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

I. Abstract

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, we have identified 14 key features 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. We do a 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. We solve this issue 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, we use a fast upcoming concept of *rendering as a service* for interface generation. 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 we report here 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. We hope the developer research community considers the proposed idea to incorporate in their workflow software systems.

***Index Terms*—workflow, rendering as a service, flow mutation, micro services.**

II. 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.

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 we can define and address in a formal and theoretical setting and demonstrate by a proof of concept of the ideas developed.

We have proposed key features (Table I) for evaluation of workflow systems on the basis that the majority of state of the art systems (Table II) do 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 [1], inter-node communication mechanisms, workflow abstraction as a graph and user interfaces. *Please note: both the tables are at the end of the manuscript in single page column setting.*

Our contributions: We propose here 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. The formalism is generic and any contemporary technology can be chosen to implement. We present a proof of concept implementation of the workflow architecture in Python environment and Flask libraries. 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.

Organization of the paper: The methods section provides details on the formalism based reasoning and light weight implementation of proof of concept. The results section provides details on theoretical implications as well as evaluation details of the proof of concept on the 5 scenarios. The conclusions and future directions enlist the scope and limitations of the proposed method and present out next steps.

III. Methods

We have reported here a general purpose workflow system. 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 our proof of concept and initial working model, we have implemented the formalism in Python using Flask micro-services framework.

A. Formal representation and reasoning

Here we present a formal representation of the system and study 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_ϕ, the functionality is $(w_\phi, m) = B_N(n_\phi, m)$
18) For w_ϕ, the functionality is $(n_\phi, m) = B_W(w_\phi, m)[0]$
19) (Note here that n_ϕ, w_ϕ are only for theoretical completeness, there are not actual function calls in any implementation)

The workflow and node deamons 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

Workflow Daemon:
$E_W = E_W + < w_\phi, m_\phi >$
//Infinite iteration
while $|E_W| > 0$ **do**
  if $\exists e \in E_W : e[0] \neq w_\phi$ **then**
    $E_W = E_W - e$
    $w = e[0]$
    $m = e[1]$
    $NL = B_W(w, m)$ //get list of nodes to which this message is mapped
    while $(\forall (n', m') \in NL)$ **do**
      $E_N = E_N + (n', m')$ //goes to node event store
    end while
  end if
end while

Algorithm 2 Workflow Process

Node Daemon:
$E_N = E_N + < n_\phi, m_\phi >$
//Infinite iteration
while $|E_N| > 0$ **do**
  if $\exists \eta \in E_N : \eta[0] \neq n_\phi$ **then**
    $E_N = E_N - \eta$
    $n = \eta[0]$
    $m = \eta[1]$
    $AL = B_N(n, m)$ //obtain a list of actions
    while $(\forall (n', m', \alpha) \in AL)$ **do**
      //applying action $\alpha(\cdot, \cdot)$
      // α can be user interaction or computational type
      if α is user interaction **then**
        $m'' = R(m')$ //using rendering as a service
      else
        //when α is computational type
        $m'' = \chi(n', m')$ //where $\chi(\cdot, \cdot)$ is a computational node
      end if
      $(w', m'') = \alpha(n', m')$
      $E_W = E_W + (w', m'')$ //goes to workflow event store
    end while
  end if
end while

B. Schematic of the system

The formalism (Section III-A) may be realized in diverse platforms and application technologies. A software design perspective of the formalism is presented in the schematic

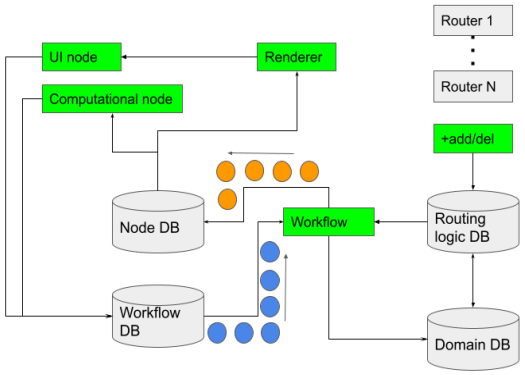

Fig. 1. Schematic diagram of workflow system is depicted in this figure. Green rectangles indicate key functional nodes for UI, Rendering, Computations, Workflow and Routing logic. There are 4 database each for node, workflow, routing logic and domain specific key-value information. Routing logic can be dynamically provided. The blue and orange small circles correspond to messages before and after modification respectively by the workflow module.

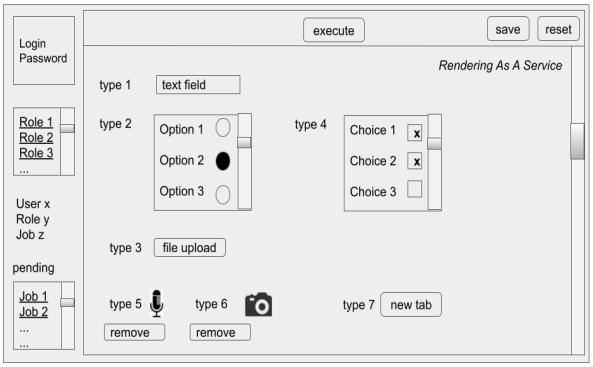

Fig. 2. Schematic depiction of a user interface type node. The node processes any message characterized by user, role and job fields. The interface has interaction elements of diverse types including - file upload of diverse types, radio button, check boxes, text boxes and submit button. The execute button performs message modification and submission to the workflow. There are convenience features to do-undo and reset.

(Figure 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.

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 we called *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 2).

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. We have then made, *rendering as a service* itself. The sequence of interactions with backend system are shown in (Figure 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

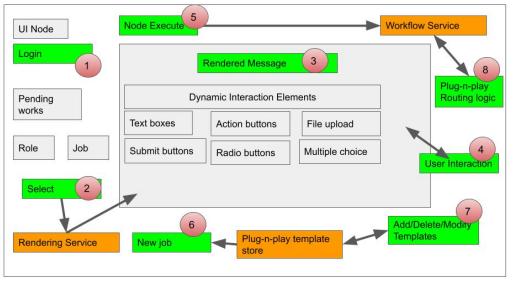

Fig. 3. This is a depiction of sequence of steps in a typical user interaction node. The green rectangles indicate action modules and the numbered circles denote the sequence of steps. The orange rectancles indicate background modules and their relationship to the foreground interface. The detailed steps are covered in the manuscript text.

C. Distributed workflow system

The system itself is highly distributed (Figure 4). On a single computer, there can be several instances of workflow system.

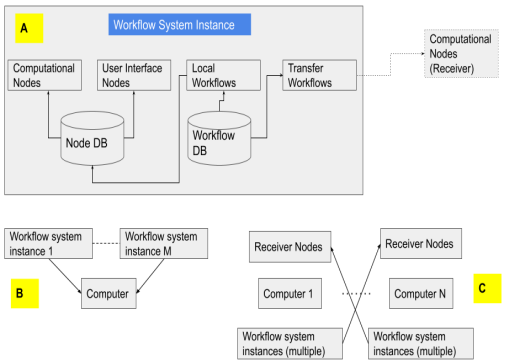

Fig. 4. A highly distributed workflow system is depicted here. (A) Denotes the workflow system instance which as nodes of two types computational and interaction type, local workflows and a transfer workflow. It has databases for nodes and workflows. (B) Denotes one computer having multiple workflow 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 reciever nodes.

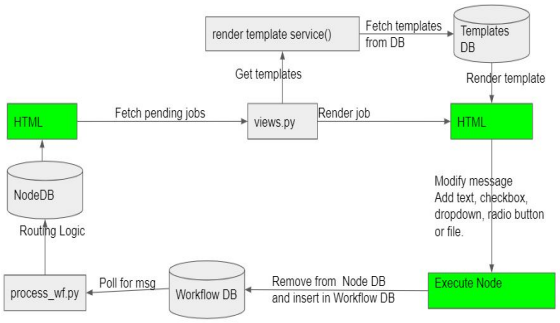

Fig. 5. This figure depicts modules in the proof of concept system built using python and flask libraries. The user interaction components are shown in green colour. The HTML box corresponds to user visualization and interaction with graphical elements. Databases for node, workflow and templates are shown as cylinders. Arrows indicate flow of data or next steps. The annotations on top of arrows denote specific relationships.

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.

D. Provision for compensatory workflows

Some actions in a workflow need to be un-done 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 are at par and served as any other regular workflows by just defining the routing logic and handling exception conditions in the routing codes.

E. Proof of concept using Python and Flask

We have implemented the formalism in Python environment using Flask libraries for micro services. A schematic view of the framework is shown in (Figure 5). 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 6). 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.

IV. Results

We have introduced a formal representation of a workflow system to include the following concepts -

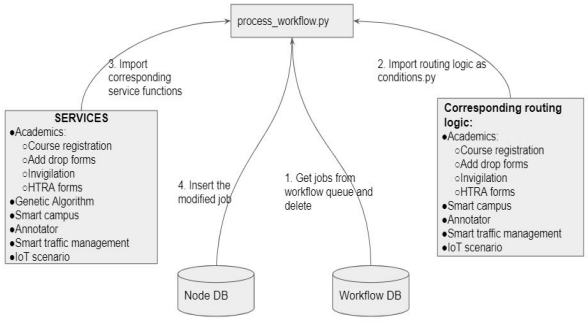

Fig. 6. This figure depicts the process_wf.py module built using python. Databases for node and workflow are shown as cylinders

- 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*
- We therefore have 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.

A. Lightweight implementation

The formalism in (Section III-A) can be realized in any state of the art web services environments. However for demonstration purposes and use cases, we have implemented in python using flask micro-services framework and tested it for smart campus and machine learning workflow scenarios. 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 (Figure 2) where rendering can also be obtained from other systems. Therefore we have an algorithm which is *general purpose and domain agnostic workflow algorithm suitable for both high end computational infrastructure to even low end IoT devices*.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*

In this set up, the nodes are just identifiers logged into the node table. They are available as soon as messages are written. However authentication is provided to login for nodes that are newly added. We therefore have a system where *the nodes can be published and consumed in other workflows*. Micro services framework allows for *decentralized execution of nodes and workflows in a multi-computer scenario*. The communication between micro services is through *micro service URL mechanism or any other custom mechanism as well for interaction among services*.

B. Novel architecture - Messaging mutation by coordinator and Rendering as a service

We present an architecture of the workflow system where nodes and workflows communicate messages (Figure 1). The concept of workflow system is not new and it has been there since more than a couple of decades. However there are several dozens of systems in the state of the art and still even more are being developed.

We have studied why there is proliferation of systems if one major type of architecture is suitable. It is understandable that there are several business scenarios that demand different level of performance, reliability, scale and storage requirements. However, there can still be one major workflow system that can cater to several scenarios with a common kernel code but applications can be many.

We discovered that the answer lies deep in the way the orchestration of workflows is carried out.

There have been several workflow systems till date that employ node and workflow daemons. The orchestration has also been a standard concept where *from* node to a *to* node mapping is maintained. The notion of orchestration in the state of the art is close to that of a *postman concept*. A postman does not open a letter and examine its contents, it focuses on deliver of the contents. It is the duty of the household to process the contents of the letter and in this case, the node has to interpret the contents.

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, we upgrade the *postman* level orchestrator to a *coordinator*. The coordinator not only looks at the message, it also modifies it as required (Figure 7). 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.

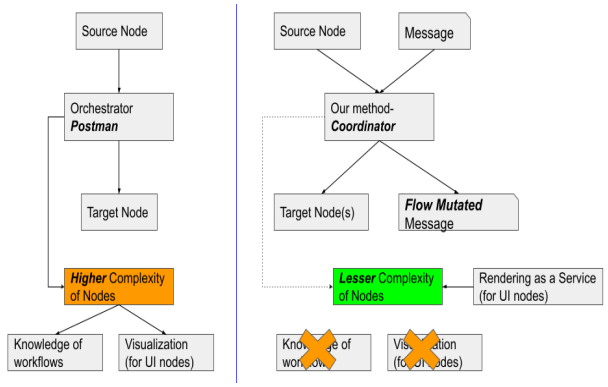

Fig. 7. A depiction of the **flow mutation** concept. The present workflow systems consider a router as a *postman* type there by leading to higher node complexity. The figure depicts a complex node coupling with routing logic and rendering. The figure depicts the effect of decoupling a node with routing 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.

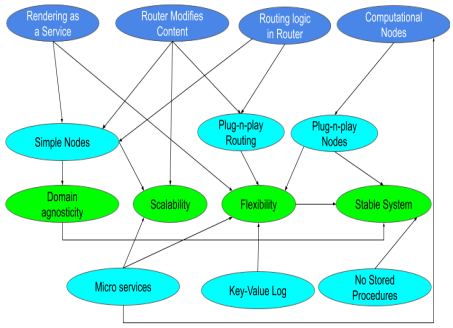

Fig. 8. A depiction of concepts and their interrelated influence in the workflow system. Each oval indicates a process or concept. An edge is drawn from one to another if there is a positive effect from source to target. The green ovals indicate major contribution of the proposed system. We can observe that all the concepts lead to a stable, flexible and scalable architecture.

In the formal representation, the orchestration logic can modify the content (Section III-A Point 15) 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 (Figure 8).

C. Rendering as a service

The service for rendering offers rich types of interface elements such as (i) file upload, (ii) text boxes, (iii) radio buttons, (iv) check boxes and (v) submit buttons (Figure 2). These interaction elements can be dynamically added as the message flows through the workflow system. We therefore have a mechanism where *each message comes with its own rendering template as an attribute for consumption by graphical nodes.*

The first study involves an example of a smart campus scenario where students enrol for courses and later change their decision is depicted in (Figure 9). The scenario is as follows—

- STEP 1: First, the academic section logs in using their username and password. The corresponding dashboard is displayed.
- STEP 2: The academics can add a new job for add/drop course from the preexisting templates. In this job, the academics can add an empty add/drop course PDF form.
- STEP 3: On execute, this job is sent to all the students.
- STEP 4: Now, students can login and see add/drop course job and the empty add/drop form that the academics has sent to them. 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.
- STEP 5: On execution, the job will be sent to that corresponding instructor.
- STEP 6: The instructor can now login and see the pending add/drop jobs from the students. The instructor can verify the PDF form signed by the student and upload his/her form after signing it. On execute, this job will be again sent to the student.
- STEP 7: The student can now only verify the add/drop PDF form. After verification, the student can now edit the course ID again and send this form to another instructor as needed.
- STEP 8: 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.
- STEP 9: On executing, the job will be sent to the student's faculty advisor.
- STEP 10: Finally, the faculty advisor checks the PDF form signed by the student and the course instructor(s), signs it and executes the job. This job is sent to academics. Under the role, add/drop course forms, academics can see all the students' forms.

D. Computational nodes

The system is experimented for application in scenarios where nodes are computational nodes. One of the scenario is a

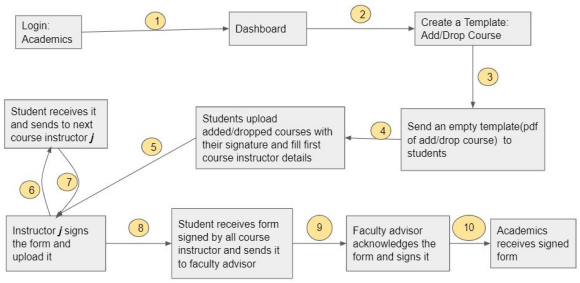

Fig. 9. This figure depicts steps in the scenario for course add and drop. There are 10 steps, which are described in the main text.

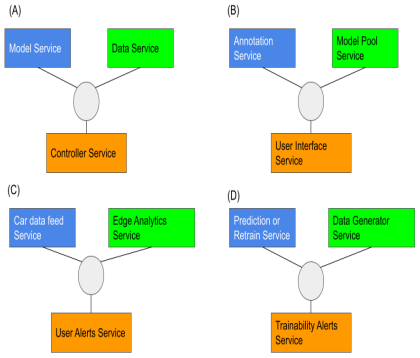

Fig. 10. Computational node as a service. The rectangles denote service components and the central circle denotes the workflow service. The parts of this figure (A) - general framework involving model, data and control; (B) application to digital document annotation as a service; (C) application to car feed analytics as a service and (D) application to data generation for building stable models as a service.

machine learning workflows where model, data and controlling logic are three components (Figure 10). We have experimented with three types of computational nodes in machine learning scenario.

1) General framework of machine learning models - Model Service, Data Service, Control Service: In a machine learning modeling scenario, the models perform mainly two types of tasks - prediction and training. The code for a model may be very particular to a scenario in question and is tightly coupled with the libraries, the hardware and the data set used. Moving a model from one system to another for the purpose of using workflow facilities is a challenging scenario. In this regard, model itself is provided as HTTP service. The post methods along with meta-data such as prediction or re-train flags indicate action items for a model.

The data itself is a major component of a machine learning system. The data may be a sampled version, fresh one or the old one, synthetic data or filtered according to certain conditions as dictated by the domain. The best way to deal with data is to provide data itself as a service.

The third important component in a machine learning model is the control logic. The communications between the model and the data. The predictions, processing the output confidence scores, generating new data, retraining a model on the data

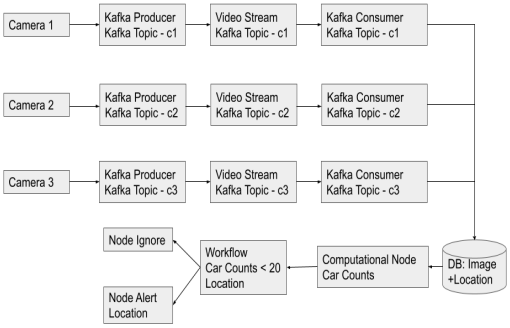

Fig. 11. This figure depicts a 'toy' use case of traffic monitoring with cameras and computer vision. The idea is to use Apache Kafka for reliable communication and workflow nodes and servers to run computer vision algorithm and decision logic.

Fig. 12. An image of the document with text is shown. The interface provides for drawing rectangular regions and provide text labels. There is an option to build models and auto annotate.

and other aspects of coordination themselves involve multiple steps. In this regard, controller is provided as a service.

This architecture has similarity to Model-View-Controller (MVC) architecture, however, the model in the MVC corresponds to data-model, view corresponds to user interface and the controller corresponds to business logic. The workflow formalism and the microservices framework allow for simplifying the scenario for a data science scenario by 3 fundamental services for model, data and control logic.

The second system involves evaluation of computational nodes *to demonstrate the system for diverse use cases including smart campus, computer vision and IoT scenarios*. The formalism is domain agnostic and we have successfully experimented *for the ability to customize the communication platform using an example of Apache Kafka* (Figure 11). We have evaluated the system for integrating with a reliable message passing mechanism such as Apache Kafka. The scenario was a toy set up involving three cameras each having its own identifier and they send image data to a centralized storage. However the communication happens using Apache Kafka based producer, consumer and topic mechanisms. The images are processed by any popular computer vision algorithm for object detection such as Yolo [2] for detection of any type of vehicles in the image, in this toy set up, we have considered 'cars'. The computer vision algorithm is run inside a node as a background process that polls for availability of messages in the node queue. The algorithm determines number cars and sends to the workflow. The workflow then decides based on a user threshold and sends the resultant message to either node ignore or node alert. The node alert would then send an SMS and do follow up action to alert a user such as police staff.

2) Examples of computational codes: The third system involves providing digital document annotation as a service to unify the efforts across several optical character recognition systems. In our system, there are multiple models built for various annotations. In (Figure 10 B) the three nodes - (i) annotation as a service, (ii) model building and pooling as a service and (iii) the user interface as a service. This work is involving and is communicated elsewhere however, it is only briefly stated for illustration purposes. A description of annotation front end interface is shown in (Figure 12). The front end provides user interaction that captures image sub-regions and their labels. The back-end machine learning model trains multiples models for different sub-region and label pairs. These models are used for prediction of labels for sub-regions in a new document. The communication between front end, back-end machine learning model and controller logic is realized via the workflow system. The system itself is simple due computational nodes and communication of image file paths. However, our intention is to present the usefulness of the same workflow system that was able to cater to the needs of a data science scenario.

The fourth system involves dealing with analytics modules running on edge device and rendering modules running on remote node as services. The car OBD 2 scanner feeder service collects photographs of a location at regular intervals. The analytics node running using scikit-learn libraries on Raspberry Pi board determines the driving profile and generate predictive text. The output of the algorithm is sent to the a remote node in a laptop computer using transfer workflow. The profile is presented to the end user using interaction type node on the laptop computer (Figure 10 C) and (Figure 13).

The fifth system involves machine learning based generation of synthetic data for augmentation to increase robustness of any machine learning model. As a model goes through retraining, the performance on the previous data diminishes. It is prohibitively costly to maintain all of the data. In this regard, [3] genetic algorithm that generates data provided as a service. The model that performs prediction as well as goes through retraining is provided as a service. The communication between the model and the genetic algorithm are abstracted as controller service (Figure 10 D). We therefore have a system *where compute nodes are headless and execute in the background*. (Figure 14))

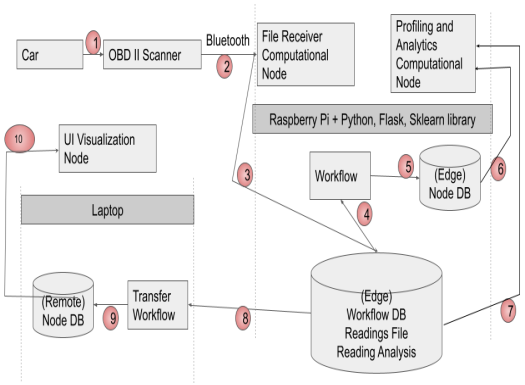

Fig. 13. Communication between edge Raspberry device and the remote laptop device for workflow sharing is demonstrated here. The OBD 2 scanner connected to the car generates data about various sensors and is received by node on the edge device. The analytics node runs on the edge itself. The report of analysis is transferred to a node on the remote device which renders the information to the end user. The sequence of steps are shown numbered over the edges to indicate information flow among the connected entities.

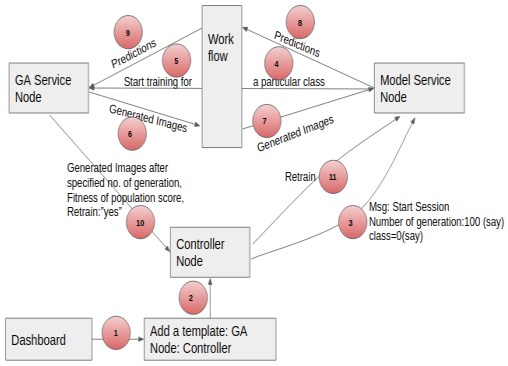

Fig. 14. A depiction of the Genetic Rehearsal Algorithm as a service. Rectangle indicates nodes. Arrow indicates flow from one node to other node. Circles in the figure indicates the steps of flow from one node to other node.

V. Conclusions and Future Directions

We present here a formalism for representation and study of workflow systems at the fundamental level. The formalism offers highly scalable and light-weight architecture to create workflows for any problem scenario involving 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. We introduce a novel concept of **flow mutation** in the formalism where the routing logic can modify the message before delivering it to a node. This offers theoretical power to decouple a node from the routing logic making it very light weight. We then introduce **rendering as a service** for user interface type nodes which are already made thin by flow mutation concept. The formalism may be realized in any contemporary technology of choice as demanded by a domain. We have presented a proof of concept using Python and Flask libraries. The system is scalable to execute on diverse levels of hardware from high end computers to even low end Iot devices. Low end edge devices can run simple workflows and simple nodes and transfer data to other remote nodes for any higher end calculations. 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. We have evaluated the usefulness of the system on a set of five diverse scenarios of (i) smart campus use case, (ii) traffic camera connectivity via Apache Kafka use case, (iii) car analytics through Edge level Raspberry Pi to Remote laptop use case, (iv) computer vision for digital document annotation use case and (v) machine learning model retraining and data generation use case. The domain knowledge is just limited to data files and routing logic in all the 5 example scenarios. This shows the usefulness of our formalism to build truly domain agnostic workflow system.

Future directions: Integration of the workflow system with contemporary block-chain technology and name based routing protocols to go further into device factors at scale. On the user convenience front, graphical interface for routing logic needs to be provided. The proof of concept system requires 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, broadcast need to be provisioned in the proof of concept implementation.

References

[1] A. Arsanjani, N. Bharade, M. Borgenstrand, P. Schume, J. K. Wood, V. Zheltonogov *et al.*, *Business process management design guide: Using IBM business process manager*. IBM Redbooks, 2015.

[2] A. Ćorović, V. Ilić, S. urić, M. Marijan, and B. Pavković, "The real-time detection of traffic participants using yolo algorithm," in *2018 26th Telecommunications Forum (TELFOR)*, 2018, pp. 1–4.

[3] B. S. H. Suri and K. Yeturu, "Pseudo rehearsal using non photo-realistic images," *arXiv preprint arXiv:2004.13414*, 2020.

[4] 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, pp. 1–18.

[5] 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, pp. 76–79.

[6] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig, "Crossflow: Cross-organizational workflow management in dynamic virtual enterprises," *Computer Systems Science & Engineering*, vol. 1, no. ARTICLE, pp. 277–290, 2000.

[7] G. Kappel, S. Rausch-Schott, and W. Retschitzegger, "A framework for workflow management systems based on objects, rules and roles," *ACM Computing Surveys (CSUR)*, vol. 32, no. 1es, pp. 27–es, 2000.

[8] 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*, vol. 2678, 2003, pp. 371–379.

[9] 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, pp. 198–205.

[10] 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, pp. 27–38.

[11] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and computation: Practice and experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[12] A. Barker and J. Van Hemert, "Scientific workflow: a survey and research directions," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2007, pp. 746–753.

[13] 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, pp. 17–30.

[14] 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, pp. 640–645.

[15] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman, "Wings: Intelligent workflow-based design of computational experiments," *IEEE Intelligent Systems*, vol. 26, no. 1, pp. 62–72, 2010.

[16] 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, pp. 1311–1314.

[17] B. Linke, R. Giegerich, and A. Goesmann, "Conveyor: a workflow engine for bioinformatic analyses," *Bioinformatics*, vol. 27, no. 7, pp. 903–911, 2011.

[18] 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, pp. 377–385.

[19] 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, pp. 1–8.

[20] 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, pp. 1–10.

[21] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher *et al.*, "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic acids research*, vol. 41, no. W1, pp. W557–W561, 2013.

[22] 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, pp. 342–345.

[23] 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, pp. 162–173.

[24] W. Chen, R. F. da Silva, E. Deelman, and T. Fahringer, "Dynamic and fault-tolerant clustering for scientific workflows," *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 49–62, 2015.

[25] 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, pp. 1–8.

[26] 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, pp. 535–544.

[27] 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 *et al.*, "Pgen: large-scale genomic variations analysis workflow and browser in soykb," in *BMC bioinformatics*, vol. 17, no. 13. BioMed Central, 2016, pp. 177–186.

[28] X. Li, J. Song, and B. Huang, "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*, vol. 84, no. 1-4, pp. 119–131, 2016.

[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, pp. 108–120.

[30] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017.

[31] B. Tovar, R. F. da Silva, G. Juve, E. Deelman, W. Allcock, D. Thain, and M. Livny, "A job sizing strategy for high-throughput scientific workflows," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 240–253, 2017.

[32] 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, pp. 566–575.

[33] C. Simpkin, I. Taylor, G. Bent, G. de Mel, and R. Ganti, "A scalable vector symbolic architecture approach for decentralized workflows," 2018.

[34] 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.

[35] P.-C. Yang, S. Purawat, P. U. Ieong, M.-T. Jeng, K. R. DeMarco, I. Vorobyov, A. D. McCulloch, I. Altintas, R. E. Amaro, and C. E. Clancy, "A demonstration of modularity, reuse, reproducibility, portability and scalability for modeling and simulation of cardiac electrophysiology using kepler workflows," *PLoS computational biology*, vol. 15, no. 3, p. e1006856, 2019.

[36] 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, pp. 464–467.

[37] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The evolution of the pegasus workflow management software," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 22–36, 2019.

[38] 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, pp. 95–100.

[39] E. Deelman, A. Mandal, M. Jiang, and R. Sakellariou, "The role of machine learning in scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1128–1139, 2019.

[40] 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, pp. 242–246.

[41] I. Altintas, S. Purawat, D. Crawl, A. Singh, and K. Marcus, "Toward a methodology and framework for workflow-driven team science," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 37–48, 2019.

[42] E. Deelman, R. F. da Silva, K. Vahi, M. Rynge, R. Mayani, R. Tanaka, W. Whitcup, and M. Livny, "The pegasus workflow management system: Translational computer science in practice," *Journal of Computational Science*, p. 101200, 2020.

[43] B. K. Joseph and O. Mosweu, "Integrating document workflow management system in the business processes of a public institution."

[44] "Apache airflow," https://airflow.apache.org.

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 I

WE HAVE IDENTIFIED 14 IMPORTANT FEATURES IN ANY WORKFLOW SYSTEM THAT ARE CRITICAL TO ITS WIDE SPREAD USE, SCALABILITY AND FLEXIBILTY WHICH ARE DEPICTED IN THE TABULATION HERE.

S.No	Reference	Remark	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Feature coverage
1.	[4]	Distributed Workflow	1	0	1	0	0	0	1	0	1	0	1	0	1	1	50%
2.	[5]	Distributed Workflow	0	0	0	0	0	0	1	0	0	1	1	0	1	0	29%
3.	[6]	Cross Organizational	0	0	1	0	0	1	1	0	0	0	1	0	1	1	43%
4.	[7]	Object, Rules and Roles	1	0	1	0	0	0	0	0	0	0	1	0	1	0	29%
5.	[8]	Flexible Support	0	0	1	0	0	1	1	0	0	0	1	0	1	1	43%
6.	[9]	Grid Computing	0	0	0	0	0	1	1	1	0	0	1	0	1	1	43%
7.	[10]	Evaluation of Autonomic	0	1	0	0	0	0	1	0	0	0	1	0	1	1	36%
8.	[11]	Scientific Workflow	0	0	0	0	1	1	1	1	1	0	1	0	1	1	57%
9.	[12]	Scientific Workflow	0	0	0	0	0	0	0	1	1	0	0	0	1	0	21%
10 .	[13]	Lightweight	0	0	0	0	0	0	1	0	1	0	1	1	1	0	36%
11.	[14]	Scientific Workflow	0	0	0	0	0	0	1	0	1	0	0	0	1	1	29%
12.	[15]	Intelligent Workflow	1	0	0	0	0	0	1	0	1	0	0	0	1	0	29%
13.	[16]	Continuous Workflow	0	0	0	0	0	1	1	0	0	1	0	0	1	1	36%
14.	[17]	Bioinformatic Analyses	0	0	0	0	0	0	1	0	0	1	0	0	0	0	14%
15.	[18]	Restful Web Services	1	0	0	0	1	0	1	0	0	1	0	0	1	0	36%
16.	[19]	Scientific Workflow	0	0	0	0	0	0	1	0	1	0	0	0	1	0	21%
17.	[20]	Hadoop	0	0	0	0	0	0	1	0	1	0	0	0	1	0	21%
18.	[21]	Nucleic Acids Research	0	0	0	0	1	0	1	0	1	1	0	0	0	1	36%
19.	[22]	Event-oriented Workflow	0	0	0	0	0	0	1	0	0	0	1	0	1	0	21%
20.	[23]	Scientific Workflow	0	0	1	0	0	0	1	0	1	0	0	0	0	0	21%
21.	[24]	Scientific Workflow	0	0	0	0	0	0	1	0	1	0	0	0	0	0	14%
22.	[25]	Data Intensive	0	0	0	0	0	0	1	0	1	0	0	0	0	0	14%
23.	[26]	I/O Intensive	0	0	0	0	0	0	1	0	1	0	0	1	1	0	29%
24.	[27]	Generation Sequencing	0	0	0	0	0	0	1	0	1	1	0	0	1	1	36%
25.	[28]	Scientific Workflow	1	0	0	0	0	0	1	0	1	1	0	0	0	0	29%
26.	[29]	Workflow Performance Profile	1	0	0	0	0	0	1	0	0	0	0	0	1	0	21%
27.	[30]	Extreme-Scale applications	0	0	0	0	0	0	1	1	1	0	0	0	0	1	29%
28.	[31]	Job-Sizing Strategy	1	0	0	0	0	0	1	0	0	0	1	0	0	0	21%
29.	[32]	Scientific Workflows	1	0	1	0	0	0	0	0	0	0	0	0	0	0	14%
30.	[33]	Decentralized Workflows	0	0	0	0	0	0	1	1	1	0	0	0	1	1	36%
31.	[34]	Workflow composition	0	0	0	0	0	0	1	0	0	0	0	0	1	0	14%
32.	[35]	Medical Workflows	0	0	0	0	1	0	1	0	0	0	0	0	0	0	14%
33.	[36]	IoT Workflows	0	0	1	0	0	0	1	1	1	0	0	1	1	1	50%
34.	[37]	Pegasus Workflow	0	0	0	0	0	0	1	0	0	0	0	0	1	0	14%
35.	[38]	IoT Workflows	0	0	0	0	0	0	1	0	0	0	0	0	1	0	14%
36.	[39]	Machine Learning	0	0	0	0	0	0	1	0	0	0	0	0	1	0	14%
37.	[40]	Workflow Scheduling	0	0	0	0	0	0	0	0	0	0	0	0	1	0	7%
38.	[41]	Scientific workflows	0	0	0	0	0	0	0	0	1	1	0	0	1	1	29%
39.	[42]	Translational Scientific Workflow	0	0	0	0	0	0	1	0	0	0	0	0	0	0	7%
40.	[43]	Business Workflow	0	0	0	0	0	0	1	0	0	0	1	0	0	0	14%
41.	[44]	Airflow	1	0	1	0	0	0	1	1	0	0	0	0	1	0	36%
42.	**Proposed Workflow**	General Purpose Workflow	1	1	1	1	1	1	1	1	1	1	1	1	1	1	100%

TABLE II

A COMPREHENSIVE 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. *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.