Azerbaijan Journal of High Performance Computing, Vol 1, Issue 2, 2018, pp. 140-157
https://doi.org/10.32010/26166127.2018.1.2.140.157

Cloud-based Flowbster Portal to Design and Deploy Scientific Workflows

József Kovács, Zoltán Farkas, Enikő Nagy, and Bendegúz Gúlyás

Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
fjozsef.kovacs@sztaki.mta.hu, zoltan.farkas@sztaki.mta.hu, eniko.nagy@sztaki.mta.hu, bendeguz.gulyasg@sztaki.mta.hu

*Correspondence:
József Kovács, Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary,
fjozsef.kovacs@sztaki.mta.hu

Abstract

A workflow system called Flowbster has been designed to create efficient data pipelines in clouds. The entire Flowbster workflow is dynamically built by using virtual machines on a target cloud. The paper describes a recently designed and developed web-based science gateway to support Flowbster. It provides a high-level graphical environment to handle different levels of abstractions, like workflows representing the layout and deployment representing the infrastructure realizing the workflow. Detailed overview of the user interface, the portal architecture and its internal operation are given in the paper. Moreover, an insight is provided on the selection and cooperation of the web modules and on the integration of the portal in the firebase environment developed by Google.

Keywords: workflow, science gateways, orchestration, firebase.

1. Introduction

Cloud computing [1] is getting more and more popular since it can dynamically provide the required amount of resources in a very exible way. Beyond the theoretically unlimited resources seen from the cloud user point of view, the wide variety of operating systems, networking and performance enables to support practically any type of applications. One of the application areas is covered by workflow systems to process scientific data sets by a network of computational nodes.

Workflow systems [2] are typically used to process data consists of several steps where each step performs different calculation. The network of steps formalized by a workflow provides the desired overall calculation. The workflow is formalized with nodes performing the calculation and edges among the nodes representing data transfer. The most popular workflow operation is based on dataflow principle where each computational node starts processing when all the necessary inputs are available.

Flowbster [3] is a cloud-oriented workflow system based on dataflow principles. Regarding the way of scheduling the computation, instead of using the enactor

based workflow concept Flowbster applies the service choreography concept where the work- flow nodes themselves (virtual machines running Flowbster) are aware of their surrounding neighbours and are aware about which pieces of data to send to which neighbours. There is no central enactment component performing the orchestration of data processing as it is in most cases. Orchestration is hardwired at deployment time by the Occopus cloud orchestration tool.

Occopus[4] is a powerful, easy-to-use, configurable, hybrid, multi-cloud orchestration tool. It is an open source software providing features for configuring and orchestrating distributed virtual infrastructures both on single and multi-cloud environments. One of its main advantages is that it has a plugin based architecture enabling the addition of new cloud providers relatively easily. It has a simple description format for defining infrastructures, nodes, contextualization and health checking. Currently, Occopus can support the deployment of a Flowbster workflow with its native set of features.

CloudiFacturing[5] is an EU project which aims to help participants to optimize their production processes and to increase producibility using Cloud/HPC-based modelling and simulation. With this support the project contributes to the competitiveness and resource efficiency of manufacturing SMEs. To pursue the project's mission, computationally demanding production engineering and simulation as well as data analytic tools are provided as Cloud services to ease accessibility and make their use more affordable. The CloudiFacturing project empowers over 60 European organizations (many of them being manufacturing SMEs) and supports about 20 cross-national application experiments that are primarily selected via two Open Calls. The experiments are formed by partners having expertise on the following areas: Cloud/HPC, data analytic, simulation, modelling, security and business modelling.

One of the tools in the Cloudifactoring project to support the experiments is the Flowbster cloud-oriented workflow system powered by Occopus. With the combination of Occopus and Flowbster data processing workflows can be deployed on demand in a cloud. In order to further support the creation of new Flowbster workflows and to deploy them as simple as possible, a new Flowbster portal has been designed and developed. The motivation behind this work is to provide an easy-to-use science gateway where workflow creation and deployment are supported in a graphical way.

The paper introduces the entire Flowbster scientific portal where the concept and structure of the Graphical User Interface are introduced in Section 2. The architecture of the portal is detailed in Section 3 and the internal operation is described in Section

4. A few paragraphs are devoted to testing and performance in Section 5, then several related works are introduced in Section 6. Finally, conclusion is made in Section 7.

2. Overview of the Graphical User Interface

This section gives an overview of the most significant and valuable characteristics of the Flowbster Portal outlook including the workflow editor (node creation and

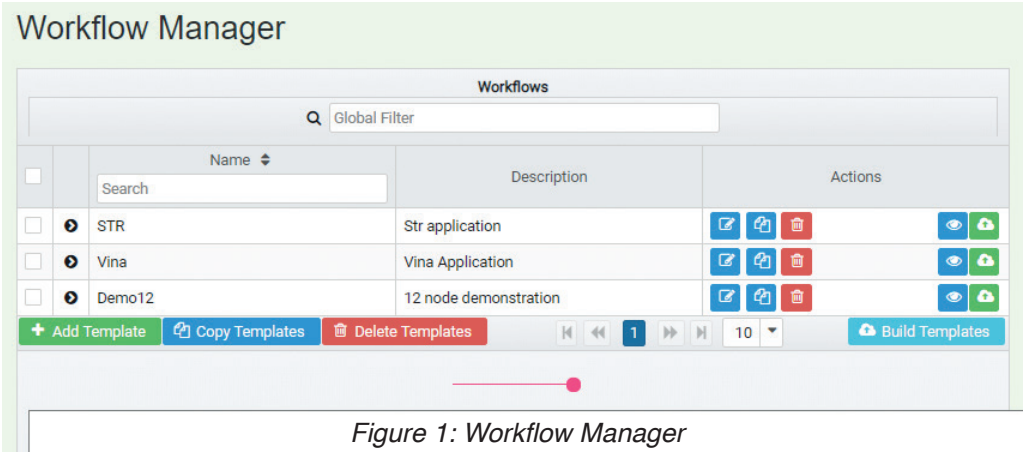

Figure 1: Workflow Manager

configuration) and the workflow deployment. The Flowbster Portal supports workflow handling at two layers: the workflow layer described in Section 2.1 and the deployment layer described in Section 2.2. At the workflow layer end-users can use the Workflow Editor to piece together the workflow nodes (called Flowbster nodes), set their configurations and create connections between them. At the deployment layer the workflows are deployed and launched in the target cloud based on the preconfigured parameters, using the Occopus cloud orchestration tool. At the Workflow Manager page, the end-users can list, edit, copy, delete, view or build their workflows (see Figure 1). Each workflow has a unique name and description can be also added to ease the distinction among them.

2.1. Workflow layer

2.1.1. Handling nodes

A user-friendly drawing canvas (see Figure 2) is available within the Flowbster Portal. On the canvas, users can add nodes (represented as boxes), input and output ports to the nodes (represented as green or red circles on the nodes) and relations between ports (represented as directed arrows). The nodes of the workflow represent an atomic calculation and will be realized by executing the appropriate application in a virtual machine within the Flowbster infrastructure. The links among the nodes represents data transfer and the names on the links are the name of the physical files that should be transferred between the nodes.

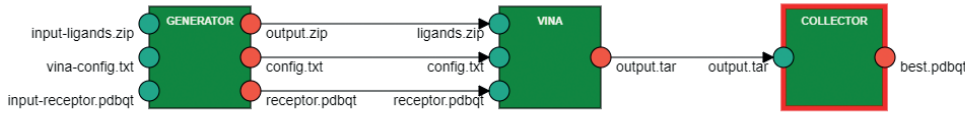

Figure 2: Flowbster nodes in a workflow

A node can be added to the workflow by double clicking on the canvas. The configuration options for a node are as follows (see Figure 3):

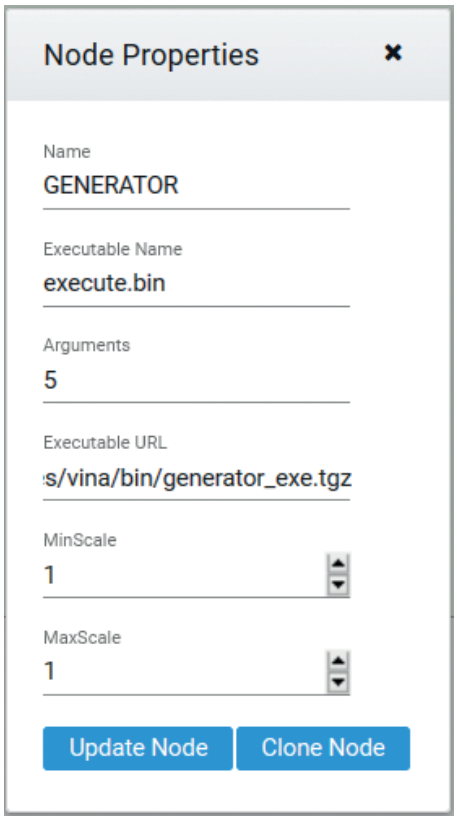

Figure 3: Generator node properties

- Name: name of the Flowbster node, which is unique across the entire Workflow.
- Executable name: name of executable for the application to be executed on the Flowbster entity.
- MinScale, MaxScale: these parameters tell the system what is the minimum and maximum number of instances to be created from the node.

The properties of a node can be re-edited anytime by double clicking on the node. In the pop-up window (Figure 3) the configuration can be update and can be saved by the ''Update Node" button. Cloning of a node is possible with the ''Clone Node" button at the bottom of the window. Cloning a Flowbster node means copying it including properties, input and output ports and a newly generated unique name.

2.1.2. Handling I/O ports

The I/O ports of an existing Flowbster node can be added with the ''Add Input port" or ''Add Output port" buttons and can be deleted with the ''Delete port" button in the Workflow Editor menu (see Figure 4).

During the configuration of the input ports, the list of attributes to be set are as

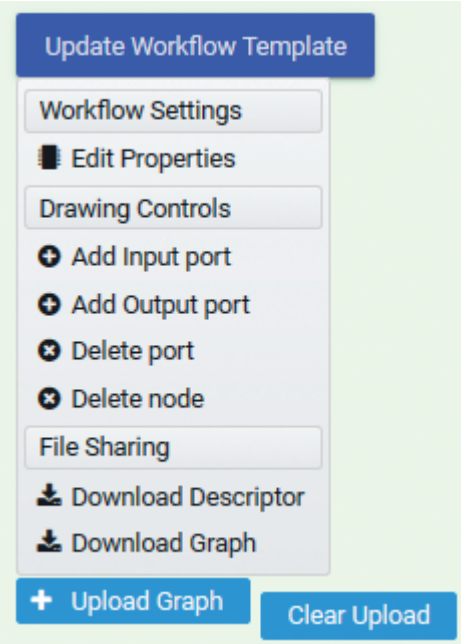

Figure 4: Workflow editor menu

follows (see left dialog in Figure 5).:

- DisplayName: name of input port to appear on the drawing area.
- FileName: name of file to be stored as input for the application on the Flowbster node.
- Collector port: true when the application on this node receives multiple inputs on this port for one run. False, otherwise.
- Format: format to be applied when naming the input files for the application.
- Filter RegExp: regular expression (filter) which helps Flowbster recognize the instances of the files on this ports. All files are considered as output which is selected by this filter expression.

The properties and options of the input or output ports on a node can be checked by clicking on the ports. Directed arrow between the ports can be defined by the drag and drop style from the input port to the output port. When the link creation is successful, a black arrow is shown on the canvas between the selected ports.

2.1.3. Flowbster workflow properties

Workflow has also properties to be set with the ''Edit Properties" button on the Work- flow Editor menu shown on Figure 4. The configuration options for the workflow are shown in a pop-up window (see Figure 6) where the attributes are as follows:

During the configuration of the input ports, the list of attributes to be set are as follows (see left dialog in Figure 5).:

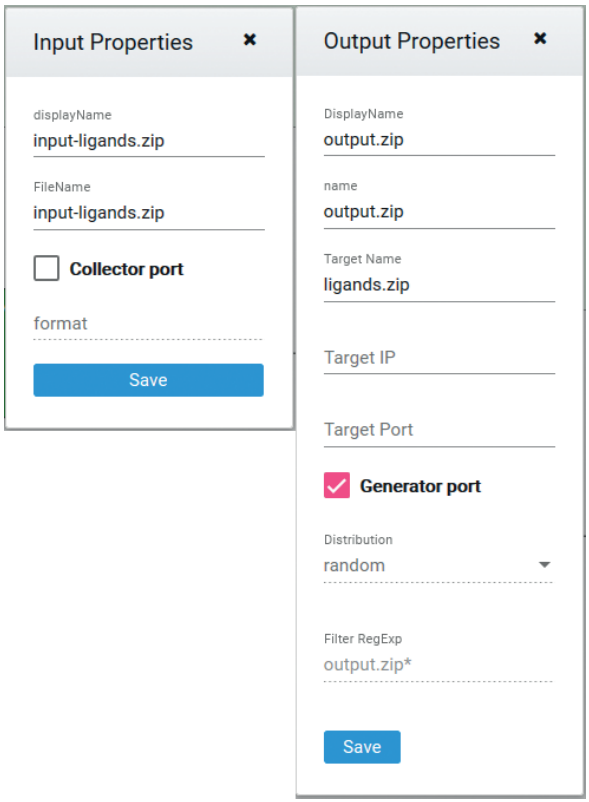

Figure 5: Input and Output port properties

- DisplayName: name of input port to appear on the drawing area.
- FileName: name of file to be stored as input for the application on the Flowbster node.
- Collector port: true when the application on this node receives multiple inputs on this port for one run. False, otherwise.
- Format: format to be applied when naming the input files for the application.

During the configuration of the output ports, the list of attributes to be set are as follows (see right dialog in Figure 5):

- DisplayName: name of output port to appear on the drawing area.
- Name: name of output file produced by the application on the Flowbster node.
- Target Name: name of input port of the target node where the output file must be sent.
- Target IP: when specified, the output is sent to this target ip address instead of the linked port. A global variable can be used here with the *variableName syntax.
- Target Port: when specified, the output is sent to this target port address instead of the linked port. A global variable could be used with the *variableName syntax.

- Generator Port: true when the application on this node generates multiple instances of this output file for each run. False, otherwise.
- Distribution: specifies the distribution mode among the multiple instances of the attached nodes. Possible values can be ”random” or ”round-robin”
- Filter RegExp: regular expression (filter) which helps Flowbster recognize the instances of the files on this ports. All files are considered as output which is selected by this filter expression.

The properties and options of the input or output ports on a node can be checked by clicking on the ports. Directed arrow between the ports can be defined by the drag and drop style from the input port to the output port. When the link creation is successful, a black arrow is shown on the canvas between the selected ports.

2.1.3. Flowbster workflow properties

Workflow has also properties to be set with the ’’Edit Properties” button on the Work- flow Editor menu shown on Figure 4. The configuration options for the workflow are shown in a pop-up window (see Figure 6) where the attributes are as follows:

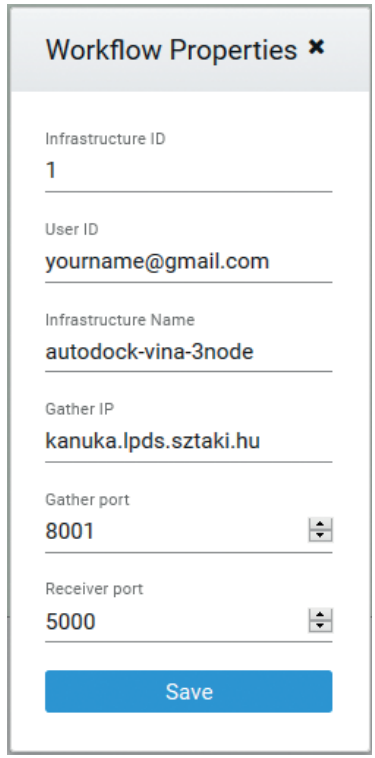

Figure 6: Workflow properties

- Infrastructure ID: a unique id for deployment, generated by the portal
- User ID: owner of the workflow, e.g. email address
- Infrastructure Name: name of the infrastructure used internally

- Gather IP: the address of the Flowbster Gather component (which collects the final results, for details see [3].
- Gather port: the port number of the Flowbster Gather component.
- Receiver port: the port number used by Flowbster nodes to receive the input data. System property, default is 5000.

Every workflow created by the user is stored in the backend database which contains the name and the detailed description of a Flowbster node. Furthermore, it contains a graph description in JSON format for the editor to visualize, and a YAML descriptor for Occopus to be deployed in the target cloud. The completed descriptors can be downloaded by the ''Download Descriptor" and by the ''Download Graph" buttons on the Workflow Editor menu (in Figure 4). Importing existing graphs can be also used, by using the ''Upload Graph" button and uploading a JSON file.

2.2. Deployment layer

The second layer is the deployment layer which represents the instantiation of the Flowbster nodes configured at the workflow layer. The Flowbster nodes are realized by virtual machines built by the Occopus orchestration tool.

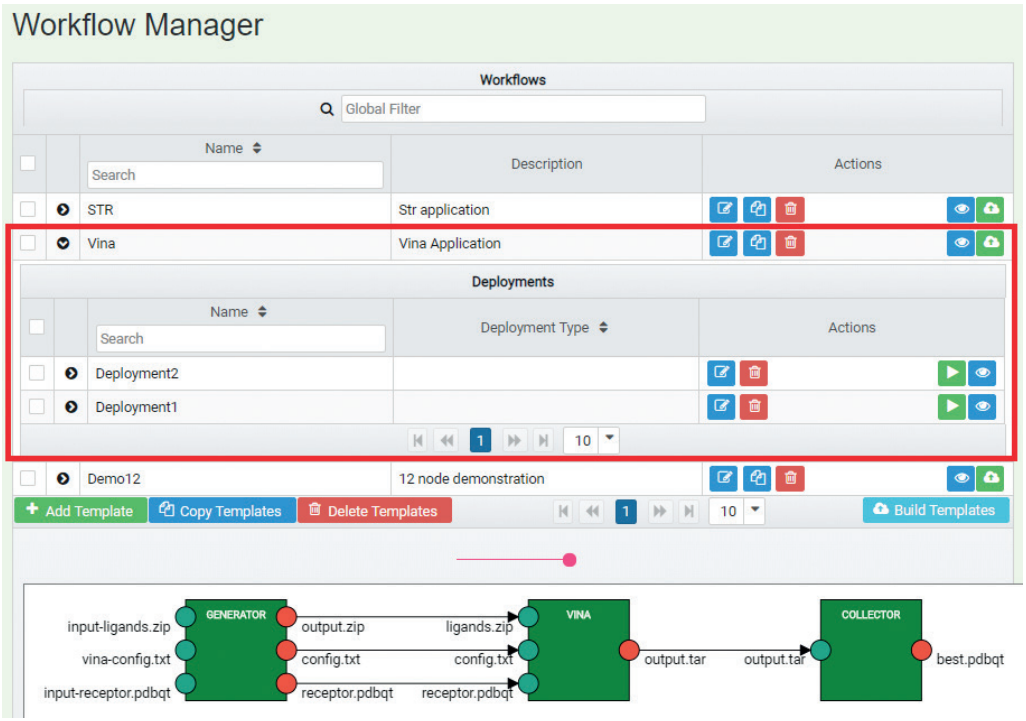

Figure 7: Workflows and their deployments

The ''Build" button on the Workflow Manager page (in Figure 1) is used for building the workflow in the target cloud. The details on how the workflow

deployment is implemented is described in Sections 3 and 4. At deployment the user is requested to provide a name for the deployment and then the workflow deployment procedure starts.

After a successful deployment the Workflow Manager page lists the running deployments of each workflow. With the drop-down arrow icon of the selected workflow item the end-users can open the list of deployments of the workflow. The framed area in Figure 7 shows the list of deployments of a selected workflow. In this Figure example, two deployments have been built by the portal. Deployments can be configured, manipulated, used and dropped when no longer needed. Deployments are equivalent with the Flowbster workflows drawn by the user and operates as described in [3].

The Flowbster portal creates Occopus description based on the workflow at the end of edition. Regarding the number of instances, the built infrastructure will consist of as many node types as many different Flowbster node is created, and as many nodes as the scaling parameter was set to for each node. The links between the virtual machines will be configured based on the connection arrows between the nodes. Data flow between two Flowbster nodes are managed by the Flowbster workflow system itself in the background.

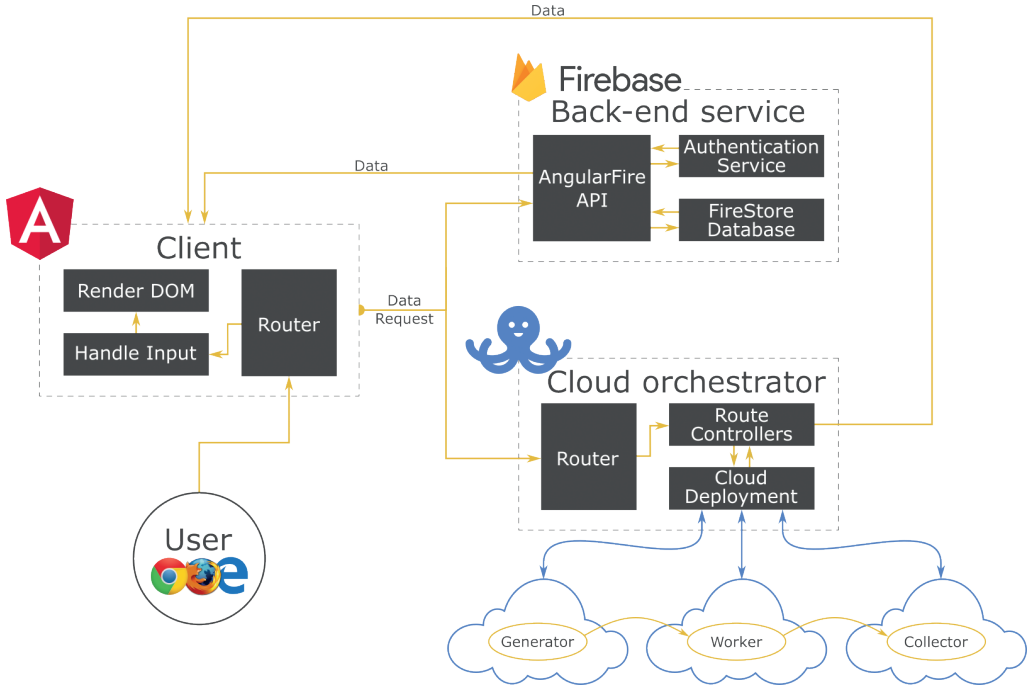

Figure 8: Architecture of the Flowbster portal

As the Flowbster portal aims to be a web-based tool for users, the minimum requirement is to have a client layer in the portal through which users can interact with the underlying Occopus orchestrator in a user-friendly way. However, the

portal cannot run solely on the client side, as we need to persist users' developed workflows or monitor their execution while the user is not logged in to the portal, so a server side component and an attached database is also necessary. Thus, the Flowbster portal is built up using a frontend layer, a backend layer and a database. A high-level overview of the Flowbster portal is shown in Figure 8.

3.1. Frontend layer

The frontend layer (Client in Figure 8) of the Flowbster portal is the entry point for the users, so it must provide adequate user experience by providing feedback on users' action as fast as possible. This can be achieved by relying on a proper framework, like Angular[6], React[7] or Vue.js[8]:

Angular is a widespread framework developed by Google, used by companies like Paypal or Nike. It's using the TypeScript[9] programming language, which implies functional programming paradigms, thus requires some effort to get familiar with. Angular is built on top of quick and standardized web components, offering convenient command-line tools for development.

React is developed by Facebook and is much easier to learn than Angular. It is not a complete framework, but is put together by a large set of components. As a consequence, one has to learn all these components. Additionally, React doesn't have built-in compile facility, like Angular has.

Vue.js is a framework trying to bring together the advantages of Angular and React. Although it has a large user base behind, it is not mature enough.

Based on the properties of the different frameworks considered, we have decided to use the Angular framework for creating the frontend layer of the Flowbster portal.

Graph creating component The key component of the Flowbster portal's frontend layer is a graph editor and presenter component. This component is used to create new workflows, to modify existing ones, or to preview any workflows. Instead of reinventing the wheel, we decided to select a library which provides as many functionalities of such an editor as possible. The basic requirements we set were this: easy integration, offer functionalities for creating new elements in the graph, offer the functionality to connect elements of the graph, enable attaching additional boxes to elements of the graph, and finally provide as many callback functions for the user actions performed on all elements of the graph as possible. This last requirement is important, as with this, we get very detailed notifications on what the user is actually performing as possible. Additionally, we can react on the user's actions.

Fortunately, there are a number of libraries which offer similar functionalities. The libraries we considered were the followings:

Raphael as a library is aiming to support creating vector graphics easily. Raphael [10] enables creating specific chart or image crop and rotate widgets. Although it is easy to use, it doesn't offer our required features out of the box.

mxGraph is offering a solution based on interactive javascript and HTML5 standards. mxGraph[ll] has a client- and server-side implementation for creating graphs.

JointJS is an open-source library. JoinUS[12] is widely used graph creating

framework, backed by a large community, offering commercial versions as well. It has a lot of extensions (for modeling discrete event systems, petri nets, etc.), has NodeJS support, is able to dump and to import the created graphs in JSON format, is a pure client-side library, and offers callback functions for all the operations performed by the user.

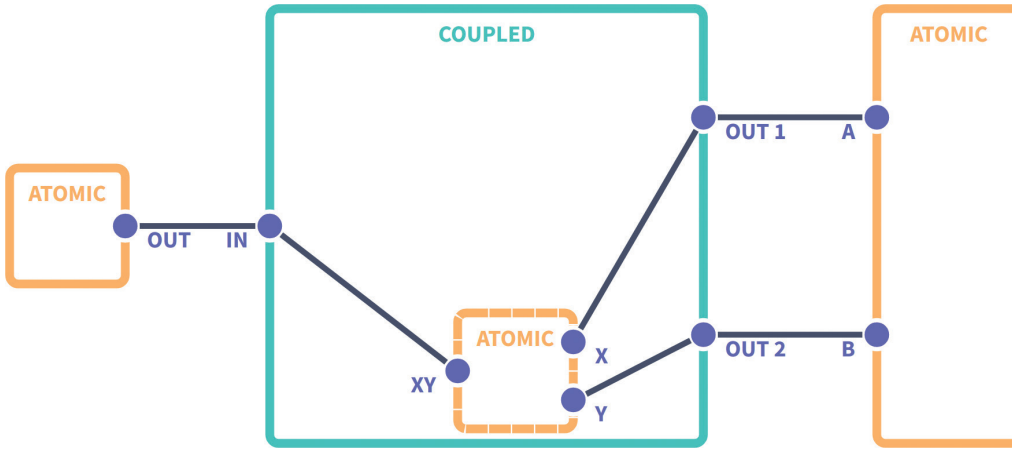

Figure 9: JointJS DEVS demo

Based on the above, we have decided to use the JointJS library for creating the graph development component in the Flowbster portal's frontend layer. The extension we rely on is the DEVS (Discrete Event System Specification). An example presenting the possibilities of this extension is shown in Figure 9. As it can be seen, one can create boxes for jobs in a workflow, attach input and output ports to them, and connect these together. There are many additional features, which we are not using at the moment in the Flowbster portal.

3.2. Backend layer

The task of the backend layer (Firebase back-end service in Figure 8) in the Flowbster portal is to respond to users' requests as quick as possible, partially based on the information stored in the database. There are numerous options for this, using different Python frameworks (Flask, Django), Go, Java, or Node.js.

We have choosen Node.js[13] as its non-blocking I/O operations satisfy our needs regarding speed and response time. Node.js can also be used to build RESTful services, it has a big number of extensions thanks to its community, it uses the Javascript programming language, it integrates with Angular very well, and it also supports JointJS.

The backend layer also encapsulates communication with the Occopus cloud or- chestrator tool. When necessary (upon building an infrastructure of a workflow or destroying the infrastructure), the backend components can invoke the RESTful API of Occopus for creating new infrastructures, subscribing to infrastructure-related events, or removing infrastructures already created. For this, we need to

know the URL of Occopus' RESTful API, which can be set through a configuration interface. The parameters for the different calls are automatically generated by backend components: a YAML description when creating an infrastructure or an identifier when subscribing to events, or destroying existing infrastructures.

3.3. Database

Since persisting the users' developed workflow graphs is needed, the database component (denoted as FireStore Database in Figure 8) becomes mandatory in the architecture. As written earlier in the subsection 3.1, JointJS is capable of dumping and parsing the graph descriptions in JSON format, so using a database with support for this format is a natural requirement. We have considered the following databases: CouchDB[14] and MongoDB[15].

CouchCB is an easy-to-use database, putting focus on providing consistency. It is mostly used to store predefined queries on rarely changing data. CouchDB also offers version control.

MongoDB is a widespread JSON-based database. It offers convenient ways to run dynamic queries, and scales well with the size of the data stored.

We have chosen MongoDB, as it can be used in the Node.js-based backend layer easily thanks to the Mongoose ORM.

Architecture summary In order to eliminate the need to host the whole Flowbster portal individually, and to enable its usage by CI tools, we decided to use Firebase[16] as the backend provider. The main advantage of relying on Firebase is its FireStore component, a flexibe, autoscaling noSQL database, enabling real-time synchronization between the client and the server. Additionally, it enables offline operation as well in case of the user's internet connection is lost. Firebase incorporates user authentication as well, so there is no need to add one such framework in the Flowbster portal. Finally, Firebase offers push notifications through Firebase Cloud Messaging, allowing us to present notifications for the users on events relating to the Flowbster workflows.

4. Operation

In this section we overview a more detailed description of the Flowbster portal's architecture, through introducing the data types necessary for modeling the different entities, and modules implementing the portal's operation.

4.1. Types

Angular's TypeScript support enables us to use our own data types. This not only helps to create more understandable code, but any type-related errors can be determined during compile time. In the Flowbster portal, we defined the types as shown in Figure 10.

The basic description of the types is as follows:

User represents the users of the Flowbster portal,

WorkflowEntry represents a workflow's properties, including the JSON description used by JointJS and the YAML description used by Occopus,

FlowsbsterNode contains all the properties of a Flowbster workflow node, including the alias, executable name, arguments, executable URL and scalability

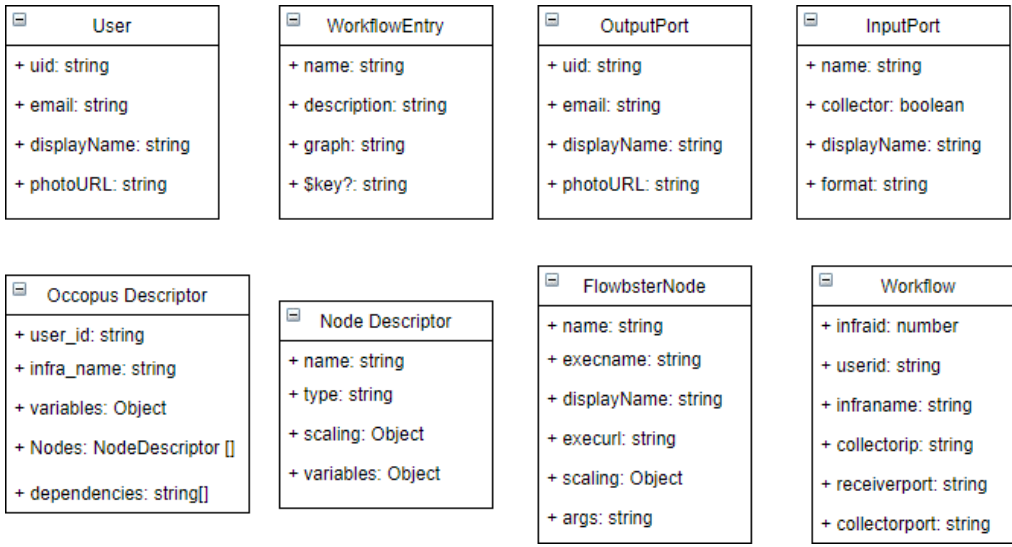

Figure 10: Data types in the Flowbster portal

parameters (see Figure 3),

Workflow represents global properties of a workflow, for example the address of the Gather component,

InputPort represents an node input port's properties, like alias, filename, and optionally the collector-related properties,

OutputPort is similar to the InputPort, but related to output ports,

NodeDescriptor description of one node of the Flowbster workflow in YAML format required by Occopus to deploy the workflow,

OccopusDescriptor contains the Occopus description of the whole workflow.

4.2. Modules

The different modules in the Flowbster portal implement the different operations inside the system. There are modules which invoke the proper module implementing the requested function based on the user's action path, which implement user authentication, which contain all the input forms, which handle the JointJS component, or implement workflow management and execution. We will now describe the most important modules in a bit more details.

App Routing Module The task of the App Routing Module is to distinguish operations based on the URL the user is accessing. Beside this, this module is also responsible for collecting any necessary information from the URL, if it is needed. Figure 11 shows the different URL paths this module handles.

The / and / signin and any other paths result in presenting a login page. The /authenticated path is protected by authentication, routes behind this path can only be accessed after login. For example in Figure 11, the path containing URL /workflow-detail / : id/ : operation loads the workflow detail component, but also contains necessary parameters - e.g. the identifier of the workflow to show the

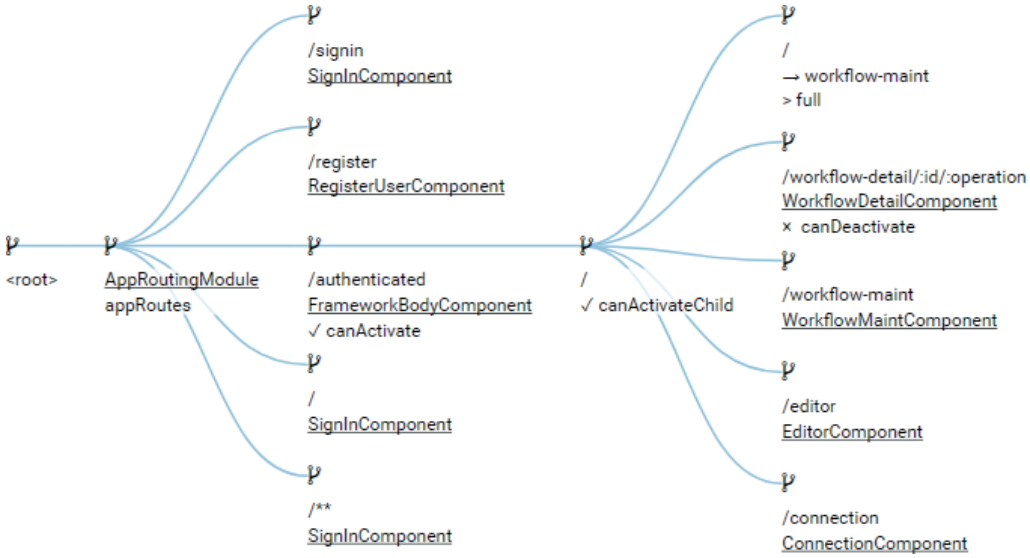

Figure 11. Routes of the Flowbster portal

details for (: id), or the operation related to the workflow (: operation - creating a new workflow, or modifying an existing one).

There are some additional services included in this module. The Auth Guard Service is responsible for checking if the selected target path has every necessary prerequisites filled in order to be loaded (for example, is the user authenticated?). Other services are used to get the parameters from the path, or to check if the user has ''saved" all of the modifications (for example, when the workflow graph is modified, but not saved, and the user wants to navigate to an other view, we can present an unsaved work alert for the user).

App Module The App Module represents the Flowbster portal and serves as an entry point for the portal. The module imports all the other modules necessary for the operation, as shown in Figure 12.

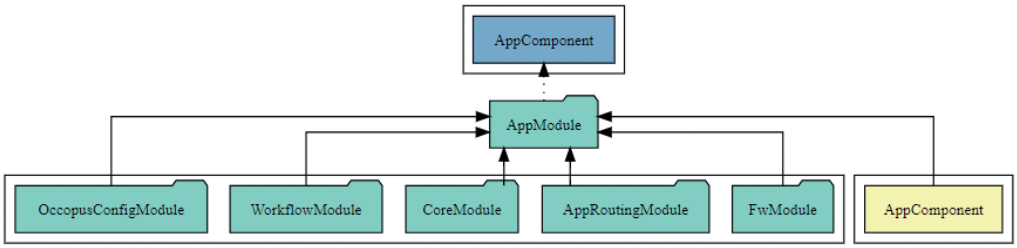

Figure 12. App Module

Flowbster Forms Module This module contains all the forms occurring in the Flowbster portal. Figure 13 gives an overview of included elements like form for defining node, input, output and workflow properties.

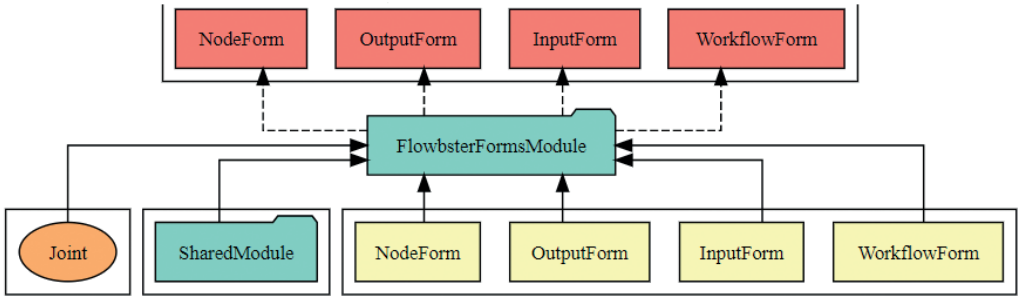

Figure 13. Flowbster Forms Module

This module also contains a service called Joint. This service class represents the abstraction layer above the JointJS library described in subsection This service is accessed by other components in the system if they want to interact with the library in any way. Also, other components can subscribe to different events triggered by the JointJS library through this service. Such events can be used to perform different sanity checks (for example check if there is a loop in the graph once the user has connected an output port to an input port, or to check if the workflow node's name matches other nodes' names to ensure identical node names).

Graph Interaction Module This module collects the graph drawing canvas and any toolset related to it. As seen in Figure 14, it includes the Flowbster Form Module as well, in order to provide graph editing and property setting functionalities. It also contains a DescriptorService service, which can be used to get a YAML-based description of the workflow processed by Occopus.

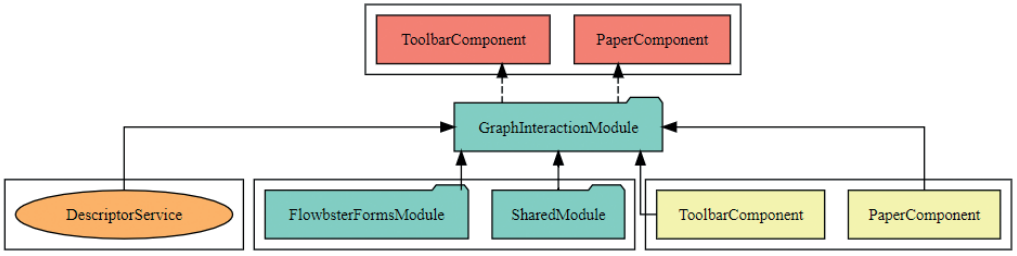

Figure 14. Graph Interaction Module

Editor Module The layout of the Editor Module is shown in Figure 15. As it can be seen the Editor Module incorporates the Graph Interaction Module as well, but it is also responsible for notifying the users on different events by displaying Toast messages, or pop-up dialogs.

Workflow Module The Workflow Module is shown in Figure 16. This module is responsible for coordinating workflow management tasks. It contains services for communicating with the Occopus service, the database and for subscribing to notifications coming from Occopus. Additionally, this module contains component for displaying workflow details and workflow maintenance.

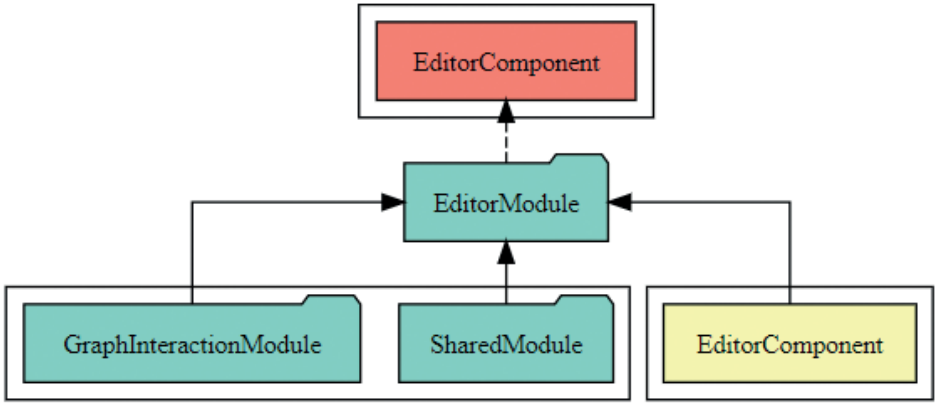

Figure 15: Editor Module

The Occo service of the Workflow Module contains all the necessary logic for communicating with the RESTful interface of Occopus. This service contains in its configuration the endpoint of the Occopus service (host and port) and the logs of the Regarding the performance of the portal, we completely rely of the features of Firebase. Thanks to its design, both the backend and database layer can dynamically scale as the load increases (more users start to use the portal). Additionally, the reactive implementation of the user interface makes sure that users' actions leave the client layer (the users' browser) only when it is really necessary. As a consequence, in the Firebase-based Flowbster portal we do not expect any performance issue from the users' point of view. We did not consider the performance of the Flowbster workflow itself, as it has already been discussed in [3].

6. Related work

Nowadays, there are numerous scientific gateways and portals offering execution of workflows on both private and commercial clouds. A short overview is given below on science gateways specialized on scientific workflows and utilizing clouds.

The WSPGRADE/gUSE [17] general-purpose science gateway offers access to a large set of DCIs (Distributed Computing Infrastructures) including clusters, supercomputers, grids and clouds. Cloud access is implemented by integrating its DCI Bridge service with the CloudBroker Platform (CBP) [18]. This approach has the advantage that all the clouds (Amazon, IBM, Eucalyptus, OpenStack, OpenNebula) managed by CBP became accessible for the WS-PGRADE/gUSE workflow enactor for utilisation and all the cloud access features (e.g. SaaS access, user and security management, etc.) provided by CBP became available for the WS-PGRADE/gUSE users.

Apache Airavata [19], inherited from the Open Gateway Computing Environments (OGCE) workflow system [20] focuses on the baseline tools of application, workflow, job and data management systems. Airavatas primary goal is to support long running applications and workflows on distributed computational resources including clouds. The Airavata XBaya workflow system provides a unique pluggable architecture for selecting the orchestration engine. One of its advantages is that XBaya workbench builds an abstract directed acyclic graph (DAG) which is independent of any work- flow runtime and the selected compiler modules are capable of producing workflow execution scripts for target runtimes.

The Pegasus [21] Workflow Management System offers execution of workflow nodes on both private and commercial cloud systems [22], Actually, the HUB zero Platform [23] itself offers its services through virtualized environment, where experiments run through these services may use cloud resources for their computation with the help of Pegasus. To enact workflows, Pegasus performs the scheduling of jobs in the workflows taking into account numerous characteristics of the resources and constraints of jobs while HUBzero provides a web-based ecosystem including many services like analytic tools, data publishing, resource sharing and collaboration of participants.

There are workflow solutions where the aim is to better organize workflows in cloud environments. For example, the work described in [24] deploys the workflow enactor together with the workflow into the target cloud system. This resulted in increased execution performance of workflows in the cloud. Another example to find new, innovative ways of executing workflows in clouds is described in [25] and [26] by Qasha et al. This approach targets service orchestration but the workflow tasks are submitted in a novel way to the cloud. When a workflow task becomes executable, a cloud orchestrator (Cloudify) dynamically deploys the workflow task node in a docker container in the cloud and initiates the associated executable to run inside the container.

7. Conclusion

The Flowbster portal aims to provide a high-level web-based graphical user interface for editing, building and maintaining Flowbster workflows. Flowbster workflows are edited by the integrated JointJS open-source library. The backend service persists data in MongodDB, while the workflows are deployed by the Occopus orchestrator tool in the cloud. Bidirectional communication is implemented between the portal and Occopus. The portal has been carefully designed to be modular and easily maintainable and extendable.

Acknowledgement

This work was partially funded by the European CloudiFacturing project, Grant Agreement No. 768892 (H2020-FoF-2017). We thank for the usage of MTA Cloud (https://cloud.mta.hu/) that significantly helped us achieving the results published in this paper.

References

[1] Buyya, R., Broberg, J., & Gościński, A. (2011). *Cloud computing: Principles and paradigms.* Hoboken, NJ: Wiley.

[2] Liu, J., Pacitti, E., Valduriez, P., & Mattoso, M. (2015). A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing, 13*(4), 457-493. doi:10.1007/s10723-015-9329-8

[3] Kacsuk, P., Kovács, J., & Farkas, Z. (2018). The Flowbster Cloud-Oriented Workflow System to Process Large Scientific Data Sets. *Journal of Grid Computing, 16*(1), 55-83. doi:10.1007/s10723-017-9420-4

[4] Kovács, J., & Kacsuk, P. (2017). Occopus: A Multi-Cloud Orchestrator to Deploy and Manage Complex Scientific Infrastructures. *Journal of Grid Computing, 16*(1), 19-37. doi:10.1007/s10723-017-9421-3

[5] Kovács, J. (n.d.). The Cloudifacturing EU project homepage. Retrieved from https://www.cloudifacturing.eu/159

[6] The Angular Framework homepage. (n.d.). Retrieved from http://www.angular.io/

[7] React – A JavaScript library for building user interfaces. (n.d.). Retrieved from https://reactjs.org/

[8] Vue.js. (n.d.). Retrieved from https://vuejs.org/

[9] Get TypeScript. (n.d.). Retrieved from https://www.typescriptlang.org/

[10] Baranovskiy, D. (n.d.). Raphaël-JavaScript Library. Retrieved from http://dmitrybaranovskiy.github.io/raphael/

[11] Jgraph. (2018, December 14). Jgraph/mxgraph. Retrieved from https://github.com/jgraph/mxgraph [12] JointJS: Visualize and interact with diagrams and graphs. (n.d.). Retrieved from https://www.jointjs.com/opensource

[13] Foundation, N. (n.d.). Retrieved from https://nodejs.org/en/

[14] Anderson, J. C., Lehnardt, J., & Slater, N. (2010). *CouchDB: The Definitive Guide: Time to Relax.* « O'Reilly Media, Inc.».

[15] Kristina, C., & Michael, D. (2010). MongoDB: the definitive guide. *O'Reilly Media, 1ed.–2010.*

[16] (n.d.). Retrieved from https://firebase.google.com/

[17] Farkas, Z., Kacsuk, P., & Hajnal, Á. (2016). Enabling workflow-oriented science gateways to access multi-cloud systems. *Journal of Grid Computing, 14*(4), 619-640.

[18] CloudBroker Platform. (n.d.). Retrieved from http://cloudbroker.com/platform/

[19] Marru, S., Gardler, R., Slominski, A., Douma, A., Perera, S., Weerawarana, S., . . . Chinthaka, E. (2011). Apache airavata. *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments - GCE 11.* doi:10.1145/2110486.2110490

[20] Perera, S., Marru, S., & Herath, C. (2008, June). Workflow infrastructure for multi-scale science gateways. In *TeraGrid Conference.*

[21] Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman, C., ... & Laity, A. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming, 13*(3), 219-237.

[22] McLennan, M., Clark, S., Deelman, E., Rynge, M., Vahi, K., McKenna, F., ... & Song, C. (2013). Bringing scientific workflow to the masses via pegasus and hubzero. *parameters, 13,* 14.

[23] McLennan, M., & Kennell, R. (2010). HUBzero: a platform for dissemination and collaboration in computational science and engineering. *Computing in Science & Engineering, 12*(2), 48-53.

[24] Balis, B., Figiela, K., Malawski, M., Pawlik, M., & Bubak, M. (2015, September). A lightweight approach for deployment of scientific workflows in cloud infrastructures. In *International Conference on Parallel Processing and Applied Mathematics* (pp. 281-290). Springer, Cham.

[25] Qasha, R., Cała, J., & Watson, P. (2016, October). A framework for scientific workflow reproducibility in the cloud. In *e-Science (e-Science), 2016 IEEE 12th International Conference on* (pp. 81-90). IEEE.

[26] Qasha, R., Cala, J., & Watson, P. (2016, December). Dynamic deployment of scientific workflows in the cloud using container virtualization. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (pp. 269-276).

Submitted 21.07.2018
Accepted 12.10.2018