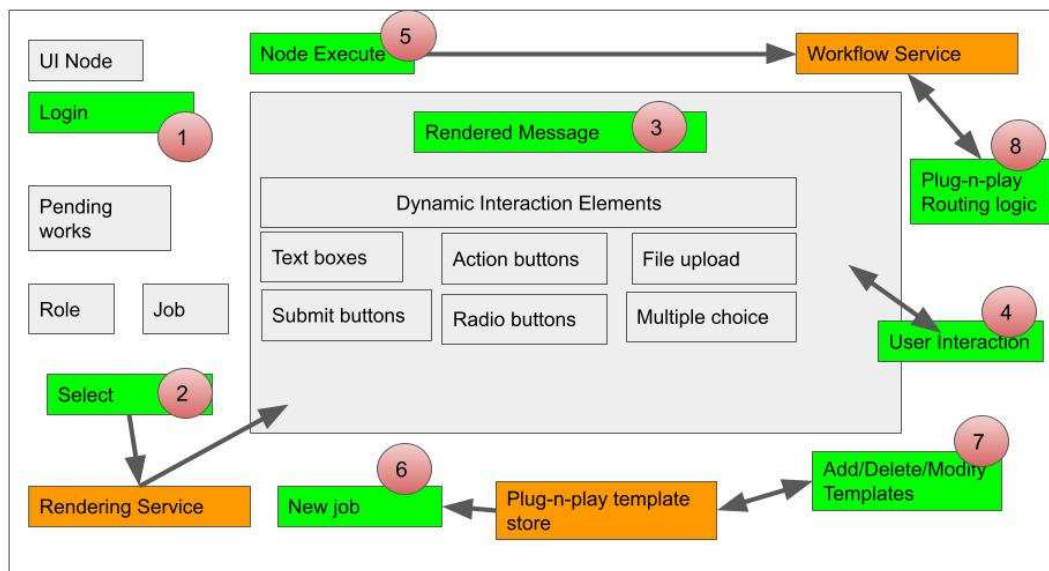


Figure 3.3: This is a depiction of sequence of steps in a typical user interaction node. The green rectangles indicate action modules and the numbered circles denote the sequence of steps. The orange rectangles indicate background modules and their relationship to the foreground interface. The detailed steps are covered in the manuscript text.

### 3.3 Features

### 3.3.1 Domain Agnosticism

All the workflow systems that have ever been built, were built for a particular domain or purpose. For example, a workflow system built for a pharmaceutical industry may not be suitable for an automobile industry as their requirements vary. The proposed engine is agnostic of all those and can cater to any use case. It provides all the bare basics upon which one can customize. The engine just acts as a ‘router’ of information from node to node, which routes based on some of the parameters of the content. The engine is unaware of this and just routes the information.

### 3.3.2 Content-based Routing and Dynamic routing

In traditional workflow systems, there’s no way to load routing conditions dynamically and content-based routing was not available. But here that problem is solved using a separate module that can be loaded dynamically, which has all the decision-making logic for content-based routing. Content-based routing means the nodes to which the information be forwarded will be decided based on some of the parameters in the information. And all that decision-making logic will be in a separate module and is loaded dynamically based on requirement. Those modules can be separate for each workflow and can be loaded based on requirement. They have a special function which will be called in the central engine. The received data in the central engine and the previous node’s ID are passed to the function and it returns a list of nodes to which the data has to be sent.

### 3.3.3 Flow Mutation

If a node is participating in multiple workflows, then corresponding to each one, there should be a piece of logic inside the node. This increases the complexity of a node and its scalability is at stake. However, in our architecture, the *postman* level orchestrator is upgraded to a *coordinator*. The coordinator not only looks at the message, it also modifies it as required (Figure 3.4). The node becomes so simple that, a typical user interface node needs to just render action fields to a user. This enables rendering itself as a service.

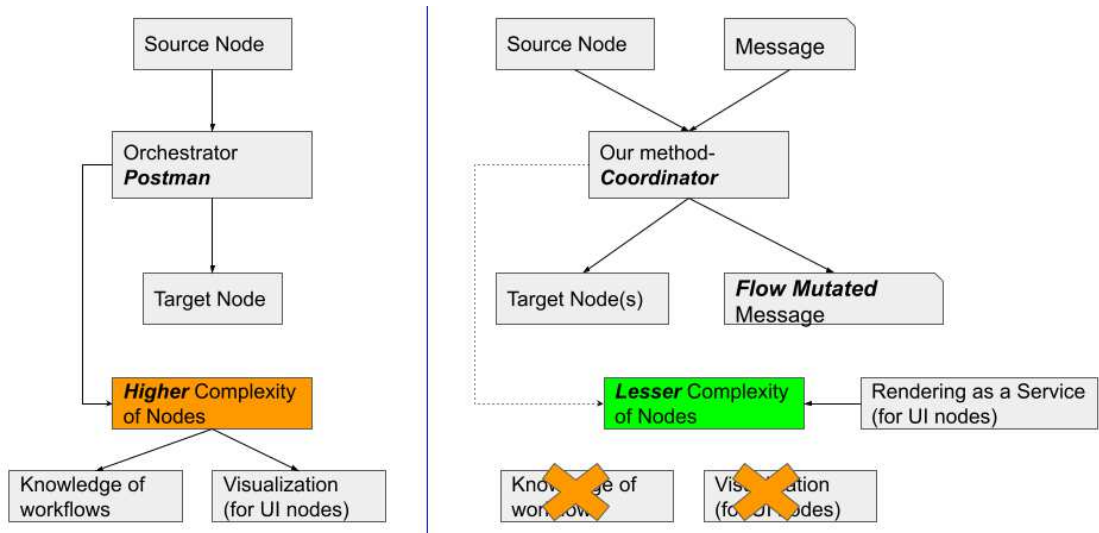


Figure 3.4: A depiction of the **flow mutation** concept. The present workflow systems consider a router as a *postman* type there by leading to higher node complexity. The figure depicts a complex node coupling with routing logic and rendering. The figure depicts the effect of decoupling a node with routing logic and making it a *coordinator* type and allowing flow mutation. A user interface node is processed by the concept of *rendering as a service* where interface itself is dynamically generated.

In the formal representation, the orchestration logic can modify the content leading to plug-n-play of workflows. In the proposed architecture there are three main aspects: (i) routing logic modifies the message, (ii) rendering as a service and (iii) two types of nodes. These aspects combined with micro services framework and plug-n-play workflow and node models, lead to a highly customizable workflow system.

### 3.3.4 Node Reuse

The nodes in the system are like functions in a programming language. Reusable. Because a node does its job irrespective of the previous node and the next, a node can be included in a number of workflows simultaneously. All the jobs will be carried out one-by-one or in parallel if the node is a compute node and can be multi threaded.

### 3.3.5 Rendering as a Service

The service for rendering offers rich types of interface elements such as file upload, text boxes, radio buttons, check boxes and submit buttons. These interaction elements can be dynamically added as the message flows through the workflow system. Hence there



is a mechanism where each message comes with its own rendering template as an attribute for consumption by graphical nodes

### **3.3.6 Distributed Nature of Workflow System**

The system itself is highly distributed (Figure 3.5). On a single computer, there can be several instances of workflow system. On multiple computers, the multitude of workflow system instances can talk to each other. The inter communication happens between a local and a remote system through use of *transfer workflows*. The transfer workflows remit messages in a receiver node dedicate in each workflow system instance for receipt of message.

### **3.3.7 User-Interface Node**

User should be able to view and edit the job information in a graphic user interface. Information gathered should be parsed and sent to workflow engine.

### **3.3.8 Computational Node**

The workflow should be able to do computation without user interaction. Computational nodes are used in machine learning scenario. Refer chapter 5 5 for use cases on computational nodes.

### **3.3.9 Web-API**

All the nodes communicate over HTTP via REST API, which makes a nice abstraction of resources. The entire information that flows will be in JSON making it really easy and light-weight to store and process.

### **3.3.10 Light Weight**

The system is very lightweight can be executed on low end devices such as raspberry pi in addition to server grade execution. (20 lines of code,less than 1MB of RAM required).

The user interface is flexible where rendering can also be obtained from other systems. The proof of concept implementation is light weight which enables *edge IoT devices to run mini workflows and act as mini nodes and participate in a larger workflow system*

### **3.3.11 IoT Enabled**

In today's modern world with a wide variety of scenarios, IoTs have become an integral part of business processes. They collect data from the physical world and help us take actions accordingly or may be give it certain instructions to do an operation. Surveillance cameras, TVs, air conditioners and projectors., have become internet-connected. Including IoTs in workflows could be quite useful in many use cases. 'Smart Campus' and 'Smart City' are two of the biggest use cases for workflows with IoTs. Traffic management in a city and academic activities in a University can be simplified with a workflow which has support for IoTs. Unfortunately, not all workflow systems in the market support IoT devices. And since the proposed system does not differentiate between nodes, IoTs can be seamlessly integrated in any workflow.

### **3.3.12 Communication Security**

Security can be an issue while transferring information. But as in the proposed architecture, all the communication can happen on top of HTTPS which makes it secure. Also, explicit encryption can also be done at the nodes and transfer the information.

### **3.3.13 Anonymity of Nodes**

One of the most important features of the proposed engine is the anonymity of the nodes. The nodes are anonymous to each other as they don't interact with each other. A node receives information from the central engine and irrespective of the previous node, this node does its job and sends its response back to the central engine. In this way, no node will know the existence of other nodes.

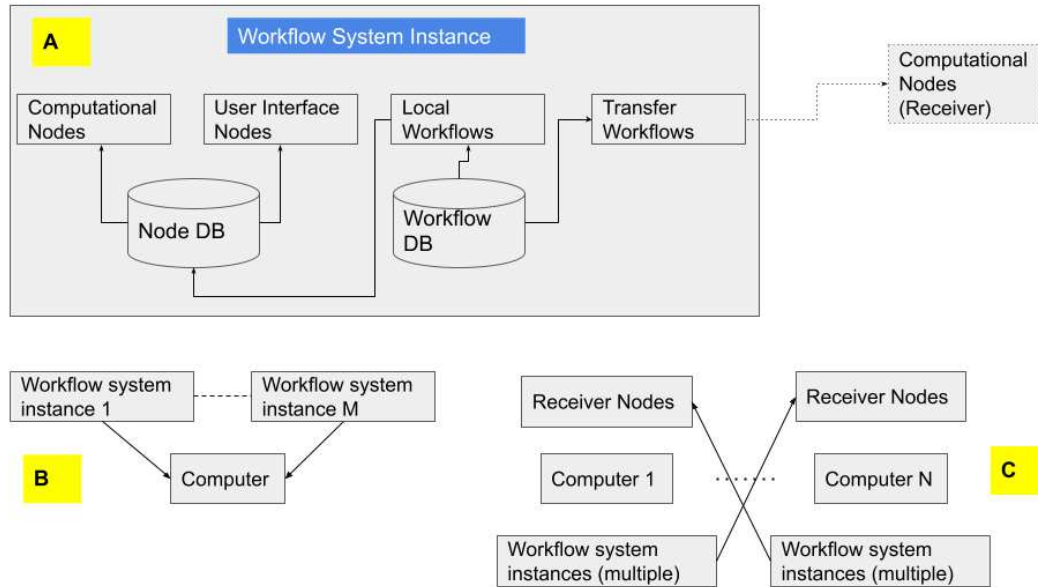


Figure 3.5: A highly distributed workflow system is depicted here. (A) Denotes the workflow system instance which as nodes of two types computational and interaction type, local workflows and a transfer workflow. It has databases for nodes and workflows. (B) Denotes one computer having multiple workflow system instanced deployed and running simultaneously. (C) Denotes a scenario of multiple computers, each with a multitude of workflow system instances and inter communicating via transfer workflows and receiver nodes.

### 3.4 Proof of concept using Python and Flask

The formalism is implemented in Python environment using Flask libraries for micro services. A schematic view of the framework is shown in (Figure 3.6). Rendering service is provided as a function inside views.py file. Templates are stored in a template database, out of which one selected and loaded by the rendering service. The render service uses Flask render\_template API to generate HTML by processing input JSON objects. The file process\_wf.py executes routing logic, which is the workflow engine. It can load on the fly routing codes. The exclusive schematic for workflow engine is shown in (Figure 3.7). The process\_wf.py fetches the routing codes and the corresponding services functions and modifies the jobs and messages. The modified jobs are deleted from workflow database and inserted into the node database.

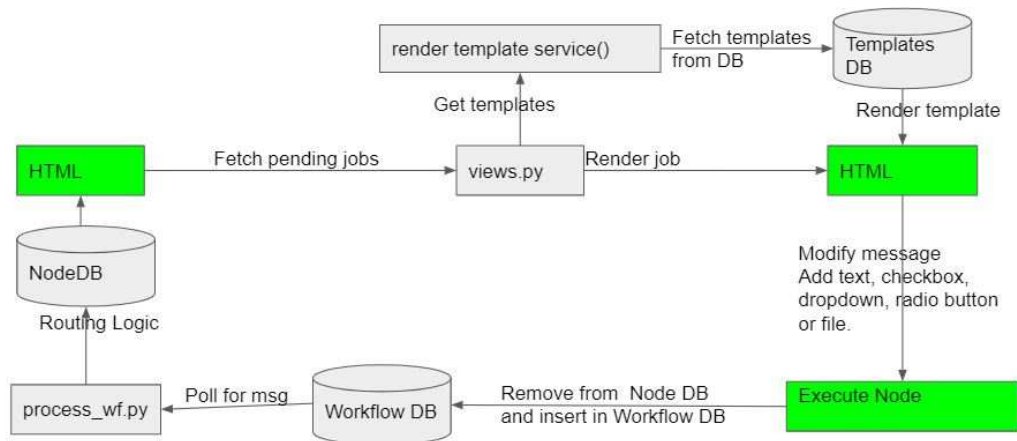


Figure 3.6: This figure depicts modules in the proof of concept system built using python and flask libraries. The user interaction components are shown in green colour. The HTML box corresponds to user visualization and interaction with graphical elements. Databases for node, workflow and templates are shown as cylinders. Arrows indicate flow of data or next steps. The annotations on top of arrows denote specific relationships.

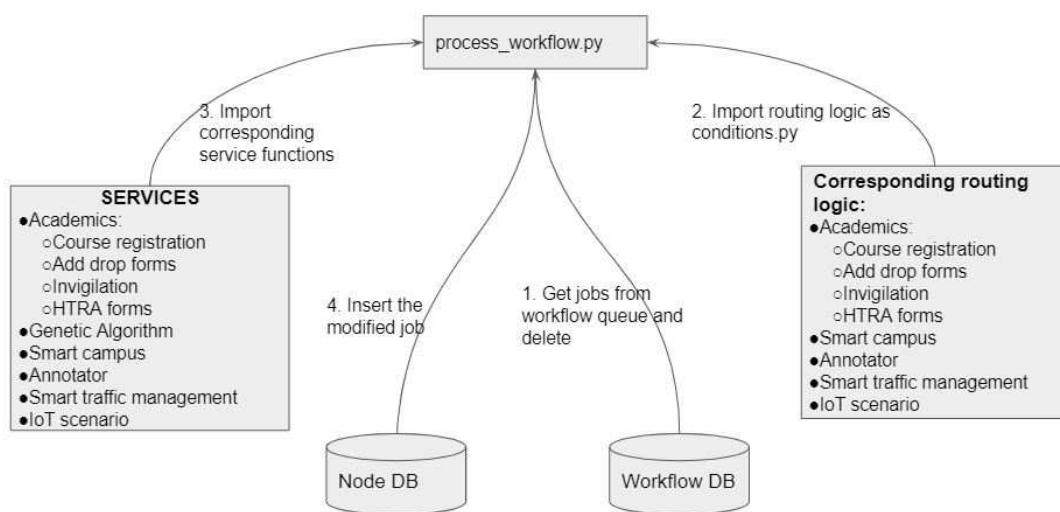


Figure 3.7: This figure depicts the process\_wf.py module built using python. Databases for node and workflow are shown as cylinders

# CHAPTER 4

## Results - Course Registration/AddDrop Forms

### 4.1 About this Chapter

The proof of concept and the core architecture discussed previously are verified in this and the subsequent two chapters by demonstrating various use cases incorporating both user nodes and computational nodes. Three types of use cases are shown here. They are:

1. Academic use cases involving processes course registration, adding or dropping courses, exam invigilation duties, half-time teaching and research assistantship forms. Implementing such use cases is part of the Smart Campus System, making such institute processes seamless.
2. A simple general use case to illustrate adding dynamic data to the user node.
3. Data Generation involving computation nodes.

The proposed system works on top of HTTPS, a secure encrypted connection over the internet, to eliminate any security issues. The proposed system implements RESTful web services, and every node will act as a micro service and irrespective of the technology used at the node. Since the REST paradigm enforces the abstraction of resources, nodes can communicate with the central engine without any problem. All the communication occurs in JSON form, a universally accepted format for data sharing, which is also lightweight for processing.

### 4.2 Course Registration

This use case involves an example of a smart campus scenario where students enrol for courses, the sequence is depicted in fig. 4.1.

First, the academic section need to login user their username and password as shown in fig. 4.2. The corresponding dashboard is shown in fig. 4.3. On the right, there is a button called as Add Job. On clicking, various templates are displayed. These templates are accessible only to the academics. Click on course registration as shown in fig. 4.4.

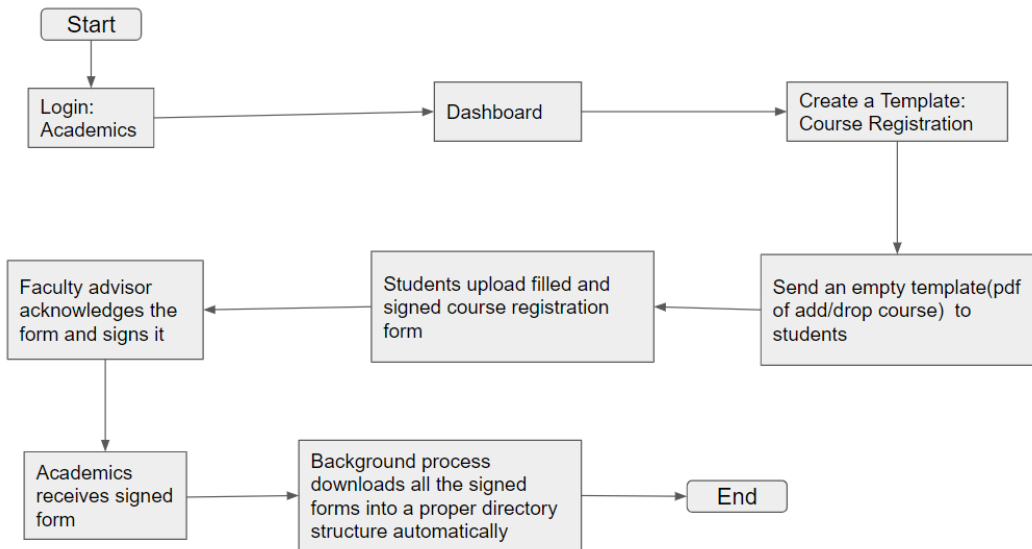


Figure 4.1: The figure depicts flow of steps in the scenario for course registration in an academics scenario.

In this job which is created, academics should give a job name, branch and an empty course registration form as shown in fig. 4.5. On clicking the submit button, the job is sent to all the students.

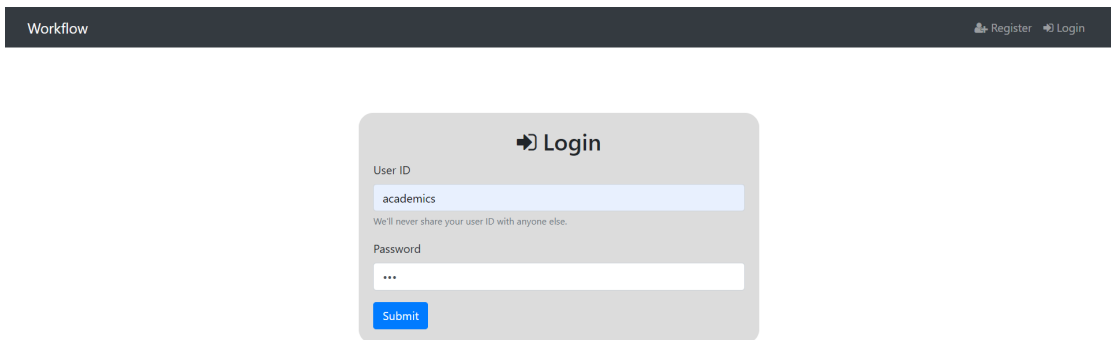


Figure 4.2: The figure depicts the login page with academics credentials.

Now, students can login and see course registration job fig. (4.6) and the empty course registration form that the academics have sent to them fig. (4.7). Students can fill this PDF form with the necessary details along with their signature. On submit, the job will be sent to that corresponding faculty advisor.

Now, the faculty advisor checks the PDF form signed by the student fig. (4.8), signs it and executes the job. This job is sent to academics fig. (4.9). The student gets a notification that the faculty advisor has sent this job to the academics.

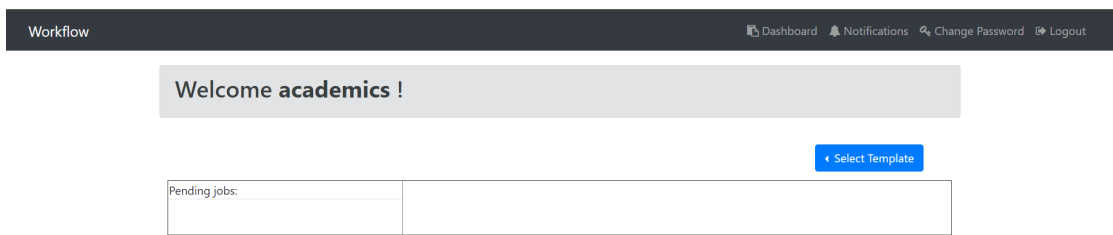


Figure 4.3: The figure depicts the dashboard of academics.

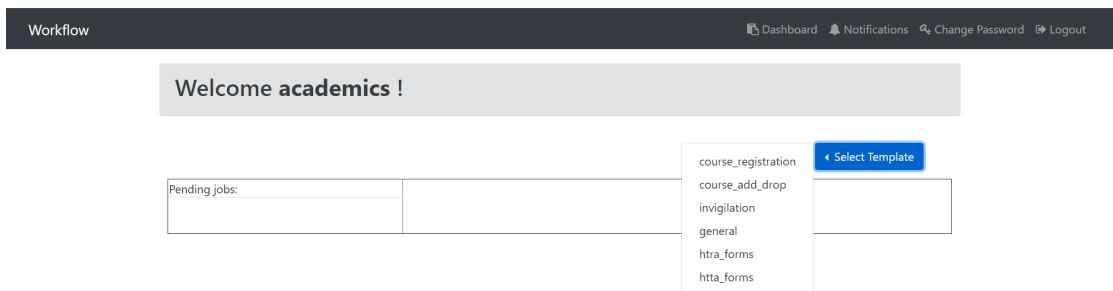


Figure 4.4: The figure depicts the templates available for academics.

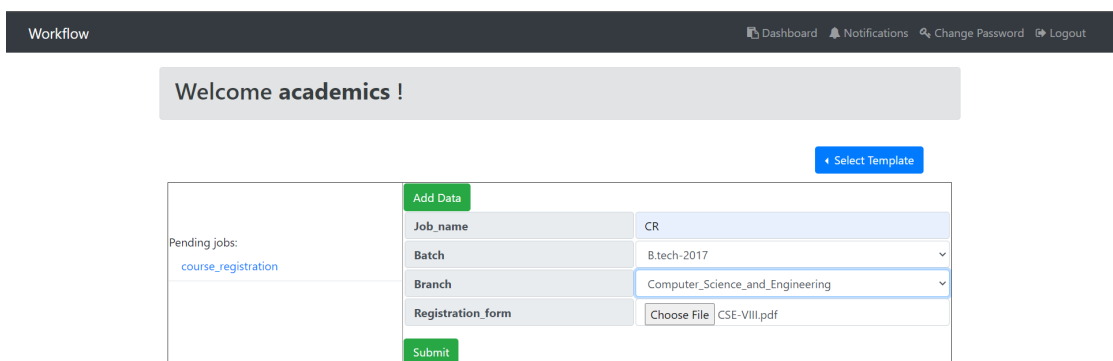


Figure 4.5: The figure depicts the rendered job for academics. Academics has entered the job name and uploaded the empty Course Registration form.

Workflow Dashboard Notifications Change Password Logout

Welcome B123 ! Select Template

Pending jobs: course\_registration CR

<span style="background-color: #28a745; color: white; padding: 2px 5px;">Add Data</span>	
Batch	B.tech-2017
Branch	Computer_Science_and_Engineering
Registration_form	CSE-VIII.pdf
Rollno	B123
Student_name	Abc
Faculty_advisor	advisor1
Unsigned_form	<span style="border: 1px solid #ccc; padding: 2px;">Choose File</span> No file chosen
<span style="background-color: #28a745; color: white; padding: 2px 5px;">Submit</span>	

Figure 4.6: The figure depicts the rendered course registration job in the Student Dashboard.

An additional service node, `acads_bg.py` can be used to download all the files in a structured directory as shown fig. (4.10).

### 4.3 Add/Drop Course

This use case involves an example of a smart campus scenario where students enrol for courses and later change their decision is depicted in fig. 4.11. The scenario is as follows:

First, the academic section logs in using their username and password. The corresponding dashboard is displayed. The academics can add a new job for add/drop course from the preexisting templates as shown in fig. 4.12. In this template job, the academics can enter the job name and upload an empty add/drop course PDF form as shown in fig. 4.13. On submit, the job is sent to all the students. The message after submit is as shown in fig. 4.14. There is a database collection containing all the student details like name, roll number, faculty advisor, branch, year, batch. Now, students can login and see add/drop course job (fig. 4.15) and the empty add/drop form that the academics have sent to them (fig. 4.16). Students can fill this PDF form with the details of courses, corresponding instructors and other information along with their signature. The students also have to enter the course ID for which the corresponding instructor's signature is required and upload the filled form as shown in fig. 4.17. On submit, the job will be sent to that corresponding instructor.





Email: academics@iittp.ac.in

**Indian Institute of Technology Tirupati**  
Renigunta Road, Tirupati - 517 506, A.P.

**ACADEMIC SECTION**

Phone: 0877-2503531

**Course Registration Form**

Jan-June 2021 Semester

Roll No: \_\_\_\_\_ Branch: Computer Science and Engineering  
Name : \_\_\_\_\_ Semester: VIII

S.No	Slot	Course No	Name	Category	Credits
1.		DPE3	Department Elective-3	PMT	3
2.	G	HS3050	Professional Ethics	HPE	2
<b>Total</b>					

Student Signature

Date:

Faculty Advisor

Signature:

Name:

Figure 4.7: The figure depicts the empty course registration form sent by academics to the student.

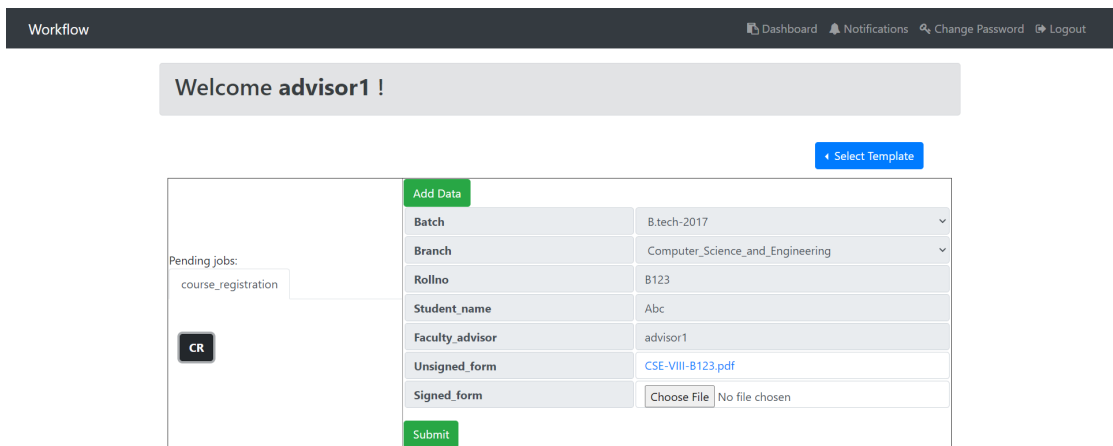


Figure 4.8: The figure depicts the rendered course registration job in the Faculty Advisor Dashboard, containing the form filled by student.

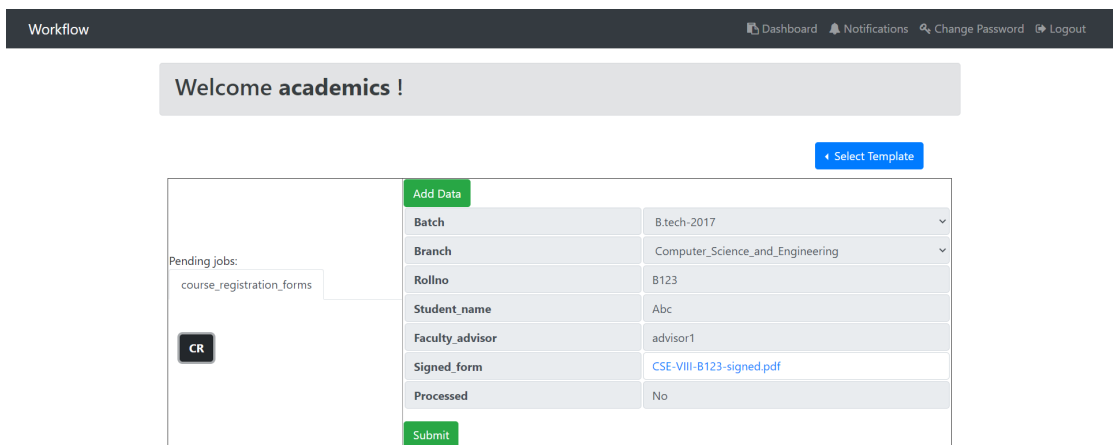


Figure 4.9: The figure depicts the rendered course registration job in the Academics Dashboard, containing the form filled by student and acknowledged(signed) by Faculty Advisor.

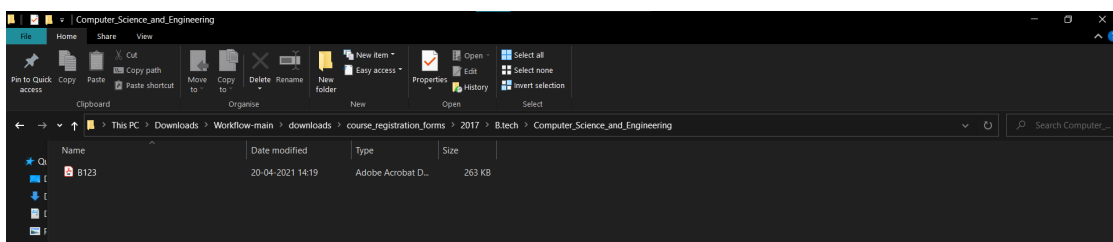


Figure 4.10: The figure depicts the directory where all the signed course registration form is downloaded automatically.

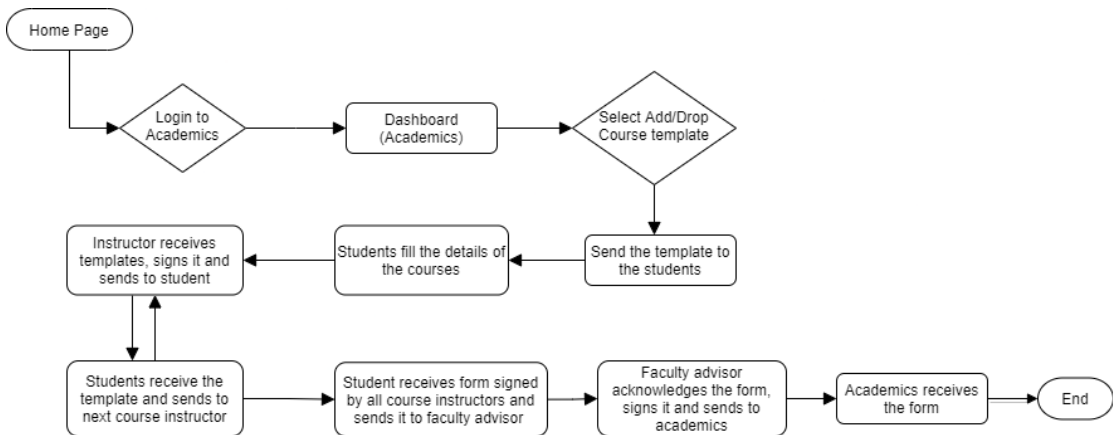


Figure 4.11: The figure depicts flow of steps in the scenario for course add and drop in an academics scenario.

Welcome academics !

[Select Template](#)

Pending jobs:	<a href="#">Add Data</a>
course_add_drop	<input type="text" value="Job_name"/>
<a href="#">course_add_drop_1</a>	<input type="text" value="Add_drop_form"/> <input type="button" value="Choose File"/> No file chosen
	<a href="#">Submit</a>

Figure 4.12: Figure showing adding a new job for add/drop course template

Welcome academics !

[Select Template](#)

Pending jobs:	<a href="#">Add Data</a>
course_add_drop	<input type="text" value="Job_name"/> March 2021
<a href="#">course_add_drop_1</a>	<input type="text" value="Add_drop_form"/> <input type="button" value="Choose File"/> Add_Drop_Courses.pdf
	<a href="#">Submit</a>

Figure 4.13: Figure showing academics entering the job name and uploading PDF form just before sending to students

Welcome academics !

[Select Template](#)

Pending jobs:	The job has been processed
---------------	----------------------------

Figure 4.14: Figure showing a message after submit and job is processed

Welcome **B123** !

[← Select Template](#)

Pending jobs:

course\_add\_drop

**March 2021**

Add Data

Course_id	
Add_drop_form	Add_Drop_Courses.pdf
Send_to_faculty_advisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Rollno	B123
Student_name	Abc
Faculty_advisor	advisor1
Unsigned_add_drop	<input type="button" value="Choose File"/> No file chosen

Submit

Figure 4.15: Figure showing student's dashboard

file
1 / 1 | - 75% + | 🖨️ ↻

**Indian Institute of Technology Tirupati**  
Renigunta Road, Tirupati – 517 506

**Course Add/Drop Form**

Roll No & Name :  
 Branch :  
 Semester & Year :  
 Name of the Faculty Advisor :

Add			
Course No	Course Name	Faculty Name	Faculty Signature

Drop			
Course No	Course Name	Faculty Name	Faculty Signature

Signature of Faculty Advisor  
Date:

Signature of the Student  
Date:

Figure 4.16: Figure showing empty add-drop course PDF form

Welcome B123 !

[Select Template](#)

Pending jobs:

course\_add\_drop

**March 2021**

Add Data	
Course_id	CS123
Add_drop_form	<a href="#">Add_Drop_Courses.pdf</a>
Send_to_faculty_advisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Rollno	B123
Student_name	Abc
Faculty_advisor	advisor1
Unsigned_add_drop	<input type="button" value="Choose File"/> Add_Drop_Courses_B123.pdf
<b>Submit</b>	

Figure 4.17: Figure showing student’s dashboard with job just before sending to course instructor

The instructor can now login and see the pending add/drop jobs from the students(fig. 4.18). The instructor can verify the PDF form signed by the student(fig. 4.19) and up-

Welcome instructor\_1 !

[Select Template](#)

Pending jobs:

course\_add\_drop

**March 2021**

Add Data	
Course_id	CS123
Course_name	course_1
Course_instructor	instructor_1
Rollno	B123
Student_name	Abc
Faculty_advisor	advisor1
Unsigned_add_drop	<a href="#">Add_Drop_Courses_B123.pdf</a>
Signed_add_drop	<input type="button" value="Choose File"/> No file chosen
<b>Submit</b>	

Figure 4.18: Figure showing course instructor’s dashboard

load their form after signing(fig. 4.20). On execute, this job will be again sent to the student. The student can now see the job and(fig. 4.21) only verify the add/drop PDF form(fig. 4.22). After verification, the student can now edit the course ID again and send this form to another instructor as needed. It is to be noted that the student can send this form to his/her faculty advisor at any time. By default, this option is given as 'NO'. After all the course instructor(s) sign the add/drop course PDF form, the student can now select 'YES' for sending it to the faculty advisor. Let’s say the student needs

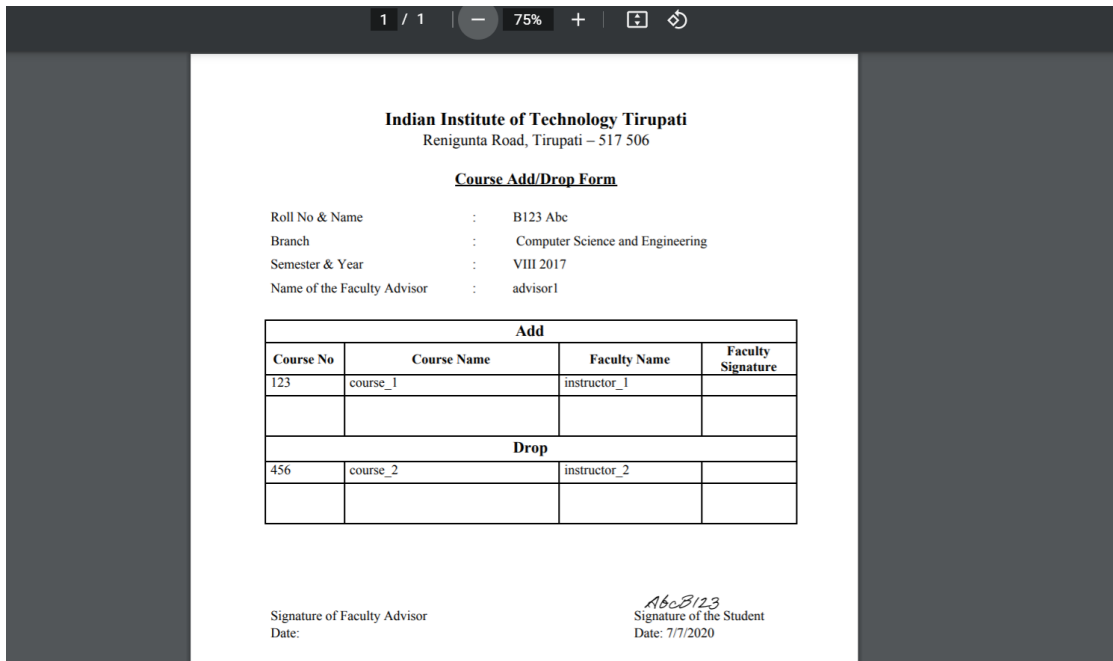


Figure 4.19: Figure showing PDF form sent by the student to the course instructor

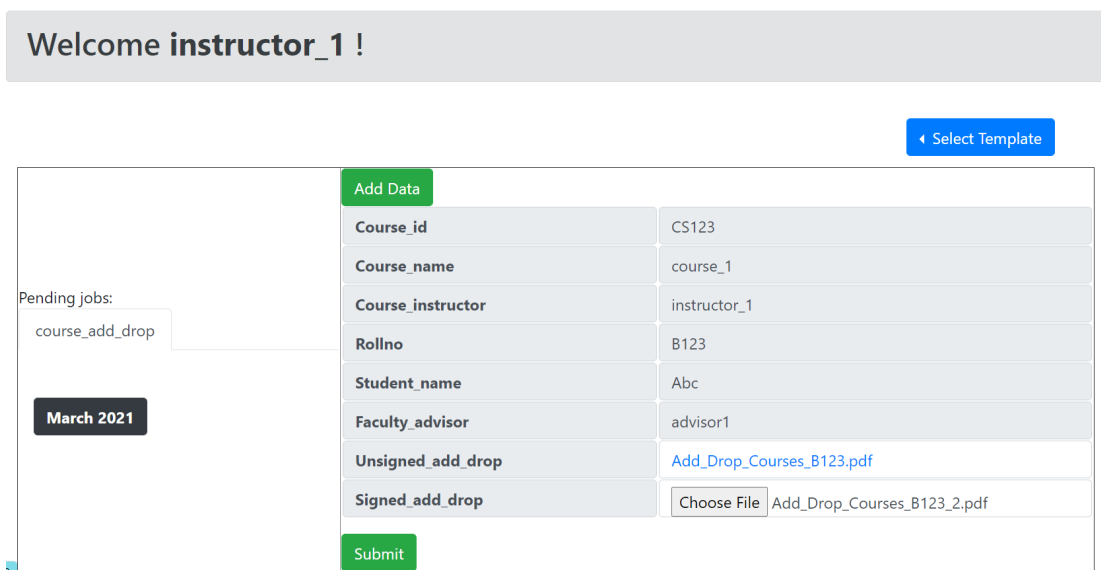


Figure 4.20: Figure showing course instructor's dashboard before sending to student

Welcome **B123 !**

[Select Template](#)

Pending jobs:

course\_add\_drop

**March 2021**

Add Data

Course_id	CS123
Add_drop_form	<a href="#">Add_Drop_Courses.pdf</a>
Send_to_faculty_advisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Rollno	B123
Student_name	Abc
Faculty_advisor	advisor1
Signed_add_drop	<a href="#">Add_Drop_Courses_B123_2.pdf</a>

Submit

Figure 4.21: Figure depicting student dashboard with job received from the first course instructor

1 / 1 | 80% +

**Indian Institute of Technology Tirupati**  
Renigunta Road, Tirupati – 517 506

**Course Add/Drop Form**

Roll No & Name : B123 Abc  
 Branch : Computer Science and Engineering  
 Semester & Year : VIII 2017  
 Name of the Faculty Advisor : advisor1

Add			
Course No	Course Name	Faculty Name	Faculty Signature
123	course_1	instructor_1	<i>instructor1</i>
Drop			
456	course_2	instructor_2	

Signature of Faculty Advisor  
Date:

*AbcB123*  
Signature of the Student  
Date: 7/7/2020

Figure 4.22: Figure showing PDF form filled by one course instructor

the signature of another course instructor. The student changes the course ID, sends to instructor, the instructor verifies, signs and on submit, the job goes to student again(fig. 4.23). This can be repeated many times as shown in fig. 4.11. Finally, student selects

Welcome **B123 !**

[Select Template](#)

Pending jobs:

course\_add\_drop

**March 2021**

Add Data	
Course_id	EE456
Add_drop_form	<a href="#">Add_Drop_Courses.pdf</a>
Send_to_faculty_advisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Rollno	B123
Student_name	Abc
Faculty_advisor	advisor1
Signed_add_drop	<a href="#">Add_Drop_Courses_B123_2.pdf</a>
<a href="#">Submit</a>	

Figure 4.23: Figure showing the student dashboard with job received from course instructor

'YES' and submits to faculty advisor(fig. 4.24).

Welcome **B123 !**

[Select Template](#)

Pending jobs:

course\_add\_drop

**March 2021**

Add Data	
Course_id	EE456
Add_drop_form	<a href="#">Add_Drop_Courses.pdf</a>
Send_to_faculty_advisor	<input checked="" type="radio"/> Yes <input type="radio"/> No
Rollno	B123
Student_name	Abc
Faculty_advisor	advisor1
Signed_add_drop	<a href="#">Add_Drop_Courses_B123_3.pdf</a>
<a href="#">Submit</a>	

Figure 4.24: Figure depicting student dashboard with sending to faculty advisor radio button changed to 'YES'

Finally, the faculty advisor logs in(fig. 4.25), checks the PDF form signed by the student(fig. 4.26) and the course instructor(s), signs it, uploads the form and submits the job. This job is sent to academics. Under the role, add/drop course forms, academics



Welcome advisor1 !

[Select Template](#)

Pending jobs:

course\_add\_drop

**March 2021**

Add Data

<b>Rollno</b>	B123
<b>Student_name</b>	Abc
<b>Faculty_advisor</b>	advisor1
<b>Unsigned_add_drop</b>	<a href="#">Add_Drop_Courses_B123_3.pdf</a>
<b>Signed_faculty_advisor</b>	Choose File No file chosen

Submit

Figure 4.25: Figure showing faculty advisor's dashboard

1 / 1 | 80% | [Zoom In] [Zoom Out]

**Indian Institute of Technology Tirupati**  
Renigunta Road, Tirupati – 517 506

**Course Add/Drop Form**

Roll No & Name : B123 Abc  
 Branch : Computer Science and Engineering  
 Semester & Year : VIII 2017  
 Name of the Faculty Advisor : advisor1

Add			
Course No	Course Name	Faculty Name	Faculty Signature
123	course_1	instructor_1	<i>instructor1</i>
Drop			
456	course_2	instructor_2	<i>instructor2</i>

Signature of Faculty Advisor  
Date:

*AbcB123*  
Signature of the Student  
Date: 7/7/2020

Figure 4.26: Figure showing PDF form filled by two course instructors

can see all the students' forms as shown in fig. 4.27. The PDF file received is completely

Welcome academics !

[← Select Template](#)

Pending jobs: add_drop_forms  <b>March 2021</b>	<b>Add Data</b>	
	<b>Rollno</b>	B123
	<b>Student_name</b>	Abc
	<b>Faculty_advisor</b>	advisor1
	<b>Signed_faculty_advisor</b>	<a href="#">Add_Drop_Courses_B123_4.pdf</a>
	<b>Processed</b>	No
	<b>Submit</b>	

Figure 4.27: Figure showing jobs received from students by academics

filled(fig. 4.28). The student can view all the notifications to get acknowledged that the faculty advisor has sent this job to the academics. This is shown in fig. 4.29. Similar to course registration, An additional service node, acads\_bg.py can be used to download all the files in a structured directory.

## Indian Institute of Technology Tirupati

Renigunta Road, Tirupati – 517 506

### Course Add/Drop Form

Roll No & Name : B123 Abc  
Branch : Computer Science and Engineering  
Semester & Year : VIII 2017  
Name of the Faculty Advisor : advisor1

Add			
Course No	Course Name	Faculty Name	Faculty Signature
123	course_1	instructor_1	<i>instructor1</i>
Drop			
456	course_2	instructor_2	<i>instructor2</i>

*advisor1*  
Signature of Faculty Advisor  
Date:

*AbcB123*  
Signature of the Student  
Date: 7/7/2020

Figure 4.28: Figure showing completely filled add/drop form

### Notifications for B123

Timestamp	Notification	
20-04-2021 10:49:07	March 2021 (course_add_drop) sent to academics	Delete
20-04-2021 10:47:57	March 2021 (course_add_drop) sent to advisor1	Delete
20-04-2021 10:46:26	March 2021 (course_add_drop) sent to instructor_2	Delete
20-04-2021 10:42:36	March 2021 (course_add_drop) sent to instructor_1	Delete

Figure 4.29: Figure showing Student Notifications

# CHAPTER 5

## Results - HTTA/HTRA Forms

### 5.1 HTTA/HTRA forms

This use case involves an example of a smart campus scenario where students fill the HTRA/HTTA form, the sequence is depicted in Fig. 5.1.

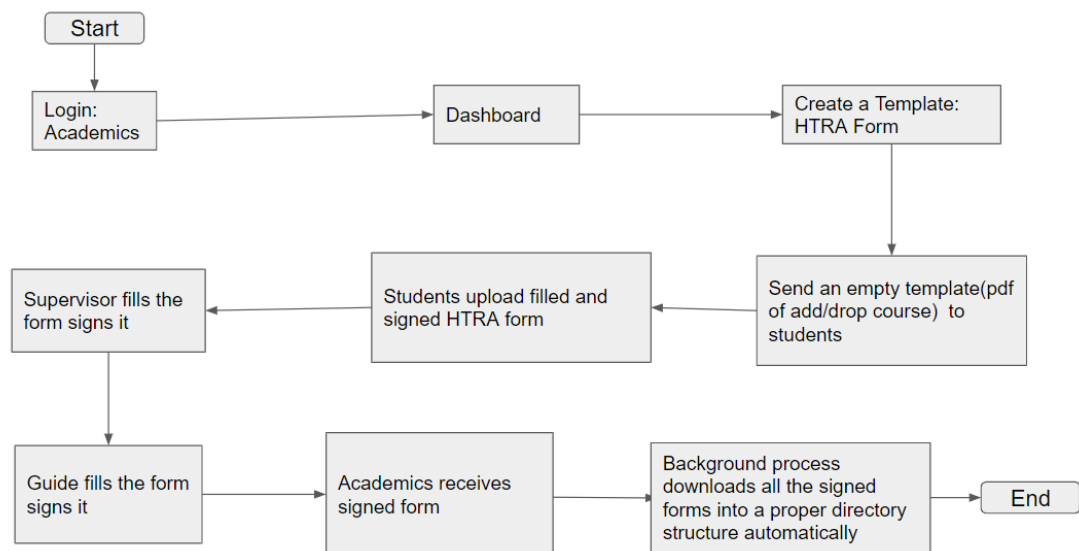


Figure 5.1: The figure depicts flow of steps in the scenario for HTRA form in an academics scenario.

First, the academic section need to login user their username and password. The corresponding dashboard. On the right, there is a button called as Add Job. On clicking, various templates are displayed. These templates are accessible only to the academics. Click on HTRA as shown in Fig. 5.2. In this job which is created, academics should give a job name and an empty HTRA form as shown in Fig. 5.3. On clicking the submit button, the job is sent to all the scholars.

Now, the scholars can login and see the HTRA job (Fig. 5.4) and the empty HTRA form that the academics have sent to them (Fig. 5.5). Students can fill this PDF form with the necessary details along with their signature. On submit, the job will be sent to that corresponding TA Supervisor.

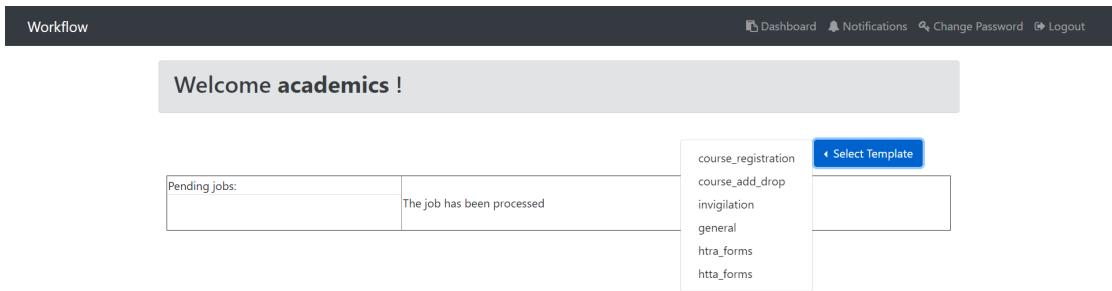


Figure 5.2: The figure depicts the templates available for academics.

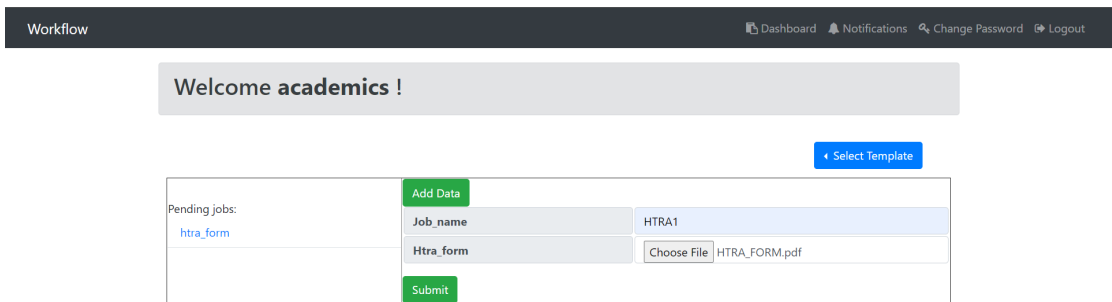


Figure 5.3: The figure depicts the rendered job for academics. Academics has entered the job name and uploaded the empty HTRA form.

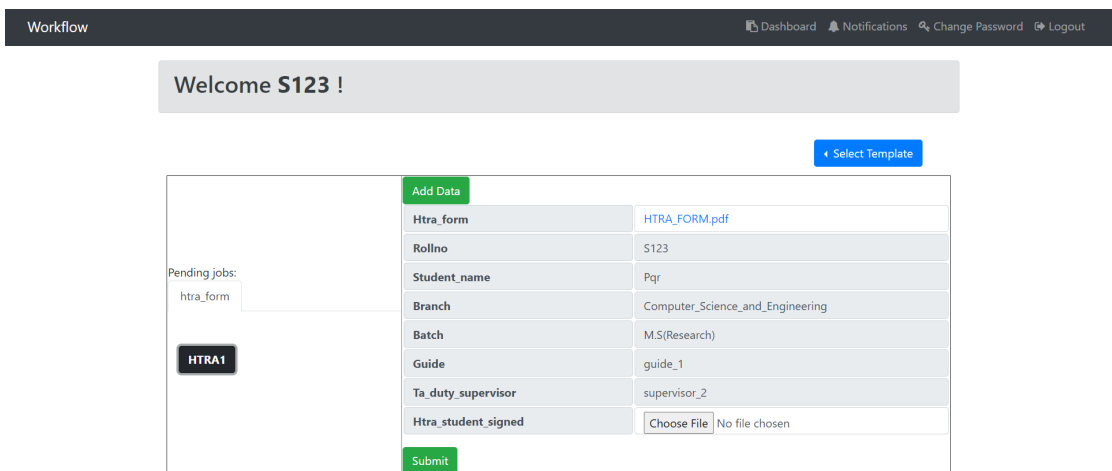


Figure 5.4: The figure depicts the rendered HTRA job in the Student Dashboard.



Date: \_\_\_\_\_

### HTRA FORM

This is to certify that I, Mr/Ms. \_\_\_\_\_,

Roll No. \_\_\_\_\_ a Student of M.S(Research)/PhD in the Department of \_\_\_\_\_ at IIT Tirupati has actually worked for 8 hrs/week under HTRA work during the month of \_\_\_\_\_ 2021. Further, I am not receiving a stipend/fellowship from any other external source for the month of \_\_\_\_\_ 2021.

Signature of the Student: \_\_\_\_\_

**Guide**

Satisfactory  Partially Satisfactory  Not Satisfactory

Remarks: \_\_\_\_\_  
\_\_\_\_\_

Name & Signature of the faculty: \_\_\_\_\_

**T.A. Duty Supervisor**

Satisfactory  Partially Satisfactory  Not Satisfactory

Remarks: \_\_\_\_\_  
\_\_\_\_\_

Name & Signature of the faculty: \_\_\_\_\_

Note: The T.A and Thesis Supervisor are requested to put a tick mark in the appropriate box and provide remarks in case of Partially Satisfactory and Not Satisfactory.  
The form has to be submitted before 20<sup>th</sup> of each month

Figure 5.5: The figure depicts the empty HTRA form sent by academics to the student.

Now, the TA Supervisor checks the PDF form signed by the student (Fig. 5.6), signs it and executes the job. This job is sent to corresponding guide. The student gets a notification that the TA Supervisor has sent this job to the guide (Fig. 5.7).

Workflow Dashboard Notifications Change Password Logout

Welcome **supervisor\_2** !

[Select Template](#)

Pending jobs:  
htra\_form

[Add Data](#)

Rollno	S123
Student_name	Pqr
Branch	Computer_Science_and_Engineering
Batch	M.S(Research)
Guide	guide_1
Ta_duty_supervisor	supervisor_2
Htra_student_signed	<a href="#">HTRA_FORM_S123.pdf</a>
Htra_ta_supervisor_signed	<input type="button" value="Choose File"/> No file chosen

[Submit](#)

Figure 5.6: The figure depicts the rendered HTRA job sent by student to the TA supervisor. The supervisor can click on the PDF file to view the form.

Workflow Dashboard Notifications Change Password Logout

Notifications for **S123**

Timestamp	Notification	
21-04-2021 12:56:10	HTRA1 (htra_form) sent to guide_1	<a href="#">Delete</a>
21-04-2021 12:55:45	HTRA1 (htra_form) sent to supervisor_2	<a href="#">Delete</a>

Figure 5.7: The figure depicts the notifications of the student.

Now,guide can check the PDF form signed by the student and TA Supervisor (Fig. 5.8), signs it and executes the job. This job is sent to academics (Fig. 5.9). The student gets a notification that the guide has sent this job to the academics. An additional service node, acads\_bg.py can be used to download all the files in a structured directory as shown in Course Registration use case.

## 5.2 General

For a general template, let's consider a simple use case. There are three nodes: **nx**, **ny** and **nz**. The routing takes place as: **nx** → **ny** → **nz**. First, **nx** is logged in. Add a general template as shown in fig. 5.10.

Workflow Dashboard Notifications Change Password Logout

Welcome **guide\_1** !

[Select Template](#)

Pending jobs:

htra\_form

**HTRA1**

Add Data

Rollno	S123
Student_name	Pqr
Branch	Computer_Science_and_Engineering
Batch	M.S(Research)
Guide	guide_1
Ta_duty_supervisor	supervisor_2
Htra_ta_supervisor_signed	<a href="#">HTRA_FORM_S123_supervisor.pdf</a>
Htra_signed	<input type="button" value="Choose File"/> No file chosen

Submit

Figure 5.8: The figure depicts the rendered HTRA job sent by TA Supervisor to the guide. The guide can click on the PDF file to view the form.

Workflow Dashboard Notifications Change Password Logout

Welcome **academics** !

[Select Template](#)

Pending jobs:

HTRA\_forms\_collected

**HTRA1**

Add Data

Rollno	S123
Student_name	Pqr
Branch	Computer_Science_and_Engineering
Batch	M.S(Research)
Guide	guide_1
Ta_duty_supervisor	supervisor_2
Htra_signed	<a href="#">HTRA_FORM_S123_guide.pdf</a>
Processed	No

Submit

Figure 5.9: The figure depicts the job containing the completely filled HTRA Form.

Welcome **nx** !

[Select Template](#)

Pending jobs:

r2 r1 general

Add Data

Job_name	
Role	general

Submit

Figure 5.10: Figure showing dashboard of nx and adding a general template



Job name and role name can be added as required(fig. 5.11). On submit, the job

Welcome nx !

Select Template

Pending jobs:

r2 r1 general

Add Data

Job_name	sample job
Role	test_role

Submit

Figure 5.11: Figure showing entering job name and role name in general template

gets sent to node *ny*. Logging in to *ny*, the role and job are displayed.

Dynamic data which could be of text, dropdown, checkbox, radio button or file type, can be added using *Add Data* button. This is shown in fig. 5.12. As an example, text(fig.

orkflow

Notifications Change Password Log

Welcome ny !

Pending jobs:

test\_role

sample job

Select Template

Table Fields

Select the type of fields..

select from options

select from options

Text

Dropdown

Checkbox

Radio Button

Upload file

Figure 5.12: Figure depicting different fields of data that can be added

5.13) and drop down(fig. 5.14) are added as shown in fig. 5.15. Also, checkbox(fig. 5.16), radio(fig. 5.17), file(fig. 5.18) fields are added as shown in fig. 5.19. There is a remove button to remove the fields as needed. The fields are filled as shown in fig. 5.20 and on submit, the job is sent to *nz*. *nz* logs in and sees the job that has been sent by *ny*(fig. 5.21).

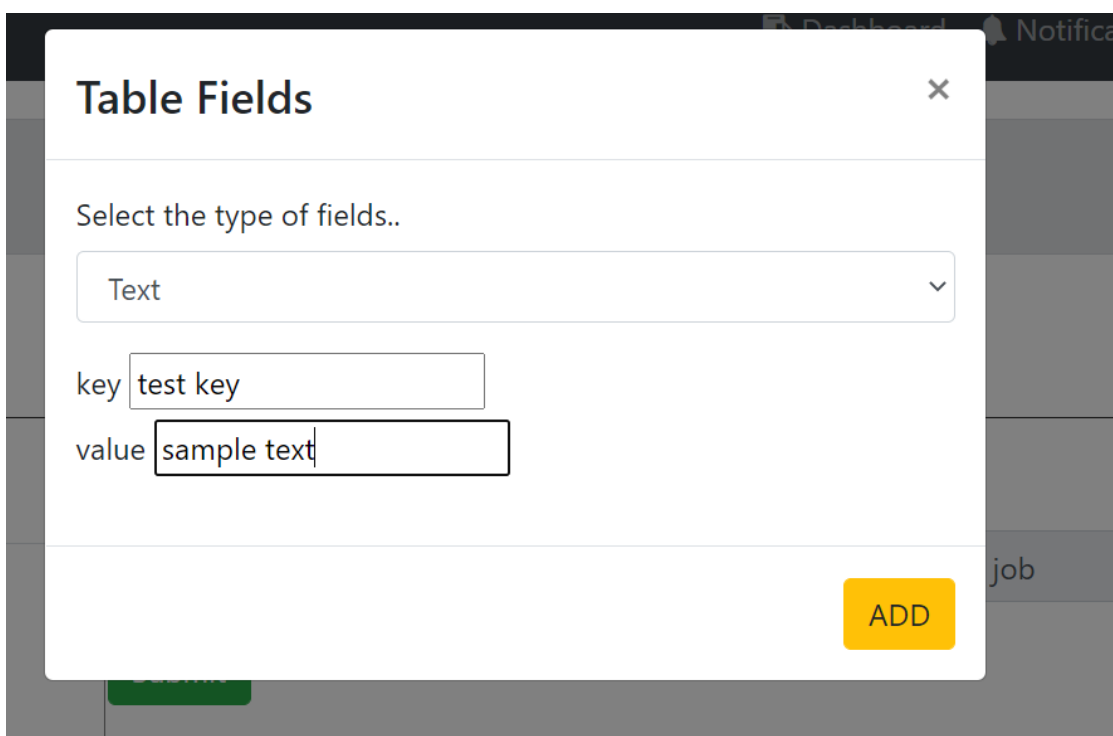


Figure 5.13: Figure depicting adding text field

**Table Fields** ×

Select the type of fields..

Dropdown ▾

Key

Values:

abc	<input type="button" value="Remove"/>
def	<input type="button" value="Remove"/>
ghi	<input type="button" value="Remove"/>

Figure 5.14: Figure depicting adding drop down field

<b>Job_name</b>	sample job
test key	sample text <input type="button" value="Remove"/>
test drop	abc <input type="button" value="Remove"/>

Figure 5.15: Figure showing job after adding text and drop down fields

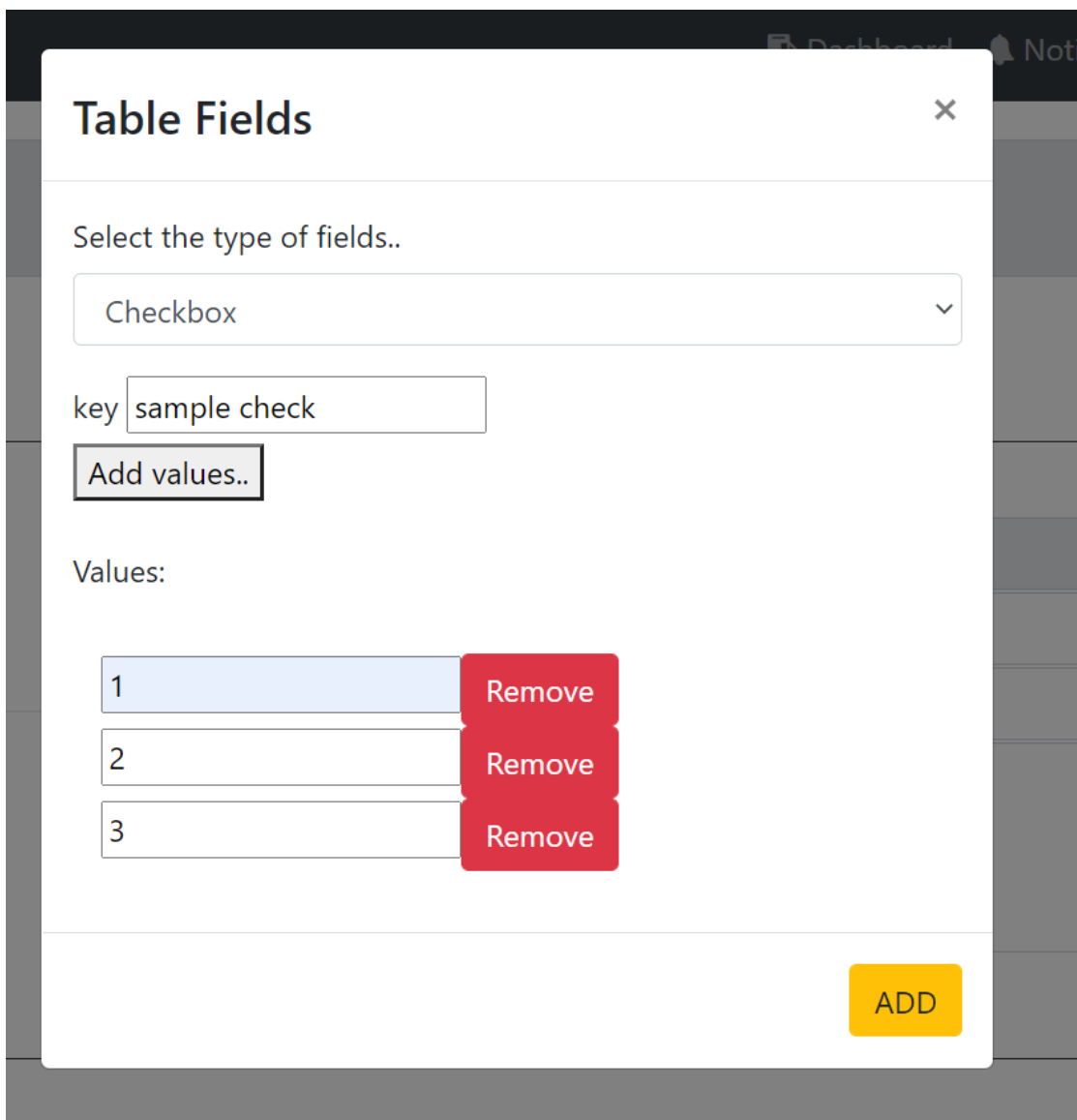


Figure 5.16: Figure depicting adding checkbox field

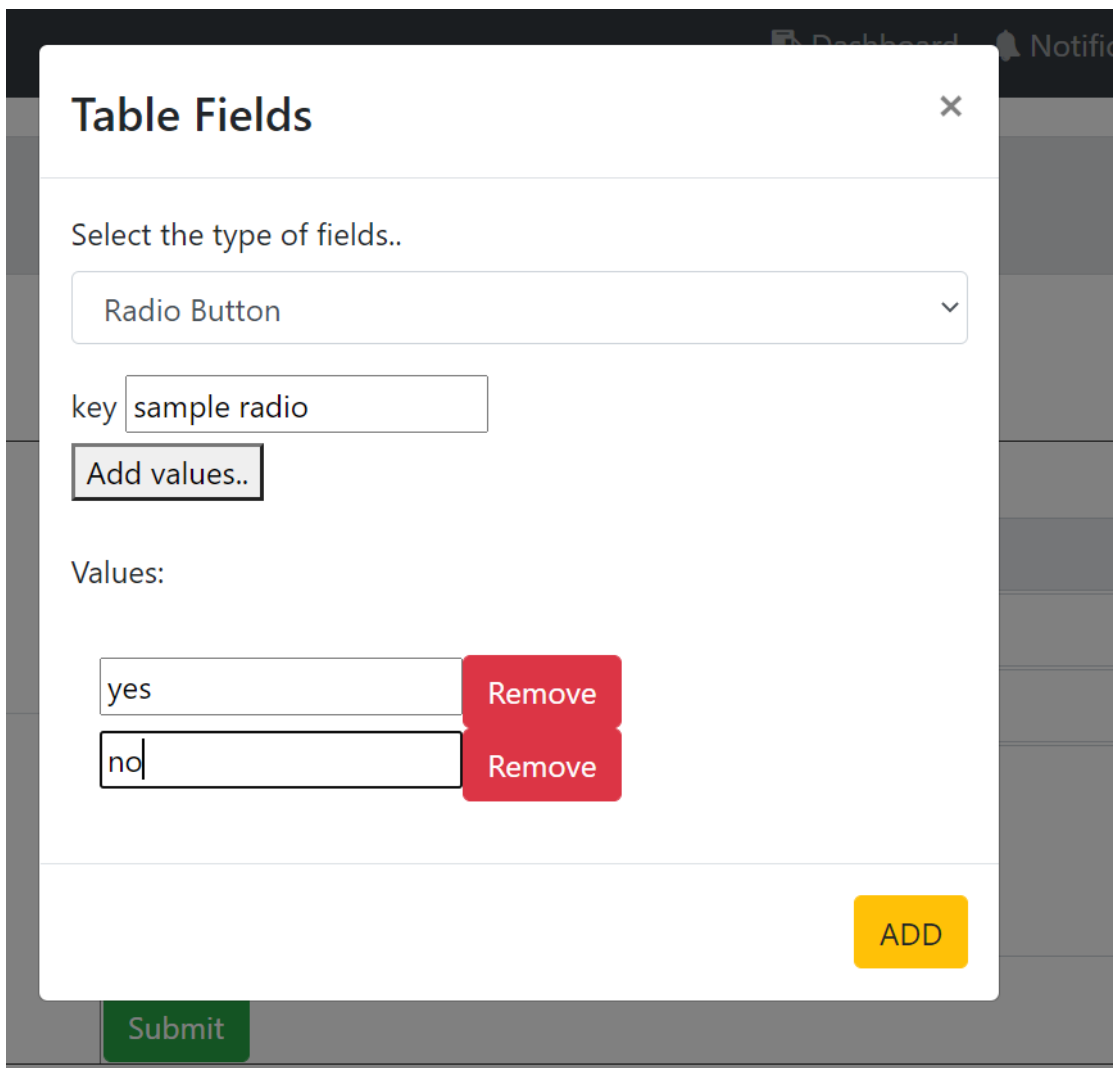


Figure 5.17: Figure depicting adding radio button field

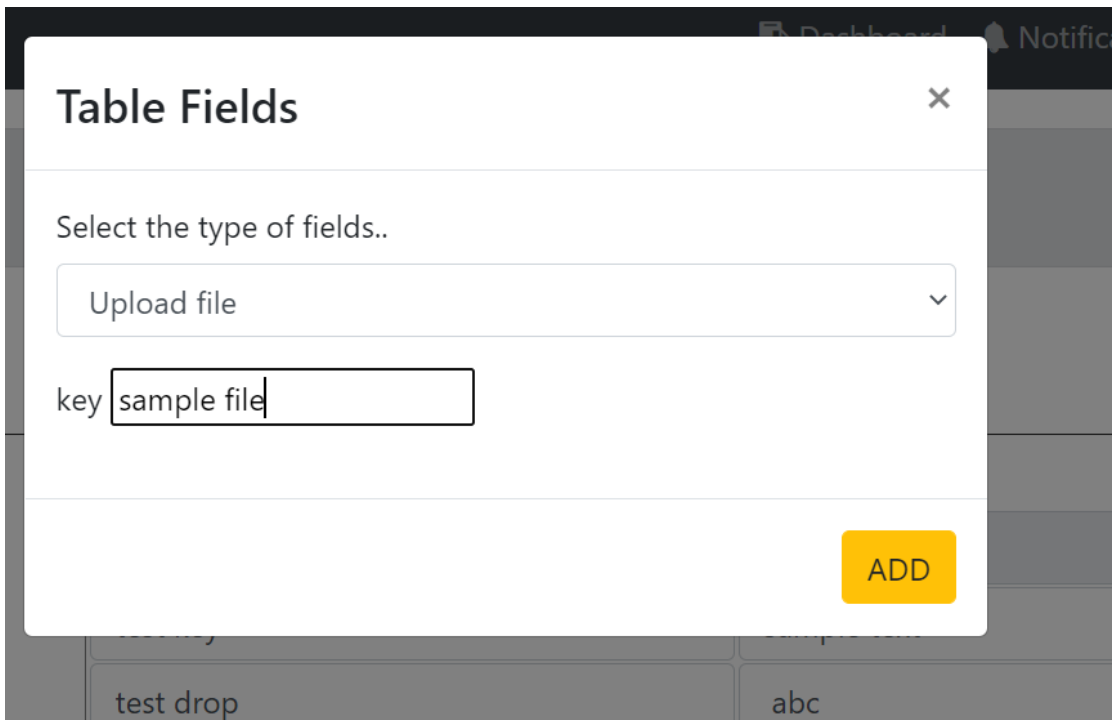


Figure 5.18: Figure depicting adding file upload field

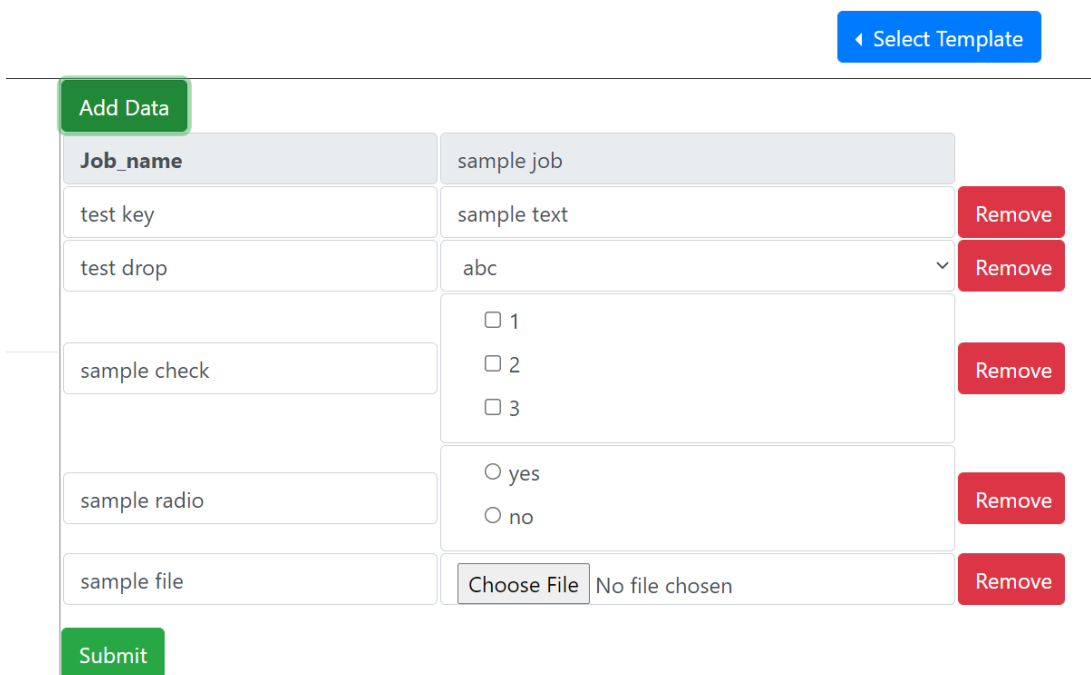


Figure 5.19: Figure showing job after adding all the possible fields

[← Select Template](#)

**Add Data**

<b>Job_name</b>	sample job	
test key	sample text	<a href="#">Remove</a>
test drop	def	<a href="#">Remove</a>
sample check	<input checked="" type="checkbox"/> 1 <input type="checkbox"/> 2 <input checked="" type="checkbox"/> 3	<a href="#">Remove</a>
sample radio	<input checked="" type="radio"/> yes <input type="radio"/> no	<a href="#">Remove</a>
sample file	<input type="button" value="Choose File"/> arctic_fox-wallpaper-1920x1080.jpg	<a href="#">Remove</a>

**Submit**

Figure 5.20: Figure showing job after filling in the data in all the fields added

**Welcome nz !**

[← Select Template](#)

Pending jobs:

test\_role

**sample job**

**Add Data**

<b>Job_name</b>	sample job
<b>Test Key</b>	sample text
<b>Test Drop</b>	def
<b>Sample Check</b>	<input checked="" type="checkbox"/> 1 <input type="checkbox"/> 2 <input checked="" type="checkbox"/> 3
<b>Sample Radio</b>	<input checked="" type="radio"/> yes <input type="radio"/> no
<b>Sample File</b>	arctic_fox-wallpaper-1920x1080.jpg

**Submit**

Figure 5.21: Figure showing *nz* dashboard and the job received from *ny*

# CHAPTER 6

## Results - Data Generator based Continual Learning Systems for Edge Devices

### 6.1 About this Chapter

Neural networks suffer from Catastrophic forgetting problem when deployed in a continual learning scenario. Pseudo rehearsal is a technique where a generator is used to synthetically generate training data of the previous task to retrain the neural network to prevent forgetting. Edge devices usually have severe computational and memory constraints which limits the deployment of pseudo rehearsal schemes directly on them. In this chapter, a continual learning system that deploys the generator on a server and regularly updates the neural networks deployed on the edge whenever required is demonstrated.

### 6.2 Remote Data Generation (RDG) architecture

This scalable and flexible system, to implement synthetic data generation is proposed as a service, that is called as Remote Data Generation (RDG) architecture and it consists of a Data service, Prediction service, Training service, and Controller service. Two types of nodes are possible in this workflow: User-Interface (UI) nodes such as the controller, where the parameters of the model are configured, and computational nodes such as Prediction service, Training service, Data service. The role of each service is as follows.

#### 6.2.1 Prediction-Service

The prediction services' primary function is to generate predictions for given input data using the deployed neural network. In case of a federated learning setting, the prediction service could be running on the edge device. And in case of a centralized setting,



the prediction service also could be running on the cloud using a copy of the deployed neural network to improve latency times. The prediction service plays an integral role when generating synthetic data using Genetic Algorithms. The synthetic data is generated by constant interactions between the data service and the prediction service.

### **6.2.2 Training-Service**

Training-service is designed to retrain the model using the synthetic-data generated by Data service. It updates the model, and that updated version can be deployed to the edge devices. One significant advantage of implementing a separate microservice for training the neural networks is that, while other services can be deployed on cheaper hardware with low computational capabilities, the training service can be deployed on specialized hardware that are optimized for training processes.

### **6.2.3 Data-Service**

Data Service's primary function is to generate synthetic data or provide original data for retraining neural networks deployed on edge devices. It takes labels or class as input and produces respective data as output. The data service could consist of a data generator like GAN, GMM or Genetic Algorithms or could be a database consisting of original data directly.

## **6.3 Genetic Algorithm (GA) as a data generator**

This is an micro-service style implementation of the system proposed by [Suri and Yeturu \(2020\)](#), where synthetic data is generated by a series of communications between Genetic Algorithms and the deployed neural network.

To generate the synthetic data for a target class, the Genetic Algorithm begins with a random set of images which are then given to the deployed network for prediction. the softmax confidence of the network on these images for the target class is considered as their fitness scores. the fittest 24% individuals are sent to the next generation, where a series of mutation and cross-over operation are followed to populate the next genera-

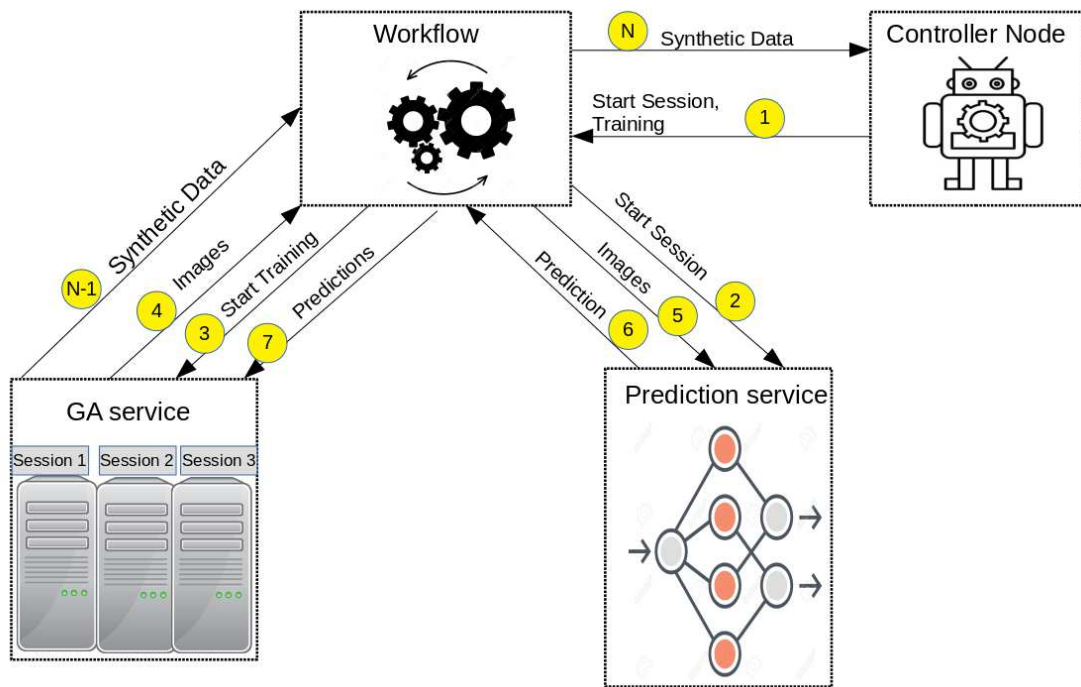


Figure 6.1: Figure shows the interaction between controller and the database via workflow system when generating synthetic data using Genetic Algorithms.

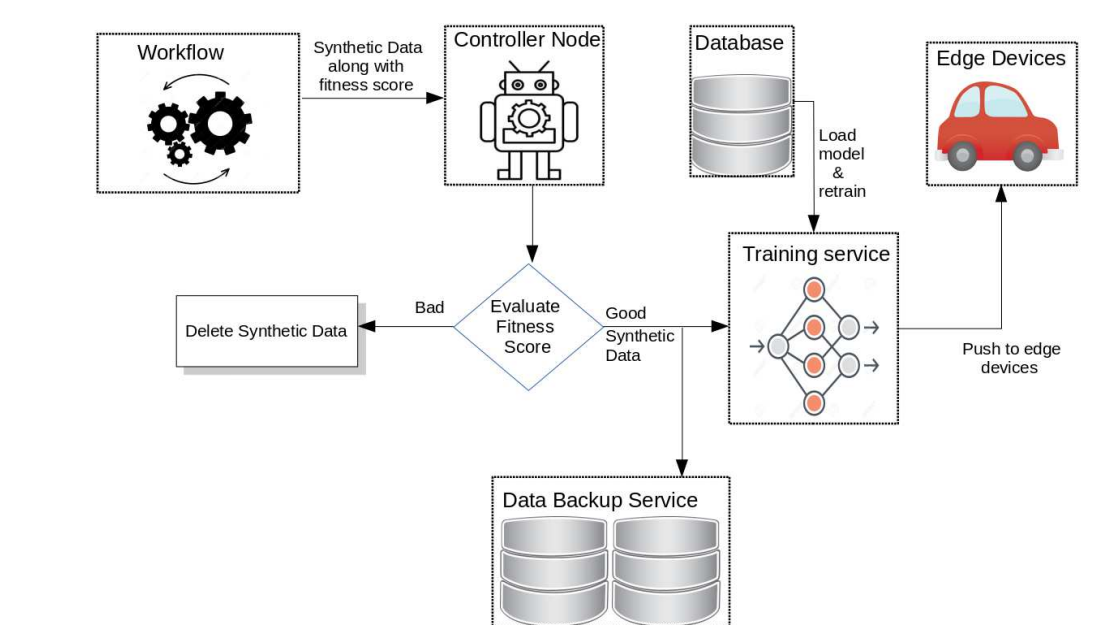


Figure 6.2: Figure shows the retraining process of the system after synthetic data has been generated.

tion. This is repeated until the organisms of a given generation reach a certain fitness threshold.

To implement this technique, a Genetic Algorithm is implemented as the generator service. the GA service generates synthetic images and sends them to the prediction service via the workflow. the prediction service predicts the score using the deployed model and returns it to the workflow system. GA service uses the predicted scores to generate a new batch of images. the controller specifies the *number of generations* up to which the service should generate images. Each generation will generate images where the fitness of population is greater than previous generation's.

Figure 6.1 shows the steps involved in generating synthetic data from a time and event perspective. A step-wise list of interactions between the services is provided below:

- STEP 1: At time step  $T_1$ , the *controller* node requests to begin the training via workflow and the GA service generates the first batch of images.
- STEP 2: At  $T_2$ , images or synthetic samples that are generated by the GA service are sent to Prediction Service via the workflow.
- STEP 3: At  $T_3$ , the Prediction service receives the images and then returns the prediction scores for all the images.
- STEP 2 and 3 are repeated until a certain fitness threshold is reached or until the number of iterations is finished.
- STEP N-2: At  $T_{N-2}$ , the score for each synthetic sample is sent to the GA service.
- STEP N-1: At  $T_{N-1}$ , the GA Service realising that the fitness threshold has been reached will send the final synthetic data to the controller via the workflow.
- STEP N: At time step  $T_N$ , the workflow routes the synthetic data to the controller and message is sent to GA service to stop the training.

All the request which happen between nodes/services and workflow happen as post requests and data flows as JSON objects. An independent session is opened between the GA service and prediction service to increase the throughput of the system by allowing simultaneous executions for multiple systems. Figure 6.2 shows the steps involved after this synthetic data is received by the controller from the workflow. the controller has two jobs to do: (1)To start the session; (2) To evaluate the quality of the synthetic data generated. After receiving the synthetic data, the controller is presented with an option to whether or not retrain the model on this new synthetic data. On being satisfied with the quality of the generated data, the controller can push the synthetic data along with the command to "retrain" the model to the workflow system. the workflow system then routes the synthetic data to the *Training service*.

A copy of the synthetic data sent is also sent to the *data backup service* that stores the synthetic data for future use. If the generated images' fitness score is not high enough, then the controller instructs the workflow (by setting the message "delete\_data") to delete the synthetic data. Training service will retrain the model based on the synthetic data and upgrade the model. Now this upgraded model can be sent to edge devices for deployment.

## 6.4 General Adversarial Networks (GAN) as a data generator

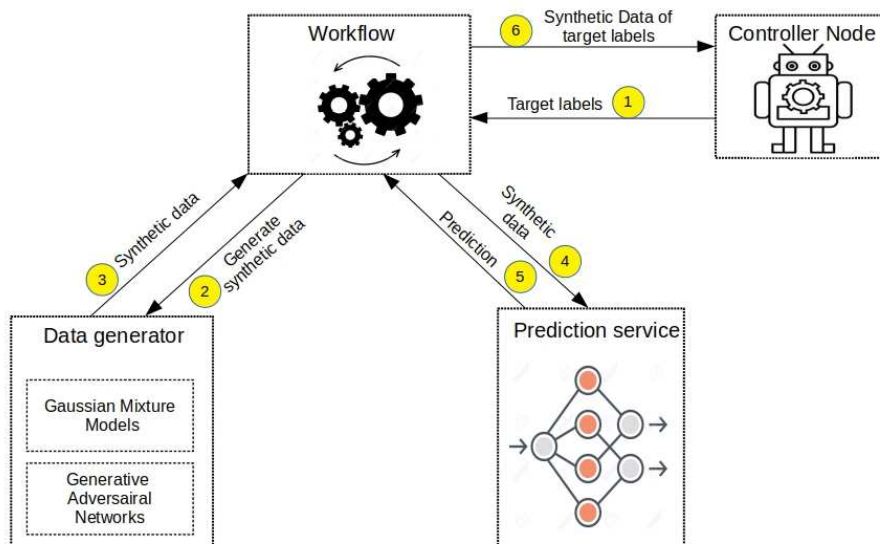


Figure 6.3: the block diagram shows the interactions between various services when either GMM or GAN are used as Data services.

[Shin et al. \(2017\)](#) suggested use of Generative Adversarial Networks (GANs) as *generators* to generate synthetic data. In this technique, instead of storing the original data, a Generative Adversarial Network is trained until it can synthetically recreate the original data. This fully trained GAN is stored in a database and the original data is then discarded. This technique greatly saves space as storing a GAN consumes much lesser space compared to storing entire original data. In the beginning of the process, the controller sends the request to workflow with the required *target labels* as shown in Figure 6.3.

The *workflow system* then sends the request to *data service* which in this case has a

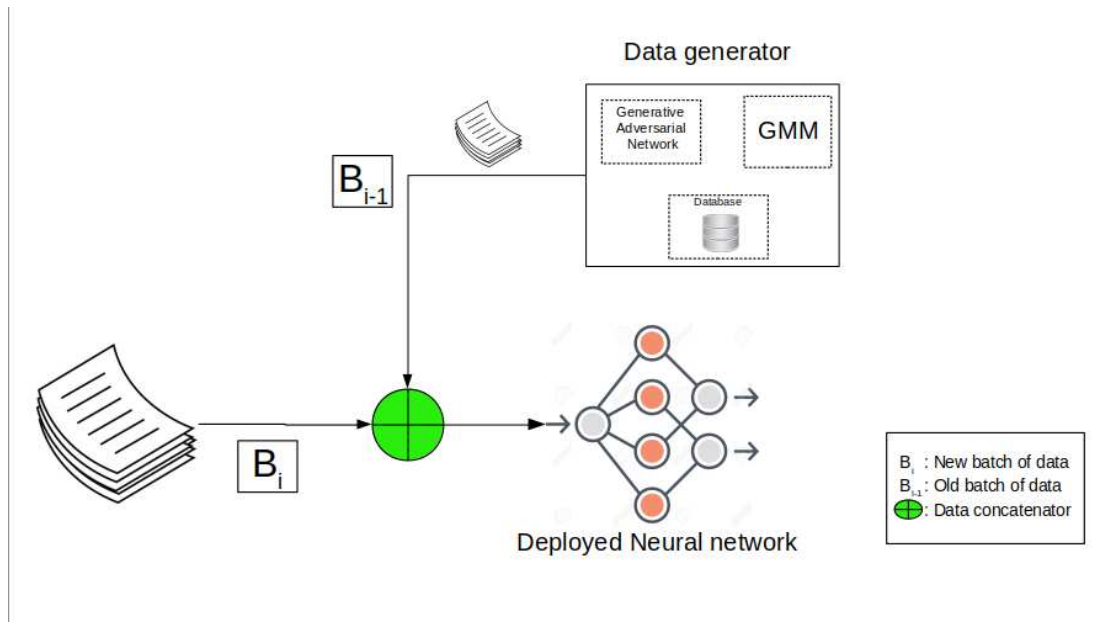


Figure 6.4: the figure shows the concept of pseudo rehearsal. the synthetic data of previous batch, generated using GAN or GMM are interleaved with the newly arrived training data.

GAN in it. the request is send as a JSON object which has message field with various flags and variables that describe the desired properties of the synthetic data.

Data service then generates the synthetic images upon request and sends back to the workflow system. Since the data generated by GAN is unlabelled, the workflow system sends the synthetic data to the prediction service where it predicts the labels of the generated samples. these predictions are sent back to the workflow. the workflow system finally filters out the samples belonging to the target classes based on the newly generated labels and sends these samples to the controller for retraining.

## 6.5 Guassian Mixture Models (GMM) as Data Generator

In a Gaussian Mixture model, the dataset is assumed as a collection of  $n$  Gaussians. A Gaussian Mixture Model can also be used as a data generator for pseudo rehearsal. Just like using GAN as a generator, using of GMM as a generator follows a similar process. the controller first initiates a request to generate samples of the target classes with the workflow. The workflow sends a command to the GMM which is in the data service, to start generating synthetic data. the data service responds back to the workflow system

with the synthetic data. As the synthetic data is unlabelled, the workflow systems sends the data to the prediction service which labels the data. The workflow system uses these labels to filter out the samples belonging to the target classes and sends them to the controller.

## 6.6 Original Data as a service

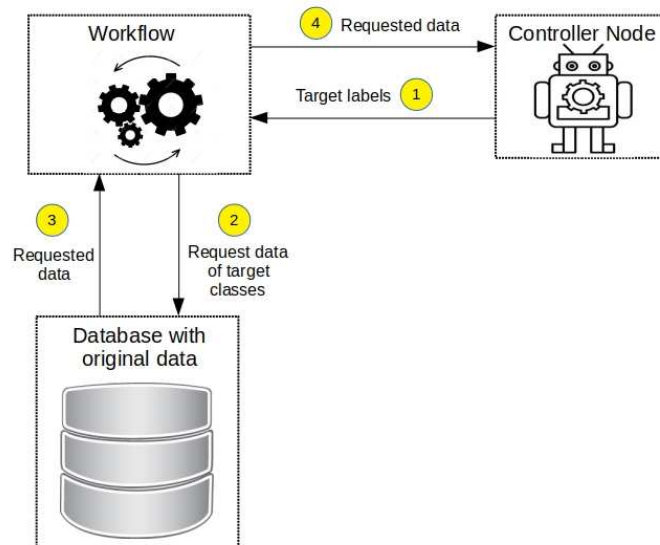


Figure 6.5: Figure shows the interactions between the Controller and the database containing the original data.

[Robins \(1995\)](#) proposed the concept of *Rehearsal*, where the neural network is trained on original data of the previous task to prevent catastrophic forgetting. However, storing of entire previous data requires allocation of considerable memory resources which is not feasible for edge devices. therefore, by deploying the original data on the cloud *as-a-service* and then requesting the subset of it according to the need of the deployed model might be an optimal solution. The proposed architecture can be used in both federated learning [Bonawitz et al. \(2019\)](#) setting where each deployed model is personalized according to the user, or a centralised setting where a central controller periodically pushes one uniform model to all the edge devices. Figure 6.5 depicts a centralised setting where a human controller initiates the request for data. The controllers request is routed to the workflow, which in turn raises a request with the original data service. the data service returns the requested data to the workflow system which routes it to the controller. the controller is then presented with a choice whether to re-train

the model on it or not. If the controller chooses to retrain the model, the synthetic data is passed to the *training service* which loads the model and retrains it. This retrained network can be considered as the upgraded model which is then pushed to edge devices for deployment. In contrast, in a federated learning setting, the retrain service is present on the edge device itself, where the deployed model is retrained and then deployed.

## 6.7 Implementation Details

The proposed system was implemented completely in Python language, however, it has to be noted that the user is free to implement any of web-services using a different as the proposed system is platform agnostic. *Flask* library [Grinberg \(2018\)](#) was used to build and deploy all the web-services while *Requests* library was used to implement the HTTP POST and GET requests. *Numpy* [Harris et al. \(2020\)](#) was used to represent and generate synthetic data throughout the system. Representing the data using Numpy arrays allows us to use the system in non-image applications as well. MongoDB was used to implement all the databases in the system. The reason for selecting MongoDB over SQL is that JSON objects were used to transmit data between different services in our workflow. As SQL has static *table-oriented* design where the columns of the table have to be predefined, inserting JSON objects with varying sizes is not possible. As MongoDB is the NO-SQL database with little restrictions over the structure of the database, JSON objects with varying sizes can be inserted into MongoDB with ease. This flexibility offered by MongoDB influenced our decision to select it over an SQL based database. *Pymongo* library was used to connect the web services with the Mongo Database.

All the neural networks were developed in Keras [Chollet \(2015\)](#) with Tensorflow [Abadi et al. \(2016\)](#) running in the back-end. The Gaussian Mixture Model was implemented in Sklearn [Pedregosa et al. \(2011\)](#). the system was tested on MNIST Digits [LeCun and Cortes \(2010\)](#) and MNIST Fashion [Xiao et al. \(2017\)](#) data sets which were available as a standard data-set in Keras.

The experimentation was carried out on a network of three systems with Intel Core i7 Quad core. The three systems had 7.7GB, 7.7GB and 16GB of available RAM space. The systems were connected using a Wi-Fi (802.11n) router with a link speed of 150

Mbps between each system and the router. All the micro-services were launched on different systems.



# CHAPTER 7

## Conclusions and Future Directions

Various state-of-the-art systems have been studied and the common features have been identified. A formalism for representation and study of workflow systems at the fundamental level is presented here. The formalism offers highly scalable and light-weight architecture to create workflows for any problem scenario involving the concept of messages and steps between processes. In the proposed system, a workflow itself can be dynamically configured and deployed in the runtime without interrupting the execution of the system. A novel concept of **flow mutation** in the formalism where the routing logic can modify the message before delivering it to a node is introduced. This offers theoretical power to decouple a node from the routing logic making it very light weight. **Rendering as a service** for user interface type nodes which are already made thin by flow mutation concept is introduced. The formalism may be realized in any contemporary technology of choice as demanded by a domain. A proof of concept using Python and Flask libraries has been presented. The system is scalable to execute on diverse levels of hardware from high end computers to even low end Iot devices. In our PoC implementation any node can be published and discovered by including its identifier in the workflow messages. The nodes and workflows may be distributed across systems and can be run on local machines in a secure way. Communication between nodes and workflows happens through micro service invocations making the system extremely flexible and adaptable. The usefulness of the system in smart campus and data generation use cases has been evaluated. This shows the usefulness of our formalism to build a truly domain agnostic workflow system.

A formal representation of a workflow system is introduced to include the following concepts -

- Formalism of *Flow Mutation* - a novel concept, where router modifies or mutates the message before delivering to the node
- *Flow mutation* results in plug-n-play of workflow routing logic
- Formalism of condition based routing where target node is determined based on message fields and their values
- Formalism of node actions and **rendering as a service** for user interaction type nodes

- Event queue, node queue for continuous flow of data in the system
- The formalism offers *highly scalable and light-weight architecture to create workflows for any problem scenario involving concept of messages and steps between processes*
- Hence there is a system where *workflow itself can be dynamically configured and deployed in the runtime without interrupting the execution of the system*
- The formalism is very generic and can be realized in any contemporary technology of choice.
- The system has been deployed at: <http://services.iittp.ac.in/workflow>.

Nodes' messages are stored on its database. As evident from the workflow architecture, node messages are workflow, workflow processes those messages and sends the data to the target node. The data floats between workflow and node in units of time to the workflow engine. The rate of providing this data to the workflow can be calculated as time taken to process its data.

Appropriate time testing of the system was done and the results were that the system processed **2000** messages in **18** seconds. Of course, the system depends on network bandwidth, processor and memory, but the workflow system (database) is highly scalable, so throughput can be met with our own requirements.

**Future directions:** Integration of the workflow system with contemporary blockchain technology and name based routing protocols can go further into device factors at scale. On the user convenience front, a graphical interface for routing logic needs to be provided. The proof of concept system requires better performance analysis on metrics for communication between nodes and workflows, storage and retrieval and user interface response. The workflow system needs to provide a dashboard as a smart application and to be evaluated for battery and processing requirements. A platform for creation, publication and market store for workflow applications on the general purpose system needs to be provided to enable widespread use and to spin the wheel of micro-economy over this open technology. Work is actively underway to provide the features for suspend, resume, fork, join need to be provisioned in the proof of concept implementation.

Provision for compensatory workflows: Some actions in a workflow need to be undone and exception handling is a routine requirement in any administrative or business scenario. For this purpose, compensatory workflows need to be created. In our proposed architecture the compensatory workflows can be included as any other regular workflows by just defining the routing logic and handling exception conditions.

## **CHAPTER 8**

### **Contributions**

The original idea and architecture was proposed by Dr. Kalidas in 2018 and majority of the core engine for flow mutation and a placeholder code rendering as a service was written by him. Subsequently 2018 batch M.Tech students experimented with addition of use cases. Our contributions in discussion with Dr. Kalidas are additional functionality to the core engine for provisioning a working subsystem for rendering as a service, concept of self service in node, and demonstration of use cases for academics. MS Scholar Suri Bhasker Sri Harsha also contributed in the synthetic data generation use case with the workflow.

# APPENDIX A

## Installation

Installation of our software is pretty easy.

- Install the latest version of Python here: <https://www.python.org/downloads/>
- Download MongoDB from: <https://www.mongodb.com/try/download/community>. Install MongoDB Community Server and MongoDB Compass from official Website for particular platform.
- Download our code from: <https://github.com/Sushmitha999/Workflow>.

After installing Python and MongoDB, follow the these steps:

Setting up a virtual environment (optional)

For **Windows**:

- Install virtual environment:

```
py -m pip install --user virtualenv
```

- Create a virtual environment:

```
py -m venv env
```

- Activation:

```
.\env\Scripts\activate
```

For **Linux/MacOS**:

- Install virtual environment:

```
python3 -m pip install --user virtualenv
```

- Create a virtual environment:

```
python3 -m venv env
```

- Activation:

```
source env/bin/activate
```

To use the software, use the following steps:

1. Run the following to install the required Python packages:

```
python -m pip install -r requirements.txt
```

2. Run the following once to insert docs into the database:

```
pip install add_collection.py
```

3. To start the flask server:

```
python views.py
```

4. Start the workflow engine:

```
python process_wf.py
```

5. For academics use cases, run the background service:

```
python acads_bg.py
```

# APPENDIX B

## Call Flow Graph

The flow of interaction between the python functions in routing, HTML files and JavaScript functions is described here using 5 figures. The blue boxes represent the python functions. The yellow boxes represent JavaScript functions. The red boxes represent the HTML files and if any JavaScript function is used in a HTML file, the yellow box with the JavaScript function resides in the corresponding HTML red box.

The figure B.1 represents the hierarchy of various HTML files where the layout.html is the main base file. The rest of the files are child files. The layout.html file contains load\_functions function which loads when the dashboard of a user is loaded. The sub functions of load\_functions are also shown.

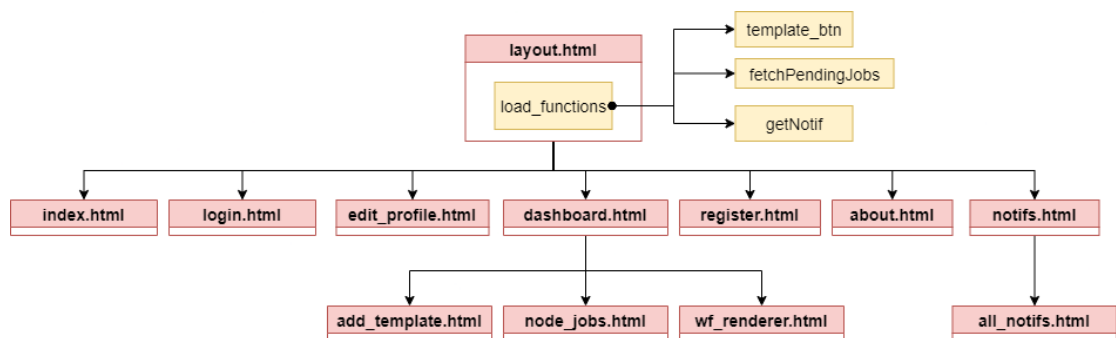


Figure B.1: The figure depicts the HTML file inheritance with layout.html as base HTML file.

The figure B.2 shows the routing between the basic HTML files and the python functions. The figure B.3 shows the routing for fetching and deleting notifications.

The figure B.4 describes the routing for adding templates which includes fetching templates, adding a selected template as a new job. It also shows the rendering of jobs and submitting a job.

The figure B.5 shows the flow of functions between the main workflow engine and the routing logic (conditions.py). The workflow engine (process\_wf.py) uses the functions for making the necessary changes to the academics messages. The functions are present in acads\_wf.py.

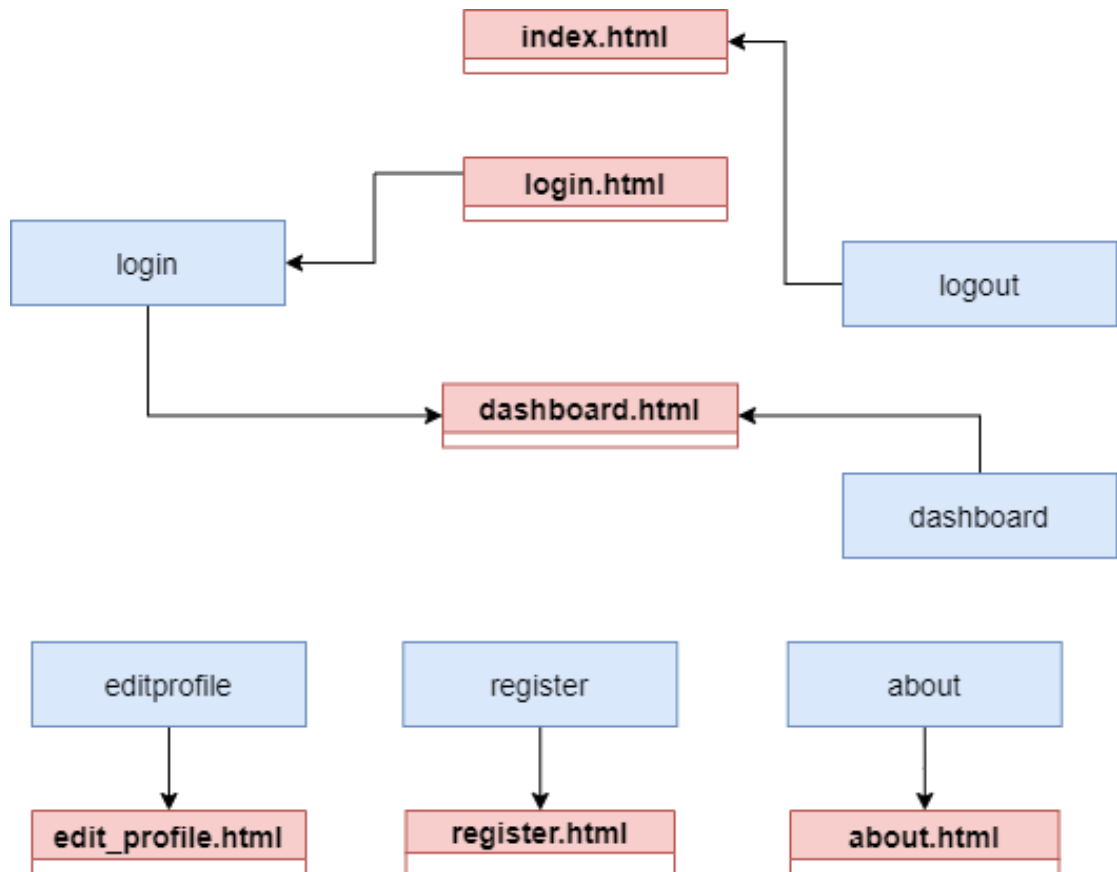


Figure B.2: The figure depicts routing for basic HTML files

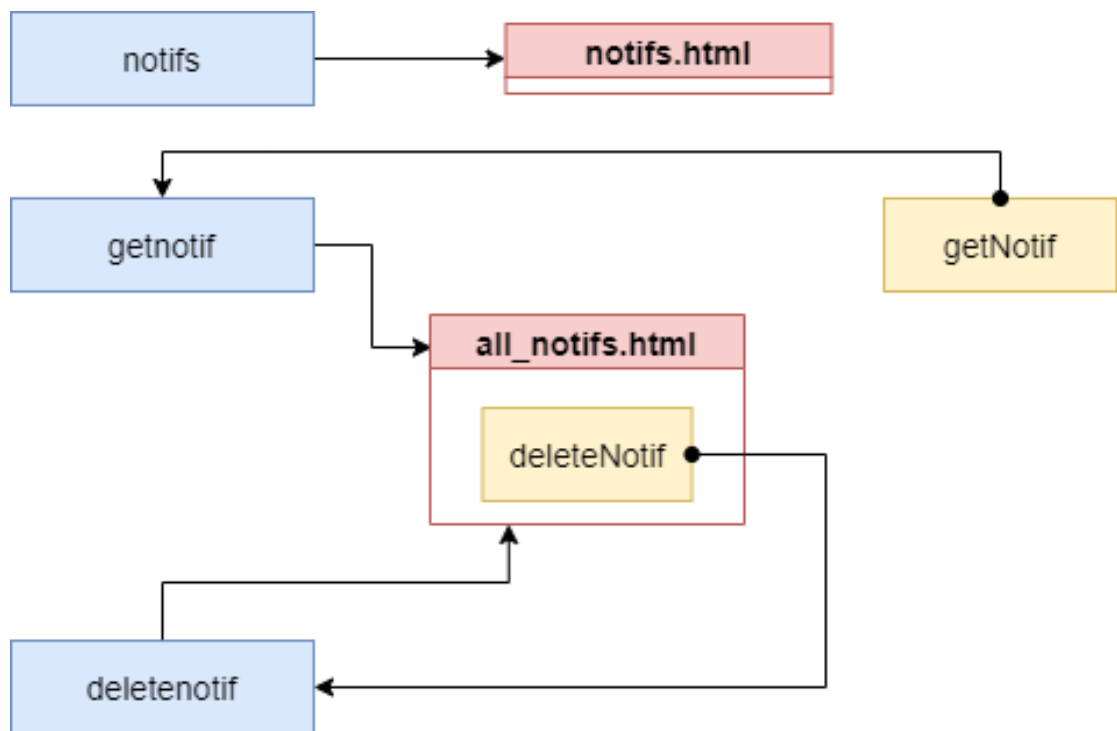


Figure B.3: The figure depicts routing for adding templates and job rendering.

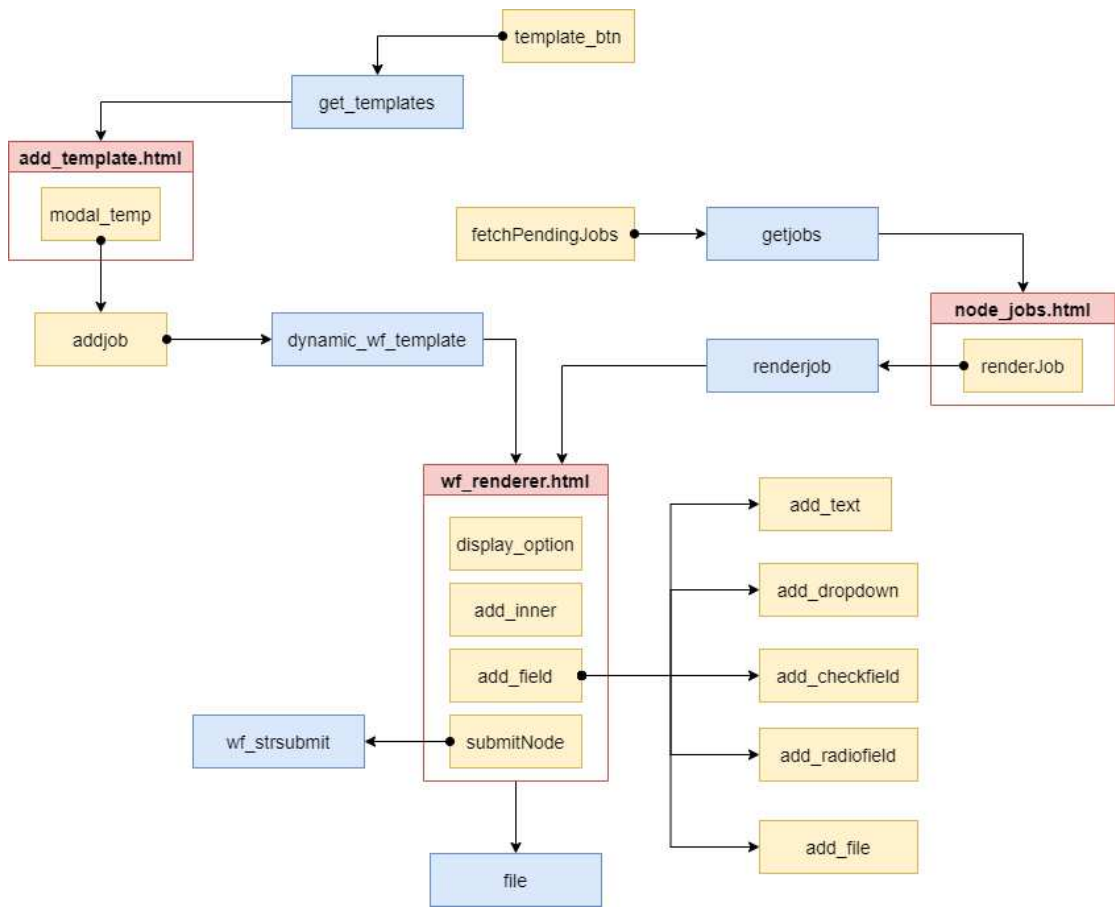


Figure B.4: The figure depicts routing for notifications.

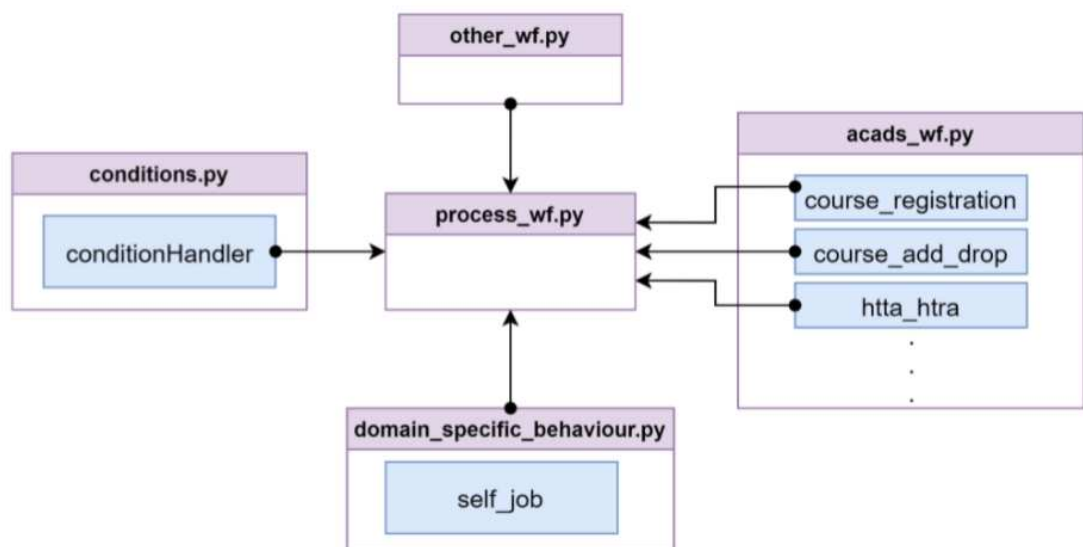


Figure B.5: The figure depicts flow of workflow engine for academic use cases as an example.



## REFERENCES

1. **M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng**, Tensorflow: A system for large-scale machine learning. *In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016.
2. **G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. El Abbadi, and M. Kamath**, Exotica/fmqm: A persistent message-based architecture for distributed workflow management. *In Information Systems Development for Decentralized Organizations*. Springer, 1995, 1–18.
3. **I. Altintas, S. Purawat, D. Crawl, A. Singh, and K. Marcus** (2019). Toward a methodology and framework for workflow-driven team science. *Computing in Science & Engineering*, **21**(4), 37–48.
4. **K. Amin, S. Kapetanakis, K.-D. Althoff, A. Dengel, and M. Petridis**, Dynamic process workflow routing using deep learning. *In International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer, 2018.
5. **Apache** (2014). Apache airflow. <https://airflow.apache.org>.
6. **A. Barker and J. Van Hemert**, Scientific workflow: a survey and research directions. *In International Conference on Parallel Processing and Applied Mathematics*. Springer, 2007.
7. **F. Betancourt, K. Wong, E. Asemota, Q. Marshall, D. Nichols, and S. Tomov**, opendiel: a parallel workflow engine and data analytics framework. *In Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*. 2019, 1–7.
8. **K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, V. T. Overveldt, D. Petrou, D. Ramage, and J. Roselander** (2019). Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*.
9. **J. Brzeziński, A. Danilecki, J. Flotyński, A. Kobusińska, and A. Stroiński**, Workflow engine supporting restful web services. *In Asian Conference on Intelligent Information and Database Systems*. Springer, 2011.
10. **J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd**, Gridflow: Workflow management for grid computing. *In CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings..* IEEE, 2003.
11. **S. Ceri, P. Grefen, and G. Sanchez**, Wide-a distributed architecture for workflow management. *In Proceedings Seventh International Workshop on Research Issues in Data Engineering. High Performance Database Management for Large-Scale Applications*. IEEE, 1997.

12. **W. Chen, R. F. da Silva, E. Deelman, and T. Fahringer** (2015). Dynamic and fault-tolerant clustering for scientific workflows. *IEEE Transactions on Cloud Computing*, **4**(1), 49–62.
13. **F. Chollet** (2015). Keras. URL <https://github.com/fchollet/keras>.
14. **R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman** (2017). A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems*, **75**, 228–238.
15. **R. F. da Silva, D. Garijo, S. Peckham, Y. Gil, E. Deelman, and V. Ratnakar**, Towards model integration via abductive workflow composition and multi-method scalable model execution. In *9th International Congress on Environmental Modelling and Software*. 2018.
16. **E. Deelman, R. F. da Silva, K. Vahi, M. Rynge, R. Mayani, R. Tanaka, W. Whitcup, and M. Livny** (2020). The pegasus workflow management system: Translational computer science in practice. *Journal of Computational Science*, 101200.
17. **E. Deelman, A. Mandal, M. Jiang, and R. Sakellariou** (2019a). The role of machine learning in scientific workflows. *The International Journal of High Performance Computing Applications*, **33**(6), 1128–1139.
18. **E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny** (2019b). The evolution of the pegasus workflow management software. *Computing in Science & Engineering*, **21**(4), 22–36.
19. **R. Filgueira, R. F. Da Silva, A. Krause, E. Deelman, and M. Atkinson**, Asterism: Pegasus and dispel4py hybrid workflows for data-intensive science. In *2016 Seventh International Workshop on Data-Intensive Computing in the Clouds (DataCloud)*. IEEE, 2016.
20. **Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman** (2010). Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems*, **26**(1), 62–72.
21. **P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig** (2000). Crossflow: Cross-organizational workflow management in dynamic virtual enterprises. *Computer Systems Science & Engineering*, **1**(ARTICLE), 277–290.
22. **M. Grinberg**, *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.
23. **C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant** (2020). Array programming with numpy. *Nature*, **585**(7825), 357–362.
24. **T. Heinis, C. Pautasso, and G. Alonso**, Design and evaluation of an autonomic workflow engine. In *Second International Conference on Autonomic Computing (ICAC'05)*. IEEE, 2005.

25. **C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good**, On the use of cloud computing for scientific workflows. *In 2008 IEEE fourth international conference on eScience*. IEEE, 2008.
26. **M. Islam, A. K. Huang, M. Battisha, M. Chiang, S. Srinivasan, C. Peters, A. Neumann, and A. Abdelnur**, Oozie: towards a scalable workflow management system for hadoop. *In Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. 2012.
27. **B. K. Joseph and O. Mosweu** (). Integrating document workflow management system in the business processes of a public institution.
28. **G. Kappel, S. Rausch-Schott, and W. Retschitzegger** (2000). A framework for workflow management systems based on objects, rules and roles. *ACM Computing Surveys (CSUR)*, **32**(1es), 27–es.
29. **D. Król, R. F. da Silva, E. Deelman, and V. E. Lynch**, Workflow performance profiles: development and analysis. *In European Conference on Parallel Processing*. Springer, 2016.
30. **Y. LeCun and C. Cortes** (2010). MNIST handwritten digit database. URL <http://yann.lecun.com/exdb/mnist/>.
31. **X. Li, J. Song, and B. Huang** (2016). A scientific workflow management system architecture and its scheduling based on cloud service platform for manufacturing big data analytics. *The International Journal of Advanced Manufacturing Technology*, **84**(1-4), 119–131.
32. **B. Linke, R. Giegerich, and A. Goesmann** (2011). Conveyor: a workflow engine for bioinformatic analyses. *Bioinformatics*, **27**(7), 903–911.
33. **Y. Liu, S. M. Khan, J. Wang, M. Rynge, Y. Zhang, S. Zeng, S. Chen, J. V. M. Dos Santos, B. Valliyodan, P. P. Calyam, N. Merchant, H. T. Nguyen, D. Xu, and T. Joshi**, Pgen: large-scale genomic variations analysis workflow and browser in soykb. *In BMC bioinformatics*, volume 17. BioMed Central, 2016.
34. **B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao** (2006). Scientific workflow management and the kepler system. *Concurrency and computation: Practice and experience*, **18**(10), 1039–1065.
35. **A. Mandal, P. Ruth, I. Baldin, R. F. Da Silva, and E. Deelman**, Toward prioritization of data flows for scientific workflows using virtual software defined exchanges. *In 2017 IEEE 13th International Conference on e-Science (e-Science)*. IEEE, 2017.
36. **A. Mandal, P. Ruth, I. Baldin, Y. Xin, C. Castillo, G. Juve, M. Rynge, E. Deelman, and J. Chase**, Adapting scientific workflows on networked clouds using proactive introspection. *In 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015.
37. **P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. Goble**, Data lineage model for taverna workflows with lightweight annotation requirements. *In International Provenance and Annotation Workshop*. Springer, 2008.

38. **A. Mujezinović** and **V. Ljubović**, Serverless architecture for workflow scheduling with unconstrained execution environment. *In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2019.
39. **H. Nawaz**, **G. Juve**, **R. F. Da Silva**, and **E. Deelman**, Performance analysis of an i/o-intensive workflow executing on google cloud and amazon web services. *In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016.
40. **P. Neophytou**, **P. K. Chrysanthis**, and **A. Labrinidis**, Confluence: Continuous workflow execution engine. *In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011.
41. **M. Pap**, **L. Z. Nagy**, and **D. Fekete** (2020). Improving e-learning material quality with the aid of deep learning and workflow management.
42. **F. Pedregosa**, **G. Varoquaux**, **A. Gramfort**, **V. Michel**, **B. Thirion**, **O. Grisel**, **M. Blondel**, **P. Prettenhofer**, **R. Weiss**, **V. Dubourg**, **J. Vanderplas**, **A. Passos**, **D. Cournapeau**, **M. Brucher**, **M. Perrot**, and **E. Duchesnay** (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, **12**(Oct), 2825–2830.
43. **A. Pradhan** and **R. K. Joshi**, Architecture of a light-weight non-threaded event oriented workflow engine. *In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*. 2014.
44. **X. Qiu**, **C. Lan**, **B. You**, and **J. Li**, Information fusion in the applications of workflow management system. *In 2019 IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. IEEE, 2019.
45. **S. Rinderle**, **M. Reichert**, and **P. Dadam**, Adept workflow management system: Flexible support for enterprise-wide business processes (tool presentation). *In International Conference on Business Process Management*, volume 2678. 2003.
46. **A. Robins** (1995). Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, **7**(2), 123–146.
47. **M. Rynge**, **S. Callaghan**, **E. Deelman**, **G. Juve**, **G. Mehta**, **K. Vahi**, and **P. J. Maechling**, Enabling large-scale scientific workflows on petascale resources using mpi master/worker. *In Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*. 2012.
48. **D. D. Sánchez-Gallegos**, **D. Di Luccio**, **J. L. Gonzalez-Compean**, and **R. Montella**, Internet of things orchestration using dagon\* workflow engine. *In 2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019.
49. **H. Shin**, **J. K. Lee**, **J. Kim**, and **J. Kim** (2017). Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*.
50. **C. Simpkin**, **I. Taylor**, **G. Bent**, **G. de Mel**, and **R. Ganti** (2018). A scalable vector symbolic architecture approach for decentralized workflows.

51. **B. S. H. Suri** and **K. Yeturu** (2020). Pseudo rehearsal using non photo-realistic images. *arXiv preprint arXiv:2004.13414*.
52. **R. Tomsett, G. Bent, C. Simpkin, I. Taylor, D. Harbourne, A. Preece, and R. Ganti**, Demonstration of dynamic distributed orchestration of node-red iot workflows using a vector symbolic architecture. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019.
53. **B. Tovar, R. F. da Silva, G. Juve, E. Deelman, W. Allcock, D. Thain, and M. Livny** (2017). A job sizing strategy for high-throughput scientific workflows. *IEEE Transactions on Parallel and Distributed Systems*, **29**(2), 240–253.
54. **K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalgo, M. P. Balcazar Vargas, S. Sufi, and C. Goble** (2013). The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic acids research*, **41**(W1), W557–W561.
55. **H. Xiao, K. Rasul, and R. Vollgraf** (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.
56. **P.-C. Yang, S. Purawat, P. U. Jeong, M.-T. Jeng, K. R. DeMarco, I. Vorobyov, A. D. McCulloch, I. Altintas, R. E. Amaro, and C. E. Clancy** (2019). A demonstration of modularity, reuse, reproducibility, portability and scalability for modeling and simulation of cardiac electrophysiology using kepler workflows. *PLoS computational biology*, **15**(3), e1006856.
57. **Y. Yang, L. Zhang, and Q. Zhang**, Constructing business simulation training platform based on workflow management systems. In *2018 14th International Conference on Computational Intelligence and Security (CIS)*. IEEE, 2018.