The Apache Airavata Application Programming Interface: Overview and Evaluation with the UltraScan Science Gateway

Marlon Pierce
Indiana University
Bloomington, IN
marpierc@iu.edu

Suresh Marru
Indiana University
Bloomington, IN
smarru@iu.edu

Borries Demeler
University of Texas Health Science Center
San Antonio, TX
demeler@biochem.uthscsa.edu

Raminderjeet Singh
Indiana University
Bloomington, IN
ramifnu@iu.edu

Gary Gorbet
University of Texas Health Science Center
San Antonio, TX
gegorbet@gmail.com

ABSTRACT

We present an overview of the Apache Airavata Application Programming Interface (API), describe the design choices and implementation details, and describe how API methods map to the UltraScan Science Gateway use case. The Airavata API is designed to standardize access to Airavata services that provide gateways with scientific application metadata and execution management. The API also represents an important milestone in the development of Science Gateway Platform as a Service (SciGaP), a hosted, multi-tenanted gateway service based on open source Airavata software. The UltraScan gateway is a production XSEDE gateway that has been using Airavata services for over three years through customized interrfaces and represents a stringent test of the API design and implementation.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features – *abstract data types, polymorphism.*

General Terms

Design, Reliability, Human Factors, Standardization

Keywords
Science gateways, application programming interface design, cyberinfrastructure, cloud computing

1. INTRODUCTION
Science gateways are coming of age, with many successful gateways serving hundreds to hundreds of thousands of users. General background on gateways is available from [1]. More recent developments are surveyed in [2] [3] [4]. In general, science gateways provide user-, science-, or application-centric views of cyberinfrastructure that adapt and build upon the resource centric views of cyberinfrastructure and access layers such as UNICORE, Globus, and gLite.

Although some gateways have proven extremely successful, there is room for improvement in the success rate and the types of communities served. We believe important elements to increasing the success rate and sustainability of gateways lie in providing transparently operated services based on open source software that allow gateways to outsource their general requirements for metadata, application and workflow execution management [5]. This permits gateway developers and operators to concentrate resources on serving their target communities.

In developing these general-purpose gateway services, it is essential to define a workable application programming interface (API). The API defines the contract between the gateway and the platform services, delimiting what the service does and how the gateway will need to interact with the service. This is essential if gateway services are to obtain the benefits of scaling. A properly designed API is easily used by the gateway to accomplish its main tasks. Properly used APIs also eliminate custom, per-gateway integration efforts. The latter result in integration or bridging code that is outside the code base and testing framework of the gateway software. Related work on cyberinfrastructure APIs for applications and science gateways is available in [6][7][8].

UltraScan Science Gateway: The Ultrascan Science Gateway [9] provides access to data analysis tools that help users interpret results from analytical ultracentrifugation (AUC) experiments. Analytical ultracentrifugation is a technique that can be used to determine properties such as size, mass, density and anisotropy distributions of macromolecules and nanostructures in solution [10]. As discussed in [11][12], supercomputing and cluster resources are needed to provide the computing power for full data analysis of experiments. Furthermore, new versions of AUC detectors are capable of acquiring data at multiple wavelengths, increasing the data density by several orders of magnitude, and thereby creating a data-parallel computing problem that greatly amplifies the computing requirements of AUC data analysis.

The UltraScan Science Gateway seeks to simplify access to high end resources for the AUC community, who typically are "wet lab" scientists, unfamiliar with high performance computing and related facilities. The current version of UltraScan serves 72 different AUC installations and their associated user communities [http://www.uslims3.uthscsa.edu]. Each of the 72 installations is associated with a separate, dedicated Laboratory Information Management System (LIMS) that manages the data for that instrument. UltraScan users interact with the LIMS systems through a desktop user interface (used for initial data exploration, pre-processing, visualization and meta-data analysis), and through a Web-enabled front end to collaborate among multiple investigators, access comprehensive analysis reports, and to stage high resolution data analysis jobs that require submission to supercomputers or clusters. During 2013, UltraScan enabled 127 users to submit over 88,000 analysis jobs on XSEDE and local campus resources, utilizing over one million XSEDE service units. UltraScan has used Apache Airavata and its predecessor software from the Open Gateway computing environments project since 2010 [13][14].

UltraScan's growing international community presents design and operations challenges for the gateway, as data locality becomes more important. The current LIMS system is hosted at the University of Texas Health Science Center, San Antonio. While the manageability provided by this centralization works well for US-based users performing analysis on US computing resources such as XSEDE, it introduces latency and usability issues for international users using international resources. This is an important problem that UltraScan must solve as it expands its collaborations with institutions in Europe, India and other countries.

Apache Airavata and Science Gateway Platform as a Service: Apache Airavata [15] is open source, open community software for managing the metadata and execution of single applications and workflows on clusters, supercomputers, and computational clouds. A summary description of Airavata's internal components and their capabilities is given in [16]; in brief, Airavata provides separate components for managing metadata (the Registry), for scheduling and job management (the Orchestrator), for workflow execution (the Workflow Interpreter), for interacting with remote resources to execute specific operations (GFAC), a for managing security credentials (the Credential Store [17]), and for messaging (the Messenger). As open community software, a goal for Airavata is for these major system components to be pluggable, facilitating distributed computing research and rapid prototyping, such as new algorithms for scheduling and alternative approaches for data management, by a wide range of stakeholders.

Besides facilitating distributed computing research, we use Airavata to power many science gateway and workflow collaborations. We thus need to think carefully about the operational requirements needed to sustainably and scalably support these collaborations. Simultaneously, we need to reduce barriers to the adoption of the gateway services by providing easily adapted code prototypes that permit rapid integration into workflows. These considerations are motivations behind the Science Gateways Platform as a Service (SciGaP) project, which has the goal of making a multi-tenanted hosted gateway service that can serve multiple gateways simultaneously that is based on Apache Airavata and selected third party services. This consolidation of services will enable us to scale our support for multiple gateways (every gateway uses the same service). Defining an API for Airavata is an important part of this overall strategy, as it explicitly defines the interactions between gateways and Airavata, removing the temptation for providing ad-hoc integrations. The latter must be avoided since they lead to services that are difficult to maintain and only fully understood by the persons responsible for the integration.

2. AIRAVATA API OVERVIEW

The Apache Airavata API has been designed to address the problems described in the previous section. In general, our goals for the API are that a) it should be general enough to support both workflows and single job submissions, b) that it should be easily integrated with gateways developed using a number of different programming languages, and c) it should expose all of Airavata's capabilities through a single entry point.

A full discussion of the API is beyond the scope of this extended abstract, but we provide a summary of the elements and implementation. Although REST-based APIs are popular, we had reservations about using them in Airavata. Airavata is built upon earlier projects that made extensive use of XML and Web Services, providing us with rich data models for describing applications and computing resources within the Airavata Registry. Although we are not retaining these implementations, we do want to make it easy to keep the depth and strong typing of our data models in the API. For this reason, we have chosen Apache Thrift [18] as the interface definition language. Thrift has the additional advantages of providing numerous language bindings and explicitly defined error methods for fine-grained client-side exception handling. A bonus is that well crafted thoughtfully designed Thrift based API's can facilitate backward and forward compatibility. Thrift is network transport- and language-neutral, and its vital open source community has been very responsive to our questions. Further, it is used in production by very high-profile clients, including Facebook and Evernote.

The Airavata API is described as Thrift data structures and service methods and language specific software development kits bind these definitions to client stubs and server side skeletons, for handling network transport, and for marshalling and unmarshalling objects. We visually summarize the API's major components in Figure 1. A complete definition of the API is available from the Airavata source code [19]; [20] provides navigable HTML documentation.

The *application catalog API* facilitates gateway administrators and advanced users to register and manage computational resources, describe the application and workflow descriptions and register application deployments on various computational resources. This information is used by Airavata during application and workflow task executions. The *experiment catalog API* enables users to browse, query for previously executed experiments. These include the input data and configurations, generated data sets, execution logs. The experiment catalog also facilities users to organize executions into projects. In the future we plan to enable user and group level sharing and publishing of results to allow read access to all users. The *execution management API* facilitates the creation and launch of experiments. An experiment is conceptualized by a gateway action typically associated with execution of an application or a workflow. This grouping also allows managing of executing experiments like interacting (resume, pause, cancel, clone) experiments. The *user management API* proxies to gateway user store (when available like the Ultrascan usecase) or provides an implementation using third party identity store like the WSO2 Identity Server.

The API methods summarized above interact with rich data models, also defined with Thrift. Application catalog data models (appCatalogModels.thrift) include computeResourceModel, applicationInterfaceModel, applicationDeploymentModel, and gatewayResourceProfileModel. These are used, respectively, to describe computing resources that will be used by the gateway, applications that the gateway will run, details about how the application is deployed on a specific resource, and gateway-specific metadata about gateway preferences (such as preferred allocations and workspace directories on a specific machine). These data models are directly implemented in the Airavata Registry component; the API Server provides the interface for gateways to access and modify the data. Write, update, and delete operations associated with these data models in the API are associated with gateway operators rather than regular gateway users.

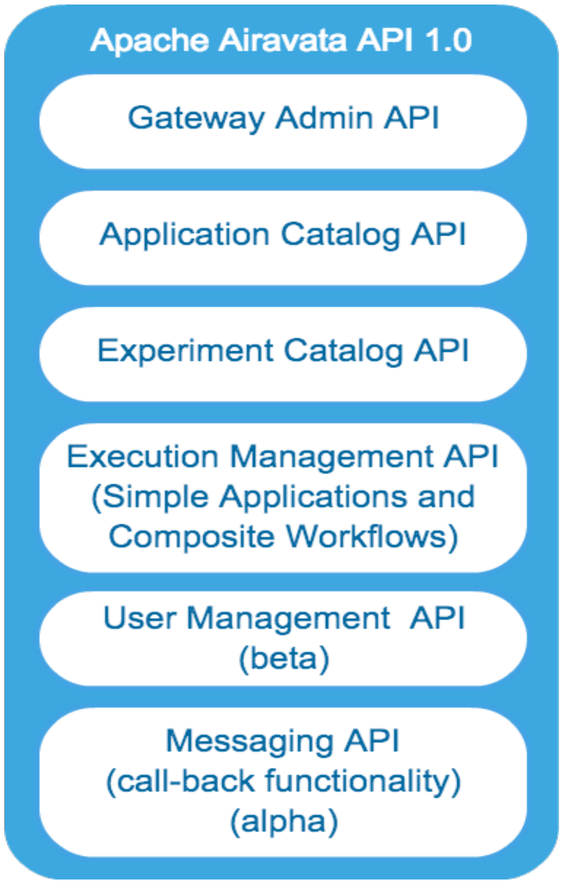
Apache Airavata API 1.0
Gateway Admin API
Application Catalog API
Experiment Catalog API
Execution Management API (Simple Applications and Composite Workflows)
User Management API (beta)
Messaging API (call-back functionality) (alpha)

Figure 1: Apache Airavata API functional grouping

Airavata's workspace data model is the second major category and includes per-user information. We divide information into projects, which in turn contain experiments. The experiment (Figure 2) is the most complicated data model and describes everything associated with a specific job or workflow execution. API methods associated with these data models include Registry database accessing and searching methods as well as methods for launching and monitoring experiments, which interact with Airavata's Orchestrator and GFAC components.

3. IMPLEMENTATION AND EVALUATION WITH ULTRASCAN

UltraScan's Web components are developing in PHP, so we implement UltraScan clients to Airavata using Thrift's PHP client library bindings; we use Java library bindings on the server side to implement Airavata's API Server. Communications between client and server go over TCP/IP connections. We extend Thrift's generic service skeleton so that the various calls in the API implement appropriate calls to different parts of Airavata, typically the Registry and Orchestrator. This API Server is a first class component of Airavata. This allows us to change the wiring of various Airavata internal components without exposing these changes directly to the remote client. More details on this are in [16].

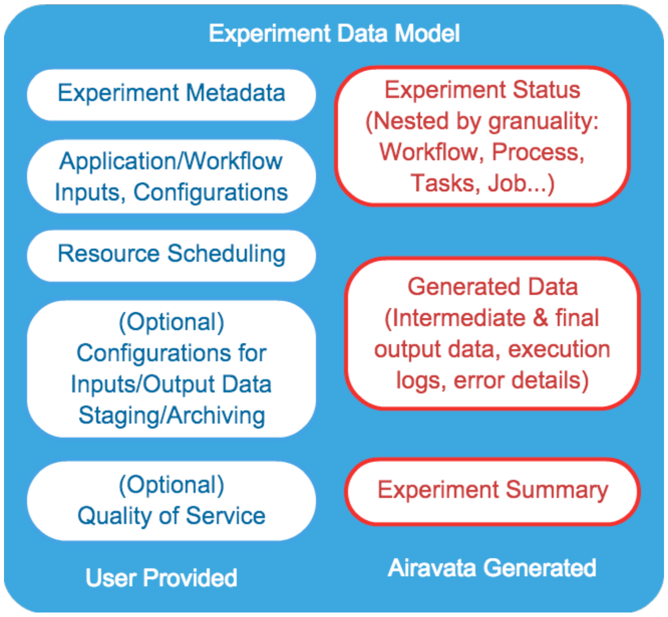
Experiment Data Model
Experiment Metadata
Application/Workflow Inputs, Configurations
Resource Scheduling
(Optional) Configurations for Inputs/Output Data Staging/Archiving
(Optional) Quality of Service
User Provided
Experiment Status (Nested by granuality: Workflow, Process, Tasks, Job...)
Generated Data (Intermediate & final output data, execution logs, error details)
Experiment Summary
Airavata Generated

Figure 2: Airavata's Experiment data model

Mapping the API to UltraScan requirements is straightforward but still requires some thought. On the one hand, UltraScan in its current form only has one application, so Airavata's extensive data models for describing applications are not fully exploited here. Also, UltraScan does not have the notion of "projects" for individual users, only "experiments" in terms of the Airavata data model. On the other hand, UltraScan can be considered a multi-tenanted umbrella gateway, with each of the 72 installations acting as a tenant. Furthermore, UltraScan is in the process of setting up branch locations to better serve the data locality and resource usage requirements of collaborators at Juelich Supercomputing Center, with additional collaborations with the Indian Institute of Science (Bangalore) being initiated. In our current implementation, we are mapping each of the individual LIMS instances to separate Airavata projects, with experiments run by users of each instrument associated with their LIMS project. Our plan for the UltraScan branch locations, when they become available, is to treat these as separate gateways from the API point of view, since they will correspond to different instances of UltraScan services and data management systems.

Actual experiment creation and job submission for an individual submission is straightforward using the PHP client bindings. First, we create an Experiment instance associated with the appropriate Project in Airavata through an API call. The Experiment object is populated in the PHP client with selections provided by the gateway: target host, number of processors, processors per node, wall time, and data sets to be used. The gateway submits the experiment with a separate call through the airavataClient object, specifying the experiment's ID. Airavata does initial validation of the launchExperiment() request to make sure it is complete and has correct entries, and then returns. Sample code is shown in Figure 3. $experiment is a local binding to the Thrift-defined data model described in the previous section, and $airavataclient is an autogenerated stub that provides access to the API methods. The airavataToken is a security token used to map the gateway to the appropriate credential in Airavata's credential store. Airavata provides multiple views of state as well, which are illustrated with the final two lines of Figure 3. Experiment status refers to the whole experiment, including output staging and possible custom output checking. Job status refers specifically to the status of the submitted job on the queueing system.

```
$experiment = new Experiment();
//Omitted code: populate $experiment
$expId = $airavataclient -> createExperiment($experiment);
$airavataclient->launchExperiment($expId, 'airavataToken');
$experimentStatus = $experiment->experimentStatus;
$jobStatus = $airavataclient->getJobStatuses($expId);
```

Figure 3: Code sample summary for creating and launching an experiment.

Actual submission is performed asynchronously by Airavata's Orchestrator, which passes the experiment to an available instance of GFAC. During this process, experiments pass from "created" to "launched" to "executing" states within the Experiment's state model. We have made the design choice to return immediately rather than blocking and waiting for full submission to the selected resource since this can take several seconds or even significantly longer if large file uploads must be staged.

As indicated by the comment in Figure 3, the main work in integrating the API with a gateway to handle a user's job submission reequest is to create the appropriate experiment object. This information is specific to a given execution and is provided directly or indirectly by user choices, but these choices are guided by the entries in the gateway's Application Catalog. Figure 4 provides code samples. The comments that precede the line show simplified versions of the inputs. The application module information is used to specify a specific version of the application. The application interface methods specify the expected input and output types and required modules. The application deployment descriptor provides specific information about running UltraScan on Stampede. Compute registration allows the gateway operator to describe how to interact with a specific resource (Stampede, in this case). Finally, the gateway profile includes UltraScan preferences, such as preferred allocations and working directories to use on a given resource.

```
//Simplified arguments: "ultrascan", "1.0", "ultrascan
application"
ultrascanModuleId = airavataClient.registerApplicationModule();

//Simplified arguments: "ultrascan", "ultrascan application",
// appModules, applicationInputs, applicationOutputs
ultrascanAppId=airavataClient.registerApplicationInterface();

//Simplified arguments: ultrascanModuleId, stampedeResourceId,
// "$WORK/us_mpi_analysis",
//ApplicationParallelismType.MPI, "ultrascan application"
ultascanStamplede=airavataClient.registerApplicationDeployment
();

//Simplified arguments: "stampede.tacc.xsede.org",
//"TACC Stampede Cluster", ResourceJobManagerType.SLURM,
// //"push", "/usr/bin", SecurityProtocol.GSI, 2222,
"/usr/local/bin/ibrun"
computeResourceId = airavataClient.registerComputeResource()
```

Figure 4: Code sample summary for registering the UltraScan application and computing resources.

4. CONCLUSIONS AND FUTURE WORK

Several security aspects of the API are still under development. First, access to API methods needs to be restricted; not everyone, for example, should have write access to an application description. Currently, access restrictions must be implemented and enforced by the gateway. On public Airavata-based SciGaP deployments, we use firewall rules to restrict access to the API Server to known gateways, and we also run the Application Catalog services on a separate port from the Workspace services. Our future approach, which was prototyped in a Google Summer of Code project [21], is to adopt a role-based mechanism. The second security consideration is directly related to UltraScan's potential future directions for using its desktop GUI rather than its Web front end is the primary user interface. It is possible to directly incorporate job submission into the desktop, but this will require the distribution of Airavata client libraries that can be directly used and modified by any user, not just the trusted gateway administrator. We are currently in consultation with the Center for Trustworthy Scientific Cyberinfrastructure on the design of this capability.

Airavata has long employed the WS Messenger component for messaging needs. Recently the community has been exploring alternative third party messaging solutions including Apache Kafka and RabbitMQ. A proof of concept implemented exists in a non-release branch. As part of a stable 1.0 release, we plan to provide a messaging API to address the call-back functionality and enable fine-grained monitoring from gateway user interfaces including mobile devices.

5. ACKNOWLEDGMENTS

This was work supported by the National Science Foundation awards OCI-1032742, ACI-1339774, and ACI-1339649.

6. REFERENCES

1. Wilkins-Diehr, Nancy. "Special issue: science gateways—common community interfaces to grid resources." Concurrency and Computation: Practice and Experience 19, no. 6 (2007): 743-749.
2. Wilkins-Diehr, Nancy, and Amit Majumdar. "XSEDE13 Special Issue Conference Publications." Concurrency and Computation: Practice and Experience (2014).
3. Marru, Suresh, Rion Dooley, Nancy Wilkins-Diehr, Marlon Pierce, Mark Miller, Sudhakar Pamidighantam, and Julie Wernert. "Authoring a Science Gateway Cookbook." In Cluster Computing (CLUSTER), 2013 IEEE International Conference on, pp. 1-3. IEEE, 2013.
4. 6th International Workshop on Science Gateways, IWSG 2014, June 3-5 2014, Dublin, IE; Proceedings in preparation. https://sites.google.com/a/my.westminster.ac.uk/iwsg2014/
5. Pierce, Marlon; Marru, Suresh; Demeler, Borries; Majumdar, Amitava; Miller, Mark (2013): Science Gateway Operational Sustainability: Adopting a Platform-as-a-Service Approach. figshare. http://dx.doi.org/10.6084/m9.figshare.790760Retrieved 01:29, Aug 28, 2014 (GMT).
6. Goodale, Tom, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Gregor Von Laszewski, Craig Lee, Andre Merzky, Hrabri Rajic, and John Shalf. "SAGA: A Simple API for Grid Applications. High-level application programming on the Grid." Computational Methods in Science and Technology 12, no. 1 (2006): 7-20.

7. Dooley, Rion, and Joe Stubbs. "Dynamically Provisioning Portable Gateway Infrastructure Using Docker and Agave." In Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment, p. 55. ACM, 2014.

8. Kacsuk, Peter, Zoltan Farkas, Miklos Kozlovszky, Gabor Hermann, Akos Balasko, Krisztian Karoczkai, and Istvan Marton. "WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities." Journal of Grid Computing 10, no. 4 (2012): 601-630.

9. Demeler, B. (2009) High-Resolution Modeling of Hydrodynamic Experiments with UltraScan. https://www.xsede.org/gateways-listing?p_p_id=sciencegateways_WAR_sciencegatewaysportlet&p_p_lifecycle=1&p_p_state=normal&p_p_mode=view&p_p_col_id=column-2&p_p_col_count=1&_sciencegateways_WAR_sciencegatewaysportlet_actionMethod=LIST&_sciencegateways_WAR_sciencegatewaysportlet_id=15

10. Demeler, Borries, Tich Lam Nguyen, Gary E. Gorbet, Virgil Schirf, Emre H. Brookes, Paul Mulvaney, Ala'A. O. El-Ballouli et al. "Characterization of Size, Anisotropy, and Density Heterogeneity of Nanoparticles by Sedimentation Velocity." Analytical chemistry (2014).

11. Demeler, Borries, Emre Brookes, and Luitgard Nagel-Steger. "Analysis of heterogeneity in molecular weight and shape by analytical ultracentrifugation using parallel distributed computing." Methods in enzymology 454 (2009): 87-113.

12. Brookes, Emre, and Borries Demeler. "Parallel computational techniques for the analysis of sedimentation velocity experiments in UltraScan." Colloid and Polymer Science 286, no. 2 (2008): 139-148.

13. Pierce, Marlon, Suresh Marru, Raminder Singh, Archit Kulshrestha, and Karthik Muthuraman. "Open grid computing environments: advanced gateway support activities." In Proceedings of the 2010 TeraGrid Conference, p. 16. ACM, 2010.

14. Demeler, Borries, Raminderjeet Singh, Marlon Pierce, Emre H. Brookes, Suresh Marru, and Bruce Dubbs. "UltraScan gateway enhancements: in collaboration with TeraGrid advanced user support." In Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery, p. 34. ACM, 2011.

15. Marru, Suresh, Lahiru Gunathilake, Chathura Herath, Patanachai Tangchaisin, Marlon Pierce, Chris Mattmann, Raminder Singh et al. "Apache airavata: a framework for distributed applications and computational workflows." InProceedings of the 2011 ACM workshop on Gateway computing environments, pp. 21-28. ACM, 2011.

16. Pierce, M, Suresh Marru, Lahiru Gunathilake, Raminderjeet Singh, Don Kushan Wijeratne, Chathuri Wimalasena and Chathura HerathApache Airavata: Design and Directions of a Science Gateway Framework" in Proceedings of the International Workshop on Science Gateways, Dublin, IE, June 3-5, 2014.

17. Kanewala, Thejaka Amila, Suresh Marru, Jim Basney, and Marlon Pierce. "A Credential Store for Multi-Tenant Science Gateways." In Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, pp. 445-454. IEEE, 2014.

18. Slee, Mark, Aditya Agarwal, and Marc Kwiatkowski. "Thrift: Scalable cross-language services implementation." Facebook White Paper 5 (2007).

19. Apache Airavata Thrift API definitions: https://github.com/apache/airavata/tree/master/airavata-api/thrift-interface-descriptions

20. Apache Airavata API Documentation: http://airavata.apache.org/documentation/api-docs/

21. Supun Chathuranga Nakandala, "Add Security capabilities to Airavata Thrift services and clients", Google Summer of Code Project, 2014, http://www.google-melange.com/gsoc/proposal/public/google/gsoc2014/scnakandala/5727390428823552