"Bring-your-own" Plug-in Management Middleware for Programmable Science Gateways

Komal Bhupendra Vekaria, Prasad Calyam, Roland Oruche, Yuanxun Zhang, Songjie Wang
University of Missouri, Columbia, USA
Email: {*kvhg2, rro2q2, yzd3b*}*@mail.missouri.edu; calyamp@missouri.edu; wangso@missouri.edu*

***Abstract*—There is a growing need for next-generation science gateways to increase the accessibility of data sets and cloud computing resources using latest technologies. Most science gateways today are built for specific purposes with pre-defined workflows, user interfaces, and fixed computing resources. There is a need to modernize them with middleware that can provide 'plug in' support to programmatically increase their extensibility and scalability to meet users' growing needs. In this paper, we propose a novel middleware that can be integrated into science gateways using a "bring-your-own" plug-in management approach. This approach features microservice architectures to decouple applications, and allows users (i.e., administrators, developers, researchers) to customize and incorporate domain-specific components in an existing science gateway. We detail the application programming interfaces in our middleware for creation of end-to-end pipelines with diverse infrastructure, customized processes, detailed monitoring and flexible programmability for a scientific domain. We also demonstrate via a OnTimeRecommend case study on how our "bring-your-own" approach can be seamlessly integrated by a science gateway administrator/developer using a web application.**

***Index Terms*—Science Gateways, Microservices, Intelligent Middleware, Modularity, Application Decoupling**

I. Introduction

Science gateways hide the complexities for scientific domain users to access distributed computing resources, and perform big data management. Through easy-to-use domain application interfaces, they handle user needs for a variety of scientific tasks related to research and education. Science gateways have been developed in many scientific domains, including bioinformatics, neuroscience, physics, chemistry, and material science [1] [2]. Many of today's science gateways help domain science users in execution of workflows, automation for data integration, and analysis/visualization of voluminous data - on distributed high-performance computing resources (HPC) and cloud resources (e.g., Amazon Web Services, CyVerse [3]).

However, most science gateways today are built for specific purposes with pre-defined workflows, user interfaces, and fixed computing resources. Such a state-of-practice makes it difficult for users whose science gateway needs constantly evolve in terms of e.g., data-intensive workflow automation or choice of cloud computing platform. Also, science gateway administrators/developers often are challenged to integrate advanced technologies (e.g., knowledge bases, recommenders, machine learning tools) due to original architecture design limitations. To overcome such issues, there is a need to modernize science gateways with middleware that can provide 'plug in' support to programmatically increase their extensibility and scalability to meet users' growing needs. By thus increasing the modularity of science gateways with such a middleware, diverse user needs can be satisfied at the front-end, and dynamic resource management can be supported at the back-end.

In this paper, we propose a novel middleware that can be integrated into science gateways using a "bring-your-own" plug-in management approach. Our middleware to manage plug-in services is inspired by a "pluganized" management framework developed in [4], which enables plugin of network protocols as extensions to support fast/secure data transmission. Using a plug-in management approach, our middleware features microservice architectures to decouple applications, and allows users (i.e., administrators, developers, researchers) to customize and incorporate domain-specific components in an existing science gateway. We detail the application programming interfaces (APIs) in our middleware for creation of end-to-end pipelines with diverse infrastructure, customized processes, detailed monitoring and flexible programmability for a scientific domain. The APIs provide science gateway administrators/developers with the following benefits:

- *Modularity* "Bring-your-own" plug-in approach leverages microservice architectures that decouples the science gateway application code, and enables addition of new services that function independently but interconnect to each other. Consequently, the microservices code can be reused by processes that have similar execution behavior across multiple science gateways.
- *Extensibility* "Bring-your-own" plug-in management approach allows science gateway administrators/developers to easily extend the middleware with additional application components or plug-ins such as e.g., multi-cloud based workflows with templates, execution pipelines involving machine learning algorithms, and more.
- *Scalability* "Bring-your-own" plug-in management approach allows users to reserve pre-configured and ready-to-use cloud infrastructure resources in order to help them scale their workloads as and when needed. Replacing a microservice for a different scale of resource needs allows for dynamic resource management for changing user workflow needs.
- *Programmability* "Bring-your-own" plug-in management approach helps science gateway administra-

Presented at Gateways 2020, Online, USA, October 12–23, 2020.
https://osf.io/meetings/gateways2020/

tors/developers to program, register and upload various components into their existing setup using a customizable application interface.

Lastly, we demonstrate the benefits of our "bring-your-own" plug-in management approach via an OnTimeRecommend case study. Specifically, we show how our middleware can be seamlessly integrated by a science gateway administrator/developer using a web application.

The remainder of the paper is organized as follows: Section II presents related work. Section III details our "bring-your-own" plug-in management middleware design and implementation. Section IV describes an OnTimeRecommend case study to show our middleware integration within a science gateway. Section V discusses the challenges we addressed in the integration of emerging AI/ML tools in next-generation science gateways. Section VI concludes the paper.

II. Related Work

Designing and developing a successful science gateway takes a significant amount of time, funding and personnel effort. However, science gateways have to continuously evolve to adapt to the changing needs in scientific research/education tasks. Leveraging advanced technologies can help science gateways to address this issue. One such technology involves use of microservices via RESTful APIs. For example, GenApp [5] leverages decoupling of application code from the science gateway to allow researchers to specify only the input and output parameters to run their command line applications via a graphical interface. Agave [6], a science-as-a-service API platform, was built largely with Docker container based microservices to seamlessly integrate API management, capacity scaling, and community contributions to provide platform services, science APIs and support services.

A number of science gateways are looking to use easily reusable and transferable building blocks. Apache Airavata [7] provides a software suite to compose, manage, execute, and monitor large-scale applications and workflows for science gateways. PaaSage [8] supports the design and deployment of multi-cloud applications by optimizing and customizing workflows in science gateways. Agave [6] offers platform-as-a-service for hybrid cloud computing and data management purposes and is being adopted as the API layer by several science gateways, such as CyVerse [3]. Globus Galaxies [9] is a domain-independent, cloud-based science gateway providing a web-based interface for creating, executing, sharing, and reusing workflows composed of arbitrary applications, tools, and scripts. MiCADO [10], a microservice-based application orchestrator middleware, offers scalable Docker container-based microservice deployment by integrating services from federated private and public cloud resource providers. Today's science gateways provide comprehensive user services and convenient workflow automation; however, they largely lack the capability for users to customize programmable plug-ins. The integration of plug-ins require modular programming to extensively maintain and deploy heterogeneous components. Our middleware development is inspired by these leading science gateways and frameworks to allow highly portable and reusable building blocks in next-generation science gateway development, as well as decoupling of system components to allow users to customize their workflows and computing tasks.

III. "Bring-Your-Own" Plug-ins Management Middleware

In this section, we detail design of our "Bring-Your-Own" plug-ins management middleware and its implementation.

A. System Design

Design of "Bring-Your-Own" plug-in management middleware is shown in Figure 1. The plug-ins are integrated with the Application Layer featuring a science gateway such as e.g., CyNeuro [2] with the help of a OnTimeRecommend application that is used by science gateway administrators/developers. In addition, the plug-ins also interface with the Infrastructure Layer through REST APIs offered by resource providers such as e.g., CyVerse or AWS. The Infrastructure Layer can include cloud templates, machine learning tools, artificial intelligence platforms featuring chatbots, recommender modules or domain-specific knowledge bases.

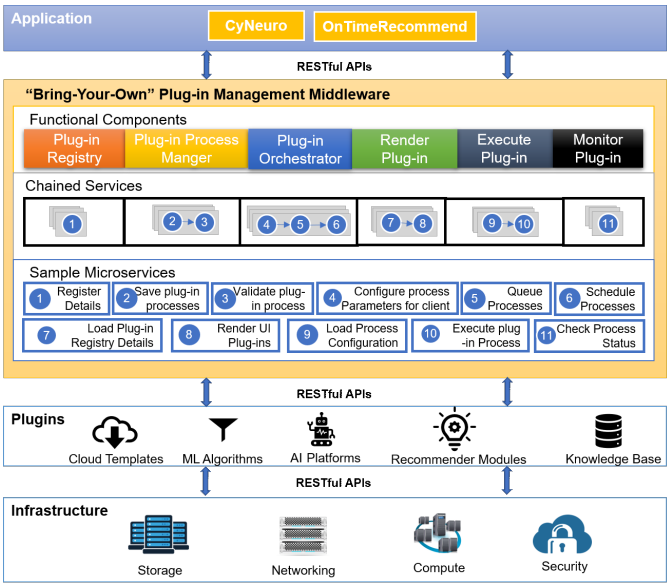

Fig. 1. Multi-layered "Bring-your-own" Plug-ins Management Design.

Application Layer is assumed to be comprised of two categories of user roles: end user researchers/scientists accessing a science gateway such as CyNeuro, and science gateway administrators/developers who use the OnTimeRecommend web application. The OnTimeRecommend web application features an 'Admin interface' to integrate, manage and execute different plug-ins using functional components implemented as microservices. Once integrated, augmented interfaces are provided to end users (researchers, students) when using the domain-specific gateways to access and use the capabilities provided by the plug-ins. As shown in Figure 1, several application-specific or infrastructure-specific microservices can be chained to customize capabilities and allow

information sharing between the different microservices to execute specific functional components.

We implement our "Bring-Your-Own" plug-in management middleware as an end-to-end framework that provides administrators/developers plug-in management capabilities. Herein, we detail the 'Admin interface' functionality with the following plug-in components:

- *Plug-in Registry* is the repository for all clients and plug-ins related data. It also includes the metadata of plug-ins–which is the configuration to execute the processes related to the client's plug-in selection–as well as the science gateway application details. This metadata is formatted using JSON and stored in the database when add or update actions are taken on plug-ins/processes. Our current implementation of this registry uses the Hibernate framework for object-relational mapping over a relational database i.e., a MySQL backend.
- *Plug-in Process Manger* is used to create plug-in processes for a science gateway. Plug-in management and execution can be broken down into multiple processes. We implemented microservices that help users to configure and execute processes for specific plug-ins. This component is also responsible as a client to the Plug-in Registry for persistently storing all process details, metadata information related to each process of plug-ins.
- *Plug-in Orchestrator* abstracts the configuration of the middleware queuing layer for individual clients. Specifically, it configures execution parameters for different processes of plug-ins, and consequently queues all processes of plug-ins. This component is also responsible as a client to the Plug-in Registry for persistently storing all parameter configurations and queuing requests for processes in a queue.
- *Render, Execute and Monitor Plug-in* provides the web-based user interface to the application layer using a client software development kit that we implemented. Execute plug-ins component executes different processes for specific plug-ins based on user requests, and the Monitor plug-ins component checks the execution status of the plug-ins.

B. Implementation Details

We have implemented microservices that follow the standard practices of RESTful API design. All microservices have been developed in the backend using Spring Boot, which is a widely used framework in Java. Spring Boot is pre-configured and pre-sugared with a set of technologies that drastically minimize the manual efforts of configuration compared to conventional frameworks. In addition, we have used Apache Maven, which is a comprehensive build management tool to manage dependencies and versions, compile source code, runs tests, package code into deployment-ready file formats, and deploy a final production code instance using Docker containers.

The microservice architecture involves enabling flexible interactions between multiple services. Each instance of a service exposes a remote REST API at a particular location (host and port), and the number of service instances as well as their locations change dynamically. In this case, a combination of service registry and client-side service discovery allow services to find and communicate with each other without hard-coding the host names and ports. The service registry handles details of services such as their instances and locations. Service instances are registered with the service registry on startup and are de-registered on shutdown. We have implemented microservices for both service registry and discovery client microservices using Spring cloud [11] and Eureka. We have also developed gateway edge service using Spring cloud and Zuul to enable dynamic routing in our middleware.

In addition, we have implemented the OAuth 2.0 security protocol with Spring Boot and Spring Security to provide authentication support to science gateway clients. It enables third-party applications (e.g., GitHub API, Google API) to obtain limited access to web applications. This allows for science gateway administrators/developers to enable access control to plug-in services they want to provision. With the integration of OAuth 2.0, we can validate users by allowing them to sign-on to the web application with necessary authorized permissions.

C. Plug-in Management Middleware Benefits

The implementation of our plug-in management middleware allows developers/administrators to integrate and monitor the use of plug-ins in science gateway applications. Herein, we detail salient benefits of our middleware architecture for administrators/developers:

- It enables administrators/developers to modularize the plug-in services used to develop microservices in science gateway applications. These design benefits can support plug-ins to independently operate by decoupling processes into microservices.
- It supports customizable design and deployment to augment scientific workflows. The architecture uses Docker containers to construct deployment patterns across the distributed resources to optimize the pre-defined workflows commonly used in a domain science community (e.g., neuroscience, bioinformatics).
- It also provides the flexibility for disparate code bases to be integrated through microservices. Administrators/developers typically create microservices using their preferred coding language. Our architecture allows developers to use their preferred coding language for creating customizable science gateways.

IV. OnTimeRecommend Case Study

In our recent prior work, we implemented the OnTimeRecommend [2], which features a variety of recommender modules to help novice/expert users with knowledge discovery through data sources such as e.g., publications, funding records, cloud templates and Jupyter notebooks. OnTimeRecommend also features a Vidura Advisor (i.e., a Chatbot) using Google DialogFlow to provide a guided user interface with step-by-step navigational support. Vidura generates distinct

responses of OnTimeRecommend recommenders for users based on novice/expert user intent to accomplish targeted research and education tasks.

A. "Bring-Your-Own" Plug-in Management Middleware Integration in OnTimeRecommend

Herein, we detail how we customized our "Bring-Your-Own" plug-in management middleware into the OnTimeRecommend system. The integration involves: (i) "Bring-Your-Own-Recommender" customized middleware and Admin interface for OnTimeRecommend to add, manage, execute recommender modules, and (ii) a recommender user interface for providing recommendations on available resources to science gateway end users. Using these two integration thrusts, OnTimeRecommend provides a 'recommender-as-a-service' functionality to the CyNeuro science gateway that is currently being used by researchers and educators in the neuroscience community. We integrate OAuth 2.0 to authenticate and authorize the clients of the CyNeuro science gateway by allowing them to customize their desired plug-in services. The plug-ins in this case study are the recommender modules within the OnTimeRecommend system, which features different recommender modules that can be integrated as shown in Figure 2. The details of the recommender modules are as follows:

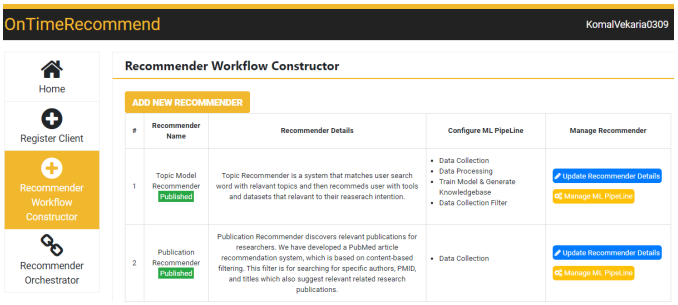

Fig. 2. 'Bring-Your-Own' Plug-in Management Interface to Add, Manage, and Execute Recommender Modules in OnTimeRecommend

- *Domain-specific Topic Recommender* guides users towards successfully identifying topic associations e.g., tools that are popularly used in specific domains such as e.g., neuroscience.
- *Publication Recommender* discovers relevant publications for researchers using key words.
- *Jupyter Notebook Recommender* searches and retrieves relevant Jupyter notebooks when users have a specific training requirement to accomplish a research/education task.
- *Workflow Template Recommender* suggests cloud templates according to researcher's requirement in terms of cost or performance.
- *Scholar Finder* suggests scholars who are experts that are competent to accomplishing a research task based on their publication and funding records.

B. Plug-in Management Middleware Integration Steps

Figure 3 illustrates the steps to configure middleware components to add and execute the different recommender modules. The steps for a science gateway administrator/developer to customize OnTimeRecommend modules for a science gateway are as follows:

- Step-1: register different recommender modules in OnTimeRecommend using the web application interface. This interface uses Plug-in Registry services to register information related to recommender modules for a science gateway.
- Step-2: register scientific plug-in processes such as data collection/processing parameters, and knowledge base, such that all the necessary information is provided to execute relevant recommenders in OnTimeRecommend using the 'Plug-in Workflow Manager'.
- Step-3: add a science gateway (i.e., CyNeuro in this case study) client and link recommender modules with the client to allow users to use recommenders on their science gateway interface.
- Step-4: configure parameters of recommender processes and queue processes of recommender modules for a specific science gateway client.
- Step-5: execute all recommender processes using provided configurations and publish recommender outputs to end users on their science gateway interface.

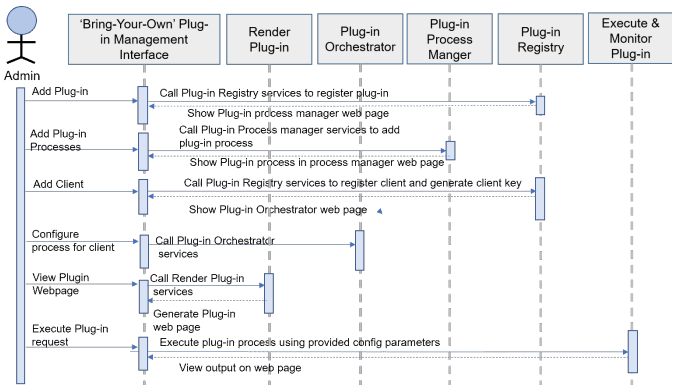

Fig. 3. Sequence Diagram to Add, Manage, Execute Plug-ins customized for a Science Gateway

V. Discussion

Our novel middleware framework for managing plug-in services provides new capabilities for users (e.g., administrators/developers, researcher) in the science gateways community. The core feature of our middleware is in the integration of plug-in management middleware components for AI/ML execution pipelines, which pose new challenges and open questions in next-generation of science gateways. We showed how our plug-in management for application providers could be designed to keep up with the rapid increase of voluminous data, tools, and various resources openly available to scientific communities. The maintenance of the plug-ins also is an

important aspect of our middleware, which involves collecting up-to-date data to train and re-train the models to ensure latest guidance is provided from the plug-in services, especially involving recommender modules in OnTimeRecommend.

Proper facilitation of updated resources via plug-in management also needs to be adapted with respect to human cognition. For instance, it is crucial to develop flexible features that ensure users are able to customize the plug-in management middleware to be relevant across different science domains (e.g., neuroscience, bioinformatics) that have unique workflow requirements. Moreover, our middleware can be augmented with additional plug-ins that support natural language processing as well as context-aware chatbots to handle diversity of user requirements and to ensure update of the knowledge bases on a regular basis. Failure to address these demands in the middleware to manage plug-in services could result in irrelevant information from the recommender responses. Consequently, users may not be willing to adopt the new tools being considered in this work, which provide user guidance to handle big data handling needs in next-generation science gateways.

VI. Conclusion

In this paper, we presented a novel "Bring-You-Own" plug-in management middleware design and implementation to enable science gateways to leverage advanced technologies in a customizable manner to increase their extensiblity and scalability. Through integration of our middleware with an Application Layer and Infrastructure Layer, changing needs of domain scientists can be satisfied. The Application Layer involves a science gateway such as e.g., CyNeuro [2] for neuroscience researchers/educators. The Infrastructure Layer involves REST APIs offered by resource providers such as e.g., CyVerse or AWS. The Infrastructure Layer can include cloud templates, machine learning tools, artificial intelligence platforms featuring chatbots, recommender modules or domain-specific knowledge bases. Through an OnTimeRecommend case study, we showed how a science gateway administrator/developer can configure relevant recommender modules for the science gateway users (i.e., CyNeuro in the case study) to perform knowledge discovery of cloud templates, Jupyter notebooks, publications and domain experts. Our plug-in management approach involving microservices can be generally applied to extend and scale any science gateway through a series of integration steps supported by our middleware via a web application.

As part of future work, we plan to develop additional plug-in support features in our middleware for: (a) integration of chatbots for guided interfaces, and (b) knowledge bases for intelligent data analytics, in other domain science gateways such as bioinformatics and health informatics.

Acknowledgment

This work is supported by the National Science Foundation under awards: OAC-1730655 and OAC-2006816. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] B. Demeler, "Ultrascan: a comprehensive data analysis software package for analytical ultracentrifugation experiments," *Modern analytical ultracentrifugation: techniques and methods*, pp. 210–229, 2005.

[2] S. S. Sivarathri, P. Calyam, Y. Zhang *et al.*, "Chatbot guided domain-science knowledge discovery in a science gateway application," *Proceedings of Gateways*, 2019.

[3] N. Merchant, E. Lyons, S. Goff, M. Vaughn, D. Ware, D. Micklos, and P. Antin, "The iplant collaborative: cyberinfrastructure for enabling data to discovery for the life sciences," *PLoS biology*, vol. 14, no. 1, 2016.

[4] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, "Pluginizing quic," *Proceedings of the ACM SIG on Data Communication*, pp. 59–74, 2019.

[5] A. Savelyev and E. Brookes, "Genapp: Extensible tool for rapid generation of web and native gui applications," *FGCS*, pp. 929–936, 2019.

[6] R. Dooley, S. R. Brandt, and J. Fonner, "The agave platform: An open, science-as-a-service platform for digital science," *Proceedings of PEARC*, pp. 1–8, 2018.

[7] M. Pierce, S. Marru, B. Demeler, R. Singh, and G. Gorbet, "The apache airavata application programming interface: overview and evaluation with the ultrascan science gateway," *Gateway Workshop*, 2014.

[8] A. Rossini, "Cloud application modelling and execution language (camel) and the paasage workflow," *Advances in Service-Oriented and Cloud Computing*, 2015.

[9] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, and I. Foster, "The globus galaxies platform: delivering science gateways as a service," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4344–4360, 2015.

[10] T. Kiss, P. Kacsuk, J. Kovács, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, and G. Terstyanszky, "Micado—microservice-based cloud application-level dynamic orchestrator," *Future Generation Computer Systems*, vol. 94, pp. 937–946, 2019.

[11] "Spring cloud netflix. accessed may 2020," [Online] Available at https://spring.io/projects/spring-cloud-netflix.