A Workflow Management Framework for the Dynamic Generation of Workflows that is Independent of the Application Environment

Andrzej Jasinski
Software Research Institute
Athlone Institute of Technology
Athlone, Ireland
a.jasinski@research.ait.ie

Yuansong Qiao
Software Research Institute
Athlone Institute of Technology
Athlone, Ireland
ysqiao@research.ait.ie

Enda Fallon
Faculty of Engineering & Informatics
Athlone Institute of Technology
Athlone, Ireland
efallon@ait.ie

Ronan Flynn
Faculty of Engineering & Informatics
Athlone Institute of Technology
Athlone, Ireland
rflynn@ait.ie

***Abstract*—Workflow is a well-known and widely used technology in business management. Traditional workflow solutions are designed for humans and generally use a graphical representation of workflow elements that reflect the involvement of human factors. Additionally, in a situation where workflow execution is not possible, human intervention is necessary. This means that current workflow design is limited in flexibility, in terms of tasks supported, and that it cannot be easily scaled or adopted. Furthermore, current workflow design is limited in efficiency and efficacy, especially in modern environments (e.g. 5G and IoT) where problems can be complex and solutions unpredictable.**

This paper proposes a workflow management framework that uses dynamically generated workflows to control a managed environment. Exception detection and handling in workflow generation produce recommendations for mitigating incidents that might occur. The key characteristics of the proposed framework are its ease of implementation, flexibility and scalability. These characteristics allow for the quick definition of new tasks, known and unknown, and to assess the quality of the generated recommendation through feedback from the managed environment. Experiments performed in two different environments, robotics and networking, demonstrate the elasticity and functionality of the proposed method to dynamically generated workflows.

Keywords—dynamic workflow generation, proactive management, incident detection, incident prevention, incident mitigation

I. INTRODUCTION

A workflow is designed for humans to control job processes within an organization. It is characterized as a set of progressing steps that need to be done in repeating sequences [1]. There are three main categories that workflows fall under: sequential workflow, state machine workflow and rules-driven workflow [2]. Workflows are implemented in many areas, such as human resources, finance, marketing and sales, providing important business benefits. These benefits include improvements in efficiency, time savings, better use of human resources and the elimination of unnecessary work [3]. The market offers many commercial enterprise-oriented management platforms that can be deployed in the cloud, on mobile platforms or standalone, such as Workflow Max [4] or Asana [5]. The availability of free and opensource solutions is limited and are primarily designed to support a single task. However, Taverna [6] is designed to support highly specialized complex tasks and is distributed under the Apache 2.0 license.

Implementation of workflows is not a trivial task. Human mistakes and design issues can lead to maintenance problems, waste generation and unwanted expenditure in a complex manufacturing process. When a problem occurs, the traditional workflow approach can handle known (predefined) issues only [7]. In complex workflows, certain unknown (unpredictable) events can happen that may lead to a situation where the workflow cycle cannot be completed. In such cases, the whole process is stopped and human intervention required to identify and fix the problem. Problem detection is important regardless of business type, whether it is human resources, production processes, or data management and analysis [8]. There has been a growth in environment infrastructures in recent times, driven by advances in technology, especially in networking (SDN, 5G) and IoT [9]. Managing such infrastructures is a challenge, in particular the problem of incident detection and avoidance [10]. Therefore, there is a need for proactive solutions, especially technologies that can dynamically handle exceptions, ideally without human intervention, or with minimal human intervention.

Research to date in the area of workflows is primarily anchored in business management. The literature shows that workflow in business management is strictly related to, and dependent on, the particular environment in question and the elements associated with it [11], [12], [13]. For the work presented here, this relationship between the environment and the workflow applied to the environment is examined. This workflow-environment relationship informs the novel approach presented here to generate workflows dynamically, in a way that is not tied to one particular environment but is environment-independent.

This paper presents a workflow management framework (WMF), using dynamic workflow generation to control the environment. The framework uses a self-assurance method for exception handling, referred to as the environment incident, and environment feedback to produce a recommendation for the best way to mitigate environment incidents that occur. A central component of the WMF is the relationship between the environment and the dynamic workflow generator engine (DWGE), in which the detection of an alarm (incident) in a particular environment (input) results in the creation of a dynamic workflow to implement a solution (output).

978-3-903176-32-4

The remainder of this paper is organized as follows. In Section II a summary of related work is presented. The key elements of the proposed framework architecture are described in Section III, which is followed by the workflow elements interaction in Section IV. Section V details the novel method for autonomous workflow generation, which is independent of the application environment. Two experiments, in a robotic environment and a networking environment, in which the dynamic workflow generation architecture is applied are described in Section VI. The paper ends with conclusions and suggestions for future work in Section VII.

II. Related Work

In [10], a graphical representation of workflow elements using computerized software or manual paper drawings is described. In both cases, human factors were involved in the design stages, a common weakness of workflow management and design [14]. A number of companies have developed commercial workflow software solutions, for example [15]. This technology has evolved, improving self-design, self-organizational and self-management aspects by adopting workflow software. An example is the use of workflow automation and business rule engine tools to make decisions. Such commercial solutions are designed to support the automation of key business processes and business rules. These solutions are designed to present human-understandable output and require human interaction in workflow management. Some authors have published research in relation to workflow applications in data analytics or cloud computing [16]. The limitation of these papers is a focus on improving aspects of the workflow impact on the managed environment (data) and not on the mechanism of workflow self-generation/self-update. An approach, limited to scientific workflows only, that supports the incremental submission of partial workflows for execution until completion is presented in [17]. The theoretical techniques for exception handling management were the focus of [18]. One of the critical issues in the design process of dynamic adaptive systems is the assurance that the main mechanisms of workflow generation will be adequately maintained when an environment user changes their behavior [19]. Apache Taverna [6], one of the most popular open-source domain-independent workflow management systems, supports the creation of workflows by using a workbench, where a human must manually design and then execute workflows using a dedicated server. However, this solution is complex, requiring trained personnel to support the workflow processes [20]. Any incident that occurs during workflow execution will corrupt the workflow process, resulting in necessary human intervention [21]. Introducing a human element in the management of a workflow problem can result in delays in recognising and resolving the problem, unfortunately, along with unwanted expenditure.

Informed by the literature, this paper presents a novel approach to control, maintain, and manage the environment, and its elements using dynamically generated workflows. Secondly, the proposed framework can detect an environment incident and, using self-adaptive behavior to make an autonomous decision, either update the existing workflow or replace it with a new one. Finally, the quality of the newly generated workflow is assessed based on feedback generated by the managed environment.

III. Workflow Management Framework Architecture

The focus of this paper is a mechanism to dynamically generate workflows that are not restricted to one specific domain of application and require minimal human intervention. This section presents the workflow management framework, the purpose of which is to manage the interaction between an application environment and the workflow generation. The key elements of the WMF are shown in Fig. 1, which includes the DWGE. In the WMF, there is a client-server relationship between the environment and the DWGE. The DWGE server waits for a message from the environment to initialize a workflow (communication interface initialization state). When the server-client link is activated, the first workflow is generated and forwarded to the environment It should be noted that the environment can exist and operate without support from the server (autonomous

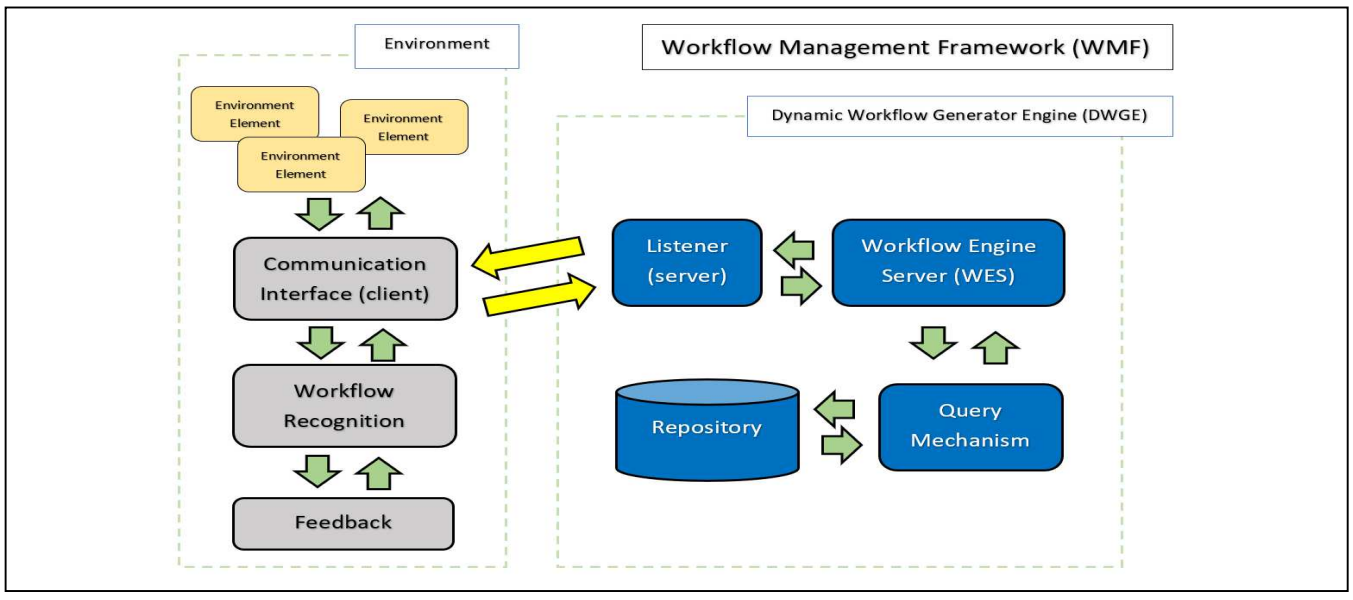

Fig. 1 Workflow management framework

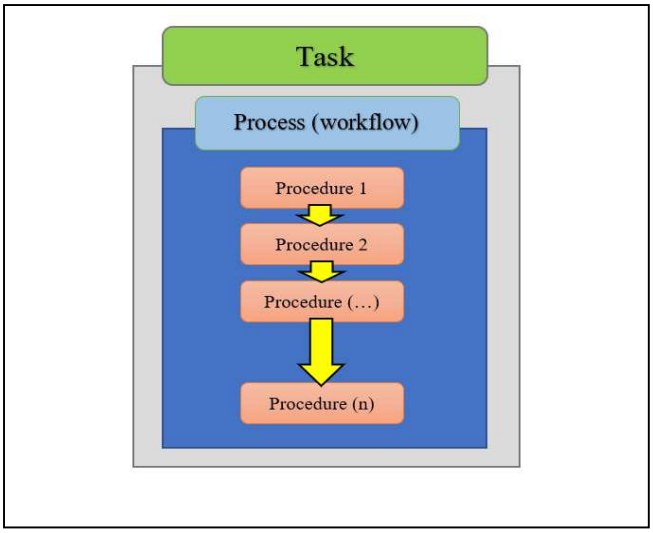

Fig. 2 Environment "Task" execution diagram

environment). It should be noted that the environment can exist and operate without support from the server (autonomous environment). Alternatively, the environment can be supported partly or entirely by the server; this depends on the specifics of the environment and the initial configuration of the principles of cooperation between the WMF's environment and the DWGE. The functions of the WMF elements are:

Server (Dynamic Workflow Generator Engine)

- Listener – maintaining incoming connections, recognizing messages and initializing procedures.
- Query mechanism – responsible for communication with the internal repository and external sources.
- Workflow engine server (WES) – responsible for prerequisite checks, generation of workflows, analyzing environment feedback and preparing recommendations.
- Repository – structured and unstructured database (data types include troubleshooting, manual, handbook and historical).

Client (Environment)

- Communication interface – establishing and maintaining the connection with the server and exchanging messages.
- Workflow recognition – platform to execute workflows, manage exceptions and detect incidents
- Feedback – workflow updater.

IV. Workflow Element Interaction

To complete a job, referred to as a Task, in the controlled environment, a workflow must be executed. The Process (workflow) to complete the Task includes a set of repeating Procedures (components) that need to be executed in sequence as illustrated in Fig. 2. Examples of Tasks include finding a path for a robot to an exit in a robotics environment or establishing a path to destination in a networking environment. The creation of environment elements, such as obstacles in a robotics environment or switches in a networking environment, are prerequisites for the generation of a workflow. Procedures that are part of the Process must be fully understandable by the environment and/or its elements (environment player) to complete a workflow cycle and end a Task.

If an exceptional incident occurs that terminates a Procedure, resulting in complete failure, the current workflow must be updated because the continuation of the execution of this Process is impossible (the Process is marked as corrupted). Such an incident can be detected either by the environment itself or by another dynamically generated workflow that is executed in parallel. The environment sends a message to the workflow server reporting the incident and waits for a server response (recommendation). To avoid an incident, the server sends back either a workflow update or generates a new workflow. Then execution of the failed Process will continue to complete the sequence.

The WES in Fig. 1 is responsible for dynamic workflow generation. All incoming requests from the environment are forwarded by the Listener to the WES. The engine uses the repository in Fig. 1 to read and save both structured and unstructured data. Firstly, the WES generates a prerequisite workflow to check that the environment meets the minimum criteria to execute workflows. In the situation that workflows generated by the WMF refer to those environment components that may not be loaded automatically, the server must create the prerequisite workflow/workflows, activate the necessary environment components and, lastly, check that all required components are loaded and working properly. An environment prerequisite requirement is stored in the repository and is available when the handshake procedure is completed. The WES module is designed to generate a workflow based on environment requests. The request can be direct or indirect. The indirect request is a prerequisite workflow that must be executed first and the decision to generate this workflow is independent of the environment in question. Successful execution of this workflow is mandatory and must be confirmed by environment feedback. An example of a direct request is an update to generate a new workflow necessary to complete a current Task or Tasks. Failure of execution of this type of workflow (incident detection) only affects the single Task (single procedure exception) and will not crash or terminate the entire environment. The WES reads information about workflow components from the Repository as illustrated in Fig. 3. The Master Components database stores data belonging to the specific environment, while the Slave Components database stores supporting (extended) components linked to the Master Components. In the event that one of the Master Components cannot hold an action, or even does subcomponent does not exist, it can be temporarily defined or overwritten using either single or multi sub-components. In general, the Master Components database stores procedures that are universal and can be executed in any environment (e.g., database input-output). The last subcomponent of the Master is a Relationship database component that stores logical procedures (e.g., sets of rules) that can be applied in the workflow process.

V. Dynamic Workflow Generation

The novelty of the proposed approach is the bidirectional nature of the relationship between a controlled environment and the dynamic workflow generation engine. The proposed method supports autonomous and proactive management,

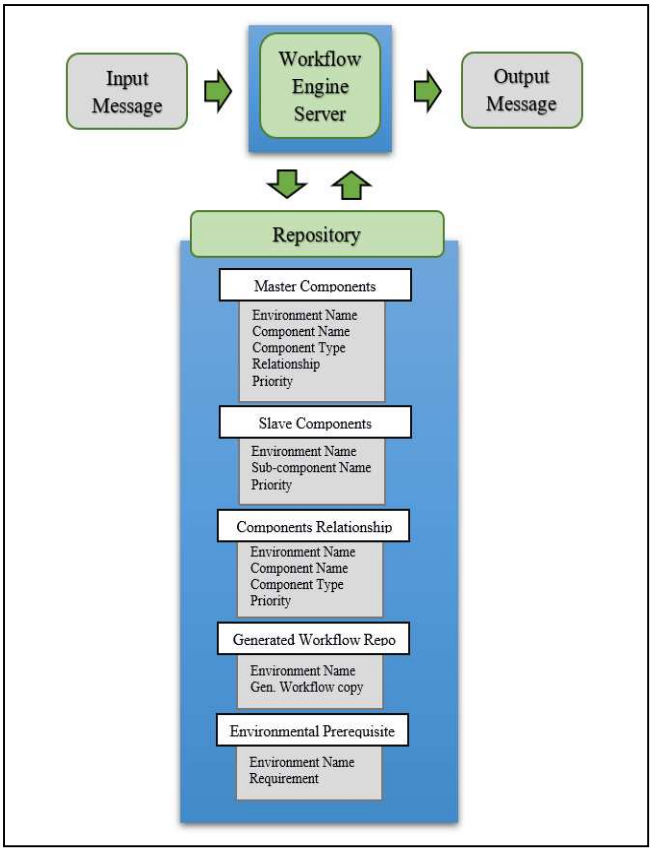

Fig. 3 WES - repository storage

characterized by the fact that it can be triggered by the environment and/or the workflow management framework. In addition to dynamically generated workflows to manage the environment events, it also uses environment feedback to manage and improve methods used for workflow generation (self-improvement). At this point in the development of the WMF, the quality scoring mechanism for a generated workflow is relatively simple but plays an important role in classifying whether or not the dynamically created workflow is still valid, and the same task can be repeated. A

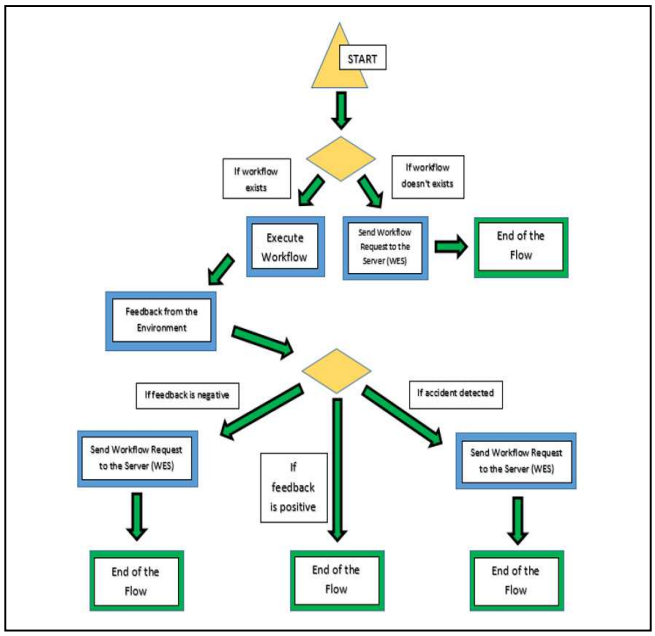

Fig. 5 Decision tree algorithm - Environment

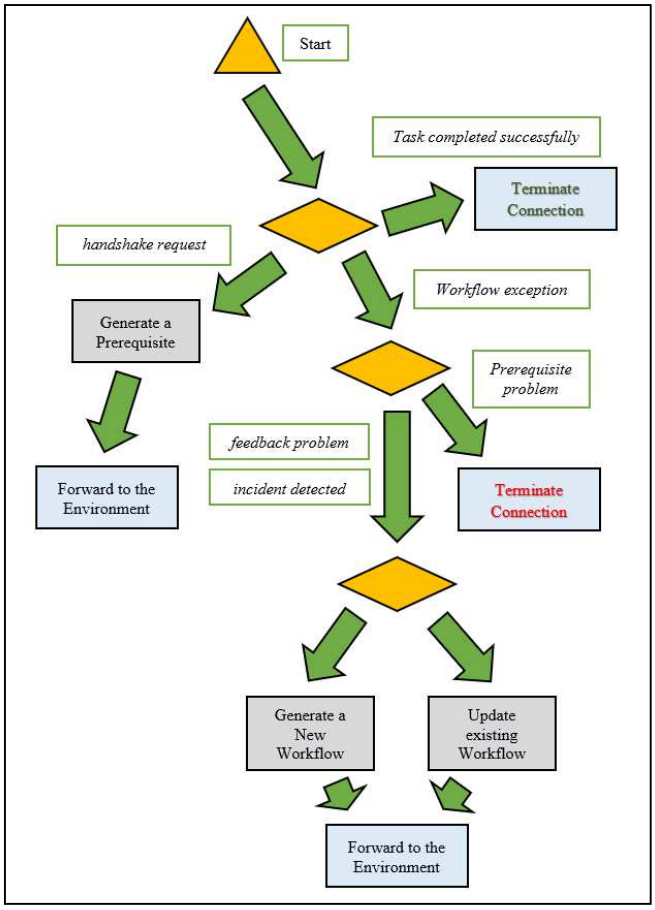

Fig. 4 Decision tree - WES

combination of a sequential workflow (main workflow generation), state machine workflow (logic description) and rules-driven workflow (prerequisites) is used in the proposed solution.

The WMF uses a decision tree algorithm to manage its elements. It should be emphasized that the decision tree must be applied to both the environment (workflow deployment as in Fig. 5) and the WES (workflow generation as in Fig. 4).

The WMF module responsible for workflow deployment on the environment side is the Workflow Recognition. The main role of the decision tree on the environment side is to execute the workflow forwarded from the server, apply calculated feedback (received from the Feedback module) and finally make a decision to either repeat a process (no incident detected) or send back a report to the server about execution exemption (alarm record).

Similarly, the decision tree algorithm applied to the WES is executed when the message is forwarded from the environment. The WES recognizes three main events: handshake request, exception and task finalization.

Most important is the workflow exception message, which is explained in detail here. When workflow exception messages are received by the WES, this suggests two possible issues: a prerequisite problem or an incident that stopped workflow execution. If it is the former, the connection will be terminated with a message indicating that the environment does not meet the initial requirements. If it is the latter, the workflow must be regenerated to avoid a problem that prevents the environment Process (workflow) from completing a sequence. The WES then uses the Repository to

```
1  Environment_ID: Env_911238443,
2  Workflow_Level: 1
3  Workflow_Seq: 1,
4  Workflow_Component: Left********,
5  State: 36@9, Feedback: 36, Step: 16,
6  Request: Update_Failed_Workflow,
7  Result: Failed
```

Fig. 6 Example of workflow information stored in the repository

```
{
    "Env_____": "Env_742204376",
    "Area____": ["29", "15"],
    "Position": ["10", "7"],
    "Exit____": ["16", "7"],
    "Feedback": "7",
    "Obstacle": ["16", "5", "7", "H"],
    "Obstacle": ["12", "11", "11", "H"]
}
```

Fig. 7 Result of calculation environment variables

download historical data associated with the environment. All previously generated workflows are automatically stored in the internal database and marked using an environment ID, as shown in Fig. 6, assigned in the initial handshake procedure. The second record shows the number of updates that have been applied to the original workflow; only successful updates increase this value. In a situation when the following update or updates fail, this number is decreased until the update is successful. The dynamic nature of workflow creation may cause a previously successful partial update on this level to be reverted to because the updates/changes applied later cannot be correctly implemented. Entry number 3 in Fig. 6 is strictly dependent on the workflow component priority value; this value is fixed and related to the component that is stored in the Repository (e.g. movement components are: left or right and the Workflow_Seq value for both is equal). The WES receives the component list from the Repository (value in entry number 4), selecting a component with the lowest sequence value (or picking one randomly if more than one component exists in the Repository with the same sequence value), then checks that the newly selected component wasn't used within the same level (see Fig. 6, entry 2). When all components with the same priority are used, the WES applies the next component with higher priority. Additionally, the WES must save the applied action to the Repository. The fifth entry in Fig. 6 includes a local state record (temporary values), calculated feedback (used for comparison purpose) and a step value (representing the number of completed cycles until failure - seventh entry in Fig. 6). In Fig. 6 there is a request at the sixth entry to update the failed workflow. In exceptional situations, the main component can be temporarily replaced, and a new Procedure performed (e.g., in the robotics environment component "left" can be overwritten by component "return") within a single Process; this new component will receive extra priority, but only temporarily. In the networking scenario this mechanism is not used but can be applied if necessary. From a workflow logic point of view, the robotics "return" must be described as a new Procedure, even though it is one of the possible movements already defined. After finishing, the workflow level will be decreased by one and the failed part of the workflow will be deleted.

VI. Experimental Validation

In this section, the WMF is examined in two application domains or environments, namely, robotics and networks. The motivation for these choices is to evaluate two scenarios: the first is where the WMF takes full control of the environment and its player (robot); the second is where the DWGE cooperates with the environment and supports a functionality extension. In the first scenario, the workflow is used for resolving a collision detection/avoidance problem. In the second, the WMF application tries to find a shorter path (build a dedicated route/tunnel to avoid traffic congestion), detect a connection problem (link down) and then resolve configuration errors. In both cases, the Python programming language was used to build the environment.

A. Robotics environment

In this scenario, the player (robot) must find a route from the starting point (opening) to the destination point (closing) and avoid the obstacles that can appear and block the way.

1) Prerequisites

Firstly, the environment establishes the connection (handshake). The DWGE checks the environment prerequisites and generates the first workflow that is responsible for the creation of the environment. This is a policy-based predefined workflow and is stored in the repository of the DWGE. The first Task that must be completed is a calculation of the area of the environment (number of columns and number of rows), robot coordinates, exit-door coordinates, initial feedback calculation and obstacles (coordinates of the first block, number of blocks, and vertical or horizontal position). Execution results of the first process are stored in the local environment repository (see Fig. 7).

Next, the workflow creates the environment elements (using variables stored in the local repository) necessary to visualize this environment, as shown in Fig. 8. All prerequisite components are represented as a set of Python scripts and must be executed without any errors. When an error occurs, the process must be repeated.

2) Dynamic Workflow Generation Management

When the initial stage is successfully completed, the DWGE can start managing the environment and the environment player. Four items that dynamically generated workflows can implement are described: robot movement, collision detection, collision avoidance and feedback calculator.

a) Robot Movement

The robot understands four basic movements: left, right, forward and backward. From the workflow management point of view, there are two additional logical movements, return and shift. The return or shift can take the form of one of the basic movements and is convergent with those that the robot already knows. The purpose of the return is to show the robot how to get back to the previous position (robot when exploring the environment memorizes each critical stage).

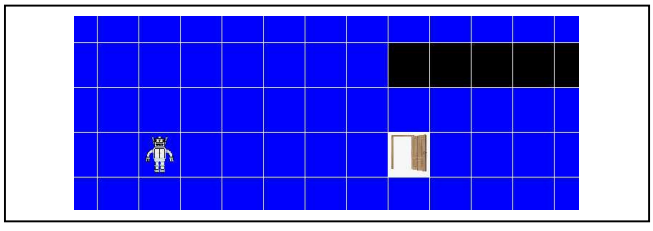

Fig. 8 Environment generated using workflow

TABLE 1 CHARACTERISTICS OF TESTED SCENARIOS FOR THE ROBOTIC ENVIRONMENT

	Average					
Area of the Environment	Obstacles	Number of Incidents / updates	Number of failure updates - reverted	Critical / new workflow	Number of repeated cycles	Successful
Small	3.3	3.6	0	0	21.7	54
Medium	4.3	15.1	0.6	0.3	80.8	137
Large	7.25	35.25	2.5	1	195.5	29

This situation will happen when all four basic movements are classified as a failure. In this situation, part of the workflow will be invalid and will have to be dynamically updated. The shift component is applied in the situation where the robot comes back to the starting point arising from the return procedure execution. The shift is a combination of two basic movements and is responsible for bringing the robot to the new starting point. All six movement workflow components are predefined and stored in the Master Component database located in the engine Repository in Fig.1.

b) Collision detection and avoidance

When the workflow instructs the robot to move, but the execution of the command is impossible, it means that an environment accident had been detected. In the robotics environment, three accidents can be recognized: collision with the obstacle, collision with the border and negative feedback. In all these situations, the workflow is not valid anymore and needs to be updated to avoid a collision.

c) Environment Feedback

The robot is playing an exploration role and every movement is strictly dependent on the dynamically generated workflow. Each time a single movement is completed, the workflow will calculate new feedback. To do this, the environment calculates a surface area located between the destination point (closing) and the current location of the robot. When each movement is applied and is successfully executed, the value of the feedback will decrease. Otherwise, it will be classified as an incident and the incident report will be sent to the server with an update request.

3) Qualitative analysis

During the experiment, a total of 220 environments was generated. The size of the environment and the number of obstacles is the result of a prerequisite workflow execution and it varies each time (see Section VI.A.1).

As seen in Table 1, three main size categories were defined: small, medium and large. The number of obstacles that appear in the environment is represented by a randomly picked variable. The workflow responsible for the generation of this prerequisite is also looking for an exception in the situation where the obstacle does not fit into the environment, in which case the obstacle will not be placed in the environment. Additionally, environment variables will be updated in the environment repository. The number of detected incidents is related to the number of obstacles and increases with the size of the environment. When an update is impossible, resulting in an update failure, this update must be reversed to the previous stage (this is a special situation when all the possibilities of the robot's movement have been used). The next column in Table 1 presents a critical exception where all possible workflows had been executed but the player is back to the starting point resulting in reversing the workflow updates. In this special circumstance, the Shift procedure is applied. When this critical situation is detected four times, the environment will be classified as unsuccessful (the main Task will fail). The next column has the number of cycles, which is the number of successfully executed workflows. The last column in Table 1 shows the successful completions of the main task in the tested environment. There were no exceptional circumstances, and all planned Tasks (jobs) were completed. The number of jobs varies and is related to the area of the environment. The probability of failures and updates is greater in the large environment (more complex) than in the small environment, where the chances of meeting an obstacle are much lower.

B. *Networks Environment*

The second domain in which WMF functionality was tested is a software defined network (SDN) emulated using Mininet software [22] and managed by a POX controller [23]. This is illustrated in Fig. 9, which shows that the Workflow Generator is using a direct connection through the programming interface installed in the POX network controller. Unlike the robotics environment, a non-invasive proactive application of dynamic workflow generation is presented, which is designed to support infrastructure and management mechanisms already implemented in the environment. Non-invasive behavior means that the DWGE will communicate with the environment using an interface installed in the managed environment without altering the environmental structure (i.e., no hardware or software changes). Also, the managed environment can still operate, while retaining its original functionality, independent of the WMF. Proactive management can add new functionality based on the request from the environment and using its resources. Some of the workflows do not require dynamic updates because they are responsible for the consistency of prerequisites, e.g., checking the POX modules or collecting data. In this case, the dynamic generation of the workflow is

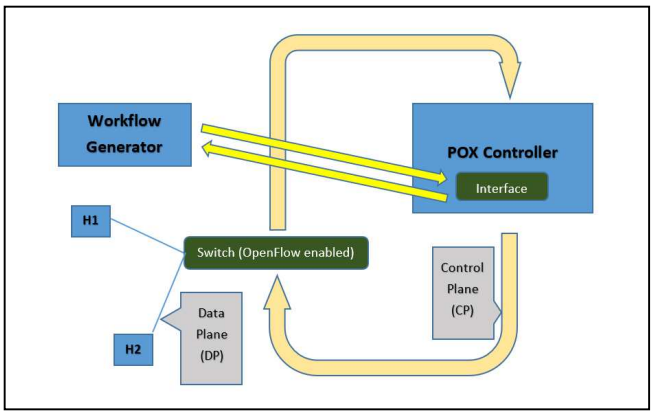

Fig. 9 Diagram showing an SDN – workflow generator relationship

TABLE 2 DATA GATHERED DURING THE NETWORKING ENVIRONMENT (SDN) TESTS

Number of Tunnels	Number of the switches in the Tunnel	ping delay after incident (s)	Prerequisite delay (s)	packet lost
1	4	1.9	18	0
2	5	1.85	20	0
3	6	1.95	19	0
4	7	2.1	21	0
5	8	2.2	20	0
6	9	1.98	21	0
7	10	2.2	19	0

used to extend the existing functionalities, thus resulting in the improvement of the environment.

1) Prerequisites

This scenario is designed to play a supporting role for the POX software where network traffic is managed by the controller. For example, the default route for hosts connected to switches S1 and S2 is through switch S3 (shortest path). The DWGE has no direct impact on the environment elements (switches are part of the network infrastructure).

However, it enables an indirect management role using POX modules (activate or deactivate them). Firstly, the communication interface (located in the POX, playing the client role) establishes the connection (handshake) with the DWGE. The server then generates a set of prerequisite workflows. Most important is the execution of the workflow that is responsible for creating a local repository because this will be used to store all data gathered during the execution of the prerequisite workflows shown in Fig. 10. The next executed workflow is responsible for checking that all required POX modules are activated, if not it will try to activate them. A list of the modules is predefined under the policy located in the server repository. The workflow generator uses this list to instruct the POX controller as to when and what modules must be activated. The purpose of using such modules is to map a network topology using a link layer discovery protocol (LLDP), create a list of switches and discover the links between them.

Before completing a prerequisite and moving to the next stage, a time delay is applied. This is related to the fact that several algorithms are time-consuming and events that possibly affect the network environment can be detected with the delay.

2) Network topology and traffic generation

The referenced topology is presented in Fig. 11 which has hosts (H1-H6) connected through switches (S1-Sn). By default, the POX controller manages the network so that all traffic between Network A and Network B is routed using switches S1, S3 (located in Network C) and S2. In this scenario, two types of workflows were used: management-oriented and monitor-oriented [15]. The aim of the management-oriented workflow is to build a path (or tunnel) between H1 and H2, which must be dynamically updated when a link problem is detected because the previously known network topology will not be valid anymore. To build a dedicated tunnel between hosts H1 and H2 (that would be available for these two hosts only), the workflow generated by the WES creates a tunnel (static route) through the unused, possibly shortest path, S1-S4-S5-S2 (switch S3 has been excluded because it is part of a default route managed by the POX controller). This workflow is executed once per minute because the installed entry in the switch table will expire within 60 seconds, then it will be deleted from the switch. This task is repeated until an incident (link failure) is detected. Executed in parallel to the management-oriented workflow is a monitor-oriented workflow, which is checking LLDP packets against link failure. When such a failure is detected, this information is stored in the Broken_Links folder (Fig. 10). Incident detection does not affect monitor-oriented workflows but executing the management-oriented workflow will be impossible. This type of workflow uses the repository data to validate itself. When a broken link is reported, the workflow will change its own state to *damaged*, then the environment (interface) will send a request to the server to update this workflow, considering the topology changes caused by the incident.

3) Environment Test

In computer networks, link failure detection is critical for traffic management. Detection of such a problem is most important to maintaining the integrity of the system. In this experiment, it was decided to use a dynamically generated workflow to detect a broken link and recommend an alternative route for the affected traffic. For test purposes, a Python script manually terminated a link between the last two switches in the tunnel (see Section VI.B.2). For the traffic

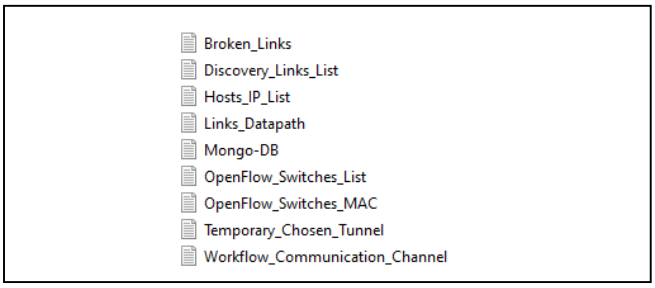

Fig. 10 Environment repository folders

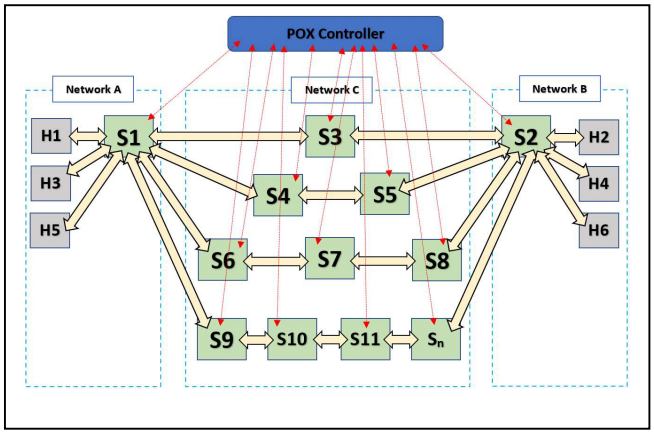

Fig. 11 SDN topology

generation, ICMP ping packets were sent between H1 and H2 through the established tunnel. In parallel, the same command was executed to generate traffic between H3 and H4 to show that default traffic is carried out without interruption. As expected, when the link had been terminated, the traffic in the tunnel was lost. However, default traffic continued without interruption. The system reacted to the incident by the generation of a new workflow that redirected traffic to the new tunnel S1-S6-S7-S8-S2.

Dynamic workflow generation is a reliable and promising management method in modern computer networks such as SDN. The performed tests focused on the suitability of the proposed method for dynamic workflow generation rather than on a comparison with existing solutions in SDN. The tests demonstrated the effectiveness of this method, both in detecting incidents and preventing them. The implemented tests were not performance-oriented but focused on the application of dynamically generated workflows to network management, especially for detecting network failures. Regarding results gathered, increasing the number of switches that were part of the tunnel did not affect the incident detection mechanism and dynamic workflow update in relation to time or delay. As mentioned in Section VI.B.1, several time-consuming algorithms must be executed for prerequisite checking (system needs on average 19 seconds to activate all modules and enable a traffic flow – see Table 2). Events that potentially affect the network environment can be detected with the delays presented in Table 2; the values are comparable and are not dependent on the number of network elements.

VII. Conclusion and Future Work

In this paper, workflow generation, with particular emphasis on the implementation of self-mitigating behavior when an execution exception occurs, was explored. As shown in the experiments, the proposed framework demonstrated its potential because it successfully implemented the tasks of environment management and self-update based on feedback from the environment. Dynamic workflow generation strictly depends on the requirements and specifications of the managed environment. The experimental evaluation shows two, (but not limited to) management-oriented applications for dynamic workflow generation and its relatively simple not intrusive implementation. The idea presented in this paper can be used as a powerful solution for infrastructure management, testing or debugging. The experiments described here provide a starting point for future exploration of a dynamic workflow generation framework that is not tied to one particular managed environment.

Future work will research a dynamic (adaptive) rating system (the quality measurement mechanism of dynamically created workflows). Automatic environment recognition, to replace the currently used static approach, replacement of the statically created workflow component with a dynamic approach will also be investigated.

Acknowledgments

This work was supported by the Irish Research Council, under grant number EPSPG/2017/336, and by Ericsson, the industrial partner.

References

[1] C. U. Press, "Cambridge Dictionary," Cambridge, 01 Jan 2020. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/workflow. [Accessed 01 06 2020].

[2] I. W. Salemme, "What types of workflow exist?," 17 Apr 2020. [Online]. Available: https://www.pipefy.com/blog/types-of-workflow-exist/. [Accessed 20 Jun 2020].

[3] Integrify, "Integrify workflow management benefits," Integrify, 15 Feb 2020. [Online]. Available: https://www.integrify.com/workflow-management-benefits/. [Accessed 20 05 2020].

[4] "Online project management made easy," Workflow max, 2020. [Online]. Available: https://www.workflowmax.com. [Accessed 25 06 2020].

[5] Asana, "Keep your team coordinated, wherever you are," Asana, 2020. [Online]. Available: www.asana.com. [Accessed 25 06 2020].

[6] The Apache Software Foundation (ASF) , 2021. [Online]. Available: https://taverna.incubator.apache.org. [Accessed 10 01 2021].

[7] B. Wallace, "Bad workflows: The hidden time wasters slowing you down," Ricoch, 2020. [Online]. Available: https://www.ricoh-usa.com/en/insights/articles/bad-workflows?utm_expid=._agfg6o8rceclex3zfok2a.0&utm_referrer=https%3a%2f%2fwww.google.com%2f. [Accessed 26 05 2020].

[8] E. Fels, "Top Small Business Management Mistakes to Avoid," www.businessknowhow.com, 06 Mar 2019. [Online]. Available: https://www.businessknowhow.com/manage/small-business-management-mistakes.htm. [Accessed 28 05 2020].

[9] K. Nash, "5G accelerates network management challenges," Ericsson, 27 Mar 2018. [Online]. Available: https://www.ericsson.com/en/blog/2018/3/5g-accelerates-network-management-challenges. [Accessed 16 06 2020].

[10] C. Hagen, G. Alonso, "Exception handling in workflow management systems," *IEEE Transactions on Software Engineering,* vol. 26, no. 10, pp. 943 - 958, 2000.

[11] M. z. Muehlen, orkflow-based Process Controlling: Foundation, Design, and Application of Workflow-driven Process Information Systems, Business & Economics, 2002.

[12] Deniz Appelbaum, Alexander Kogan, Miklos Vasarhelyi Zhaokai Yan, "Impact of business analytics and enterprise systems on managerial accounting," *International Journal of Accounting Information Systems Elsevier,* vol. 25, pp. 29-44, 2017.

[13] Guido Dinkhoff, Volker Gruhn, Armin Saalmann, Michael Zielonka, "Business process modeling in the workflow management environment Leu," in *International Conference on Conceptual Modeling*, Berlin, 2005.

[14] P. Carayon, "Human factors analysis of Workflow in health information technology implementation," in *Handbook of human factors and ergonomics in health care and patient safety*, London, CRS Press, 2012, p. 824.

[15] Thenuwara Hannadige Akila Sanjaya Siriweera, Incheon Paik, Banage Thanne Gedara Samantha Kumara, "Constraint-Driven Dynamic Workflow for Automation of Big Data Analytics Based on GraphPlan," in *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, 2017.

[16] Mainak Adhikari, Santanu Koley, "Cloud Computing: A Multi-workflow Scheduling Algorithm with Dynamic Reusability," *Arabian Journal for Science and Engineering,* vol. 43, no. 2, pp. 645-660, 2018.

[17] Ratnakar, Yolanda Gil, Varun, "Dynamically Generated Metadata and Replanning by Interleaving Workflow Generation and Execution," in *Proceedings of the Tenth IEEE International Conference on Semantic Computing*, Irvine, 2016.

[18] Peter J. Kammer, Gregory Alan Bolcer, Richard N. Taylor, "Techniques for Supporting Dynamic and Adaptive Workflow," in *Computer Supported Cooperative Work (CSCW)*, Irvine, 2000.

[19] D. Weyns, N. Bencomo, R. Calinescu, J. Camara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J.M. Jezequel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli, "Perpetual Assurances in

Self-Adaptive Systems," 18 Jan 2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-74183-3_2. [Accessed 20 06 2020].

[20] T. A. S. Foundation, "web.archive.org," Taverna, 15 May 2020. [Online]. Available: http://web.archive.org/web/20200515113032/https://taverna.incubator.apache.org/introduction/taverna-in-use/. [Accessed 28 Jan 2021].

[21] Johann Eder, Walter Liebhart, "Contributions to Exception Handling in Workow Management," in *EDBT Workshop on Workflow Management Systems*, Valencia, Spain, 2006.

[22] Bob Lantz, Brandon Heller, Nikhil Handigol, Vimal Jeyakumar, "Mininet - Virtual Network Emulator," 2020. [Online]. Available: http://mininet.org. [Accessed 03 04 2020].

[23] J. McCauley, "The POX network software platform," POX GitHub repository, 23 Nov 2017. [Online]. Available: https://github.com/noxrepo/pox. [Accessed 04 04 2020].

[24] decisions.com, "Decisions," 2020. [Online]. Available: https://decisions.com. [Accessed 26 06 2020].

[25] Soon Ae Chun, Vijayalakshmi Atluri, Nabil R. Adam, "Dynamic Composition of Workflows for Customized eGovernment Service Delivery," 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.3765&rep=rep1&type=pdf. [Accessed 20 Jun 2020].