# GenApp Module Execution and Airavata Integration

Emre H Brookes[1], Nadeem Anjum[2], Joseph E. Curtis[3], Suresh Marru[4], Raminder Singh[4], Marlon Pierce[4]
[1]Corresopnding Author / University of Texas Health Science Center, Department of Biochemistry, San Antonio, Texas
[2] Department of Computer Science and Engineering, IIT Kharagpur, Kharagpur 721302, India
[3] NIST Center for Neutron Research, National Institute of Standards and Technology, Gaithersburg, Maryland 20899
[4] Pervasive Technology Institute, Indiana University, Bloomington, Indiana

## ABSTRACT
A new framework (GenApp) for rapid generation of scientific applications running on a variety of systems including science gateways has recently been developed. This framework builds a user interface for a variety of target environments on a collection of executable modules. The method for execution of the modules is unrestricted by the framework. Initial implementation supports direct execution, and not queue managed submission, on a user's workstation, a web server, or a compute resource accessible from the web server. After a successful workshop, it was discovered that long running jobs would sometimes fail, due to the loss of a TCP connection. This precipitated an improvement to the execution method with the bonus of easily allowing multiple web clients to attach to the running job. Finally, to support a diversity of queue managed compute resources, a Google Summer of Code project was completed to integrate the Apache Airavata middleware as an additional execution model within the GenApp framework.

## Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques - *Computer-aided software engineering (CASE).*

## General Terms
Design, Human Factors, Languages.

## Keywords
CASE tools, science gateway, middleware.

## 1. INTRODUCTION
The GenApp framework [1] is a product of a joint United Kingdom / Engineering and Physical Sciences Research Council and United States / National Science Foundation grant entitled: "CCP-SAS – Collaborative Computational Project for advanced analyzes of structural data in chemical biology and soft condensed matter" and is an SI2-CHE cyberinfrastructure project addressing Grand Challenges in the Chemical Sciences. The CCP's initial software elements are primarily small-angle scattering (SAS) software simulation and analysis tools developed by multiple independent laboratories. Small-angle scattering can be performed using individual lab x-ray sources but is more frequently performed at high energy synchrotrons or neutron sources. A collimated beam of x-rays or neutrons with a fixed wavelength are scattered by the sample and is recorded on a two dimensional detector. For solution studies, where particles in solution are typically randomly oriented, the 2d data is radially integrated to produce a 1d scattering curve that contains structural information. Some computations can be quite trivial such as producing various transformed plot views and others can be computationally expensive such as rigid body modeling and expansion of conformational space utilizing various molecular dynamic and Monte Carlo methods and their subsequent scattering simulation and screening. The grant includes aims of developing new and enhanced SAS analysis methods as well as the development and implementation of the cyberinfrastructure bringing together three preexisting software packages which analyze SAS data [2,3,4,5]. Considerations during the design phase of GenApp were based upon observations and discussions with developers of existing independently produced scientific software. Common issues include ease of deployment in an ever-evolving software landscape, support for legacy codes and the fact that science research groups can not typically afford a dedicated software team. The design employed was to divorce the computational modules from the user interface, define their inputs and outputs on an easily extensible set of data types, define applications as seen by the user on a set of pre-existing and user supplied modules, define an easily extensible set of user interfaces and execution models, so called "target languages" and subsequently generate working instances (see figure 1). For further details about GenApp see [1].

Apache Airavata [6] is open source, open community distributed system software framework for supporting the metadata and application execution requirements of both science gateways and scientific workflows. Airavata consists of the following internal components. The Registry is used to store, access, and manage descriptions of applications, computing resources, and computational experiments. The Orchestrator is used to schedule executions of both single applications and workflows. GFAC is the component that interactions with external resources and services needed to execute a specific task. The Workflow Interpreter manages steps in experiments that consist of multiple tasks. The Messenger provides publish-subscribe messaging for both internal components and external consumers. Airavata has an Apache Thrift-defined API, which it exposes through the API Service component. These components and their interactions are more fully described in [7]. Airavata is also serving as one of the

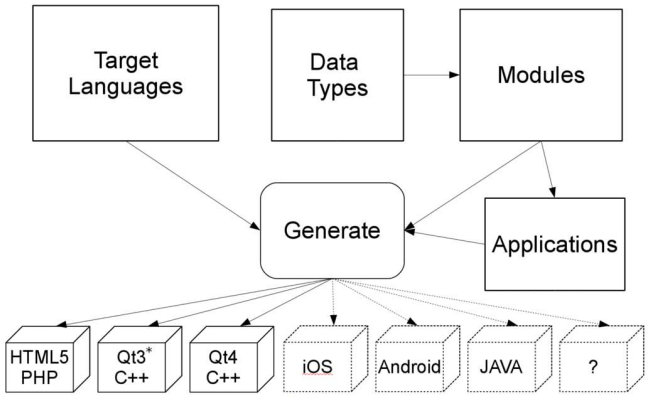


**Figure 1: Generating application instances. The generator reads application definitions, module definitions and chosen target language information to assemble the application instances.**

*Certain commercial equipment, instruments, materials, suppliers, or software are identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

2152-1093/14 $31.00 © 2014 IEEE
DOI 10.1109/GCE.2014.12

bases of a hosted science gateway platform as a service (SciGaP), currently under development. Airavata has supported the execution management of the UltraScan and UltraScan-SOMO gateways, which are precursors of the current work.

## 2. EXECUTION

Prior to this work, execution in GenApp was developed for direct execution on either the client computer (for GUI based target languages) or on a web server (for web based target languages) or via ssh from the web server. To provide support for queuing compute resources, Apache Airavata was chosen to act as middleware. In this section we will describe the execution model of GenApp and how it was modified to integrate with Airavata.

### 2.1 GenApp

User supplied executable modules are wrapped with JSON [8] definitions of their input and output. Typically, a driver script is written to wrap execution modules so that no change to the underlying scientific execution module is required. The driver script for each module must accept standard input in JSON and provide standard output as JSON. A collection of defined modules is placed in a JSON file describing the application. Executing the GenApp tool generates complete application code for an extensible set of "target languages". The primary foci of this current work are the Qt/C++ GUI and HTML5/PHP target languages.

The execution models for the executable modules, prior to Apache Airavata integration, proceed as follows and as shown in Figure 2: The Qt/C++ GUI target language generates a GUI application where each module is executed directly on the user's workstation. The HTML5/PHP target language executes the module through an AJAX call to a PHP module that directs execution of the module.

As part of the CCP-SAS project, a dedicated compute resource was installed at University of Tennessee Knoxville to host HTML5/PHP targets for SAS software, initially the SASSIE software [3]. The server is a Dell cluster running Rocks [9] with two 64-core compute nodes, a eight NVIDIA K20m GPU enclosure, and a 12-core head node. The head node is running a virtual machine hosting the web-server for the HTML5/PHP server. To utilize the compute nodes for execution, we extended the job submission mechanism to support direct ssh execution of modules. The globally available resources are defined as name/value pairs in a JSON application configuration object where global default resource is also defined (see Text 1). Each individual module's definition file can also provide an overriding module specific resource or resources. The changes to the GenApp HTML5/PHP target language module execution required an appropriate prefix to the executed command based upon the called module's determined target resource. This basic resource targeting was functional for our first user's workshop given at the American Conference of Neutron Scattering in June 2014 [10]. Of course, such a system has limited resources and the long-term plan was to support queued resources by integrating with Airavata middleware.

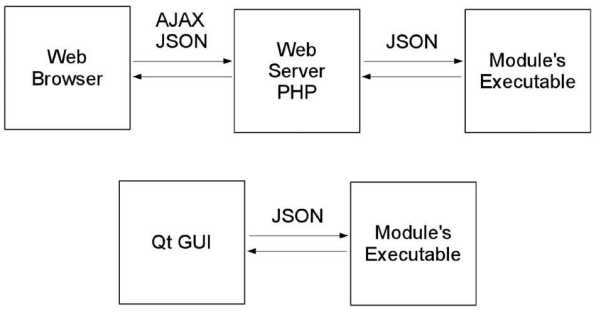


**Figure 2: Execution models for target languages HTML5/PHP (top) and Qt/C++ (bottom).**

The workshop was successful, but subsequently an issue came to light with respect to long running jobs. The HTML5's AJAX call to PHP was waiting on completion of the job to return the JSON results. This required the TCP connection to remain open. It did not appear during our testing with directly connected web clients, nor for general users running jobs of less than approximately one hour, but users reported failures of long jobs that appeared to be the result of a dropped TCP connection during the AJAX call. This precipitated a redesign of the HTML5/PHP execution method. In the updated execution method, the HTML5 initiates an AJAX call, the PHP starts the module execution under a monitor daemon and returns immediately with a simple acknowledgment JSON object. Subsequently, the monitor daemon waits on the job completion and messages the HTML5 client via a pub-sub websocket (see [1] for messaging details). Once messaged, the HTML5 client produces another AJAX call to retrieve the results. Additionally, the HTML5 client has a watchdog slow (currently 16 second) poll as a fallback for the case of a lost message. The updated execution model is shown in Figure 3.

```
{
   ,"resources" : {
                    "local"     : ""
                   ,"compute0" : "ssh compute-0-0"
                   ,"compute1" : "ssh compute-0-1"
                   ,"airavata" : [
                                    "stampede"
                                   ,"gordon"
                                   ,"trestles"
                                   ,"alamo"
                                  ]
                  }
  ,"resourcedefault" : "local"
}
```

**Text 1: JSON describing the available resources for a HTML5/PHP target language instance. "resource", "local" executes directly on the web server, "compute0" and "compute1" utilize ssh to run directly on the attached compute nodes. "airavata" allows submissions to a vector of named targets.**

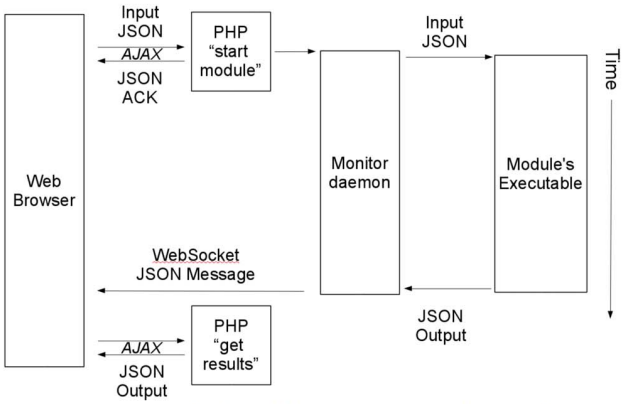


**Figure 3: Two step HTML5/PHP execution. The client web browser makes a second AJAX call to retrieve the results once messaged that the module's execution has completed. Additionally, there is a 16 second interval poll running on the web browser checks for results in the case the WebSocket message is for some reason not received. The web server components coordinate job status via a database (not shown).**

An additional benefit of the updated enhanced execution model is that previously executed module results are easily retrieved and multiple web clients can attach to an running module. These features are both implemented in the current version of GenApp. A minor downside is that additional storage is required to hold the results and should be managed and eventually purged by the system administrators.

## 2.2 Airavata Integration

Integration with Apache Airavata middleware provides GenApp an additional execution model. This execution model supports queuing compute resources. The integration was completed as a Google Summer of Code 2014 Project [11], providing GenApp the capability to execute long-running, non-interactive jobs on distributed computing resources, including local clusters, supercomputers, national grids, academic and commercial clouds. The integration has been achieved for both the HTML5/PHP and C++/Qt target languages. The overview of the integration can be seen in Figure 4.

Airavata requires applications to be registered in a catalog before they can be executed. Using a simple utility, Register Applications, all GenApp executable modules can now be registered with the Airavata Server automatically. This registration step is a one-time task and can be achieved with a single command to execute the registration utility. Once the registration utility has been executed, job submissions can be redirected to Airavata.

GenApp executes short jobs locally and only uses Airavata for long-running jobs, a configuration option has been provided based on a variable setting in the input JSON file to enable or disable job submission to Airavata. There is a global default for resources defined i.e. local, airavata or ssh, along with a module specific override. This provides one the option to execute the required modules on the required resource for maximum resource optimization.

Jobs are submitted to Airavata through Apache Thrift [12] based Airavata clients written in PHP and C++ which make calls to the Airavata API. The execution steps include creating an Airavata project, creating an *experiment* (Airavata's terminology for a specific instance of a module execution) within the project, launching the *experiment*, checking its status and retrieving the output(s). The *experiment* creation step includes specifying the relevant GenApp module, which had been registered by the

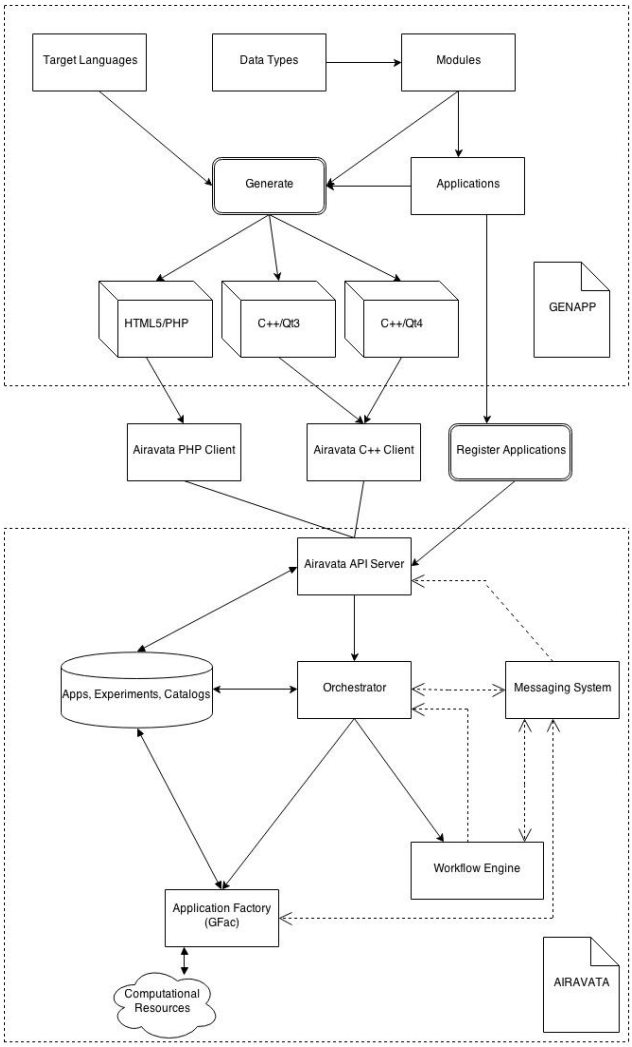


**Figure 4: Overview of GenApp-Airavata Integration. GenApp (top) produces applications (middle) that call the appropriate Airavata (bottom) client as part of their execution of modules. The Airavata API provides manages submission to a diverse set of computational resources.**

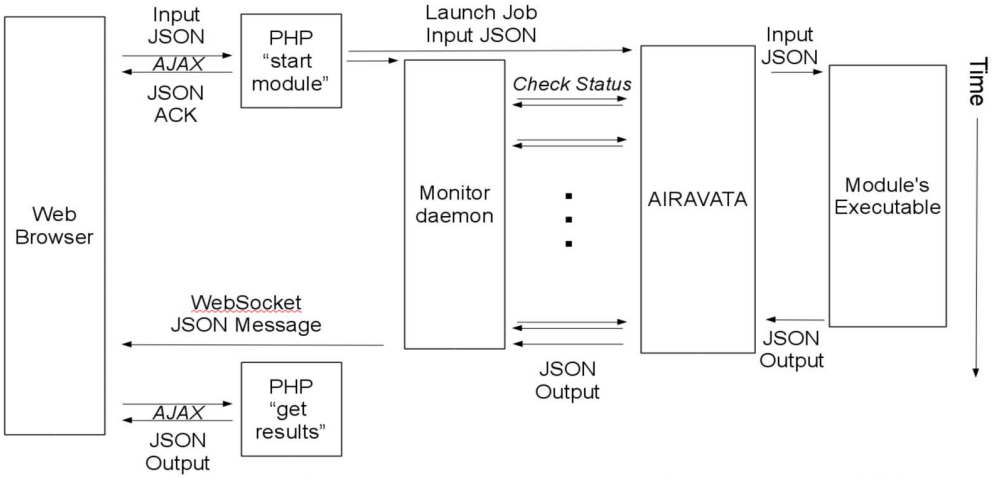


**Figure 5: Two-step Airavata managed HTML5/PHP execution. The module's execution is launched on the first AJAX call from the client and monitored via a "Check Status" poll. Airavata manages the submission of the module's execution on remote compute resources.**

"Register Applications" utility, as the executable and the user inputs retrieved in JSON format as the input. After launching the *experiment*, the status is checked for completion. The *experiment* ends with either "COMPLETED" state or "FAILED" state. On successful completion, the output(s) are retrieved and displayed to the user. In case the *experiment* fails, the appropriate error message is retrieved and displayed to the user. Extensive error handling mechanisms have been implemented to notify the user in case of any exceptions in any step of the execution. The overview of the two-step HTML5/PHP submission mechanism with Airavata managed execution can be seen in Figure 5.

## 3. CONCLUSIONS

The GenApp framework produces user-interfaces for JSON wrapped modules generating GUI and web based applications. The initial community consists of chemical and chemical biology researchers involved in small-angle scattering studies. The framework could easily extend to other scientific disciplines. GenApp's web based execution has been improved with better handling of AJAX calls and the ability to have multiple instances attach to running or view results from previously run jobs. The integration of Airavata provides access to a diverse set of computational resources. In addition to the intellectual contributions, the work described is a good demonstration of bringing computational science and open source software experience to next generation students.

## 4. RESOURCES

The software is currently stored on a subversion integrated Trac Wiki (http://trac.edgewall.org) hosted on an Indiana University Quarry node http://gw105.iu.xsede.org:8000/genapp. A separate virtual machine containing multiple HTML5 application instances is hosted on another Quarry node. A 128 core, 256 GB ram, 8 Tesla K20m GPU cluster is installed at University of Tennessee Knoxville dedicated to computations running under this tool. The Alamo cluster at the University of Texas Health Science Center in San Antonio will also make cycles available callable via Airavata. When usage demands, we will submit an XSEDE proposal for additional cycles to support the Science Gateways developed utilizing this tool.

## 5. ACKNOWLEDGEMENTS

This work was supported by NSF grant 1265817 to E. Brookes and Google Summer of Code funding to N. Anjum. This work benefited from CCP-SAS software developed through a joint EPSRC (EP/K039121/1) and NSF (CHE-1265821) grant. J. Curtis acknowledges support from NIST.

## 6. REFERENCES

[1] Brookes, E.H. 2014. *An Open Extensible Multi-Target Application Generation Tool for Simple Rapid Deployment of Multi-Scale Scientific Codes.* XSEDE '14[1]. ACM DOI=10.1145/2616498.2616560

[2] Perkins, S. http://www.ucl.ac.uk/smb/perkins

[3] Curtis, J. E, Raghunandan, S., Nanda, H., and S. Krueger. *SASSIE: A program to study intrinsically disordered biological molecules and macromolecular ensembles using experimental restraints.* Comp. Phys. Comm. 183, 382-389 (2012) & http://www.smallangles.net/sassie

[4] Brookes, E.H. US-SOMO http://somo.uthscsa.edu

[5] Brookes, E.H., Singh, R., Pierce M., Marru, S., Demeler, S., and Rocco, M. 2012. *Ultrascan solution modeler: integrated hydrodynamic parameter and small angle scattering computation and fitting tools.* XSEDE '12. ACM DOI=10.1145/2335755.2335839

[6] Marru, S., Gunathilake, L., et al. 2011. *Apache airavata: a framework for distributed applications and computational workflows.* Proc. Workshop Gateway computing environments. ACM

[7] 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.

[8] Standard ECMA-404. 2013 *The JSON data interchange format.* Geneva

[9] Philip M. Papadopoulos, Mason J. Katz, and Greg Bruno. 2001. NPACI Rocks Clusters: Tools for Easily Deploying and Maintaining Manageable High-Performance Linux Clusters. In *Proceedings of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Yannis Cotronis and Jack Dongarra (Eds.). Springer-Verlag, London, UK, 10-11.

[10] H. Zhang. Simulation of Neutron Data of Intrinsically Disordered Proteins and Nucleic Acids: Part II. ACNS 2014, Jun 1-5, 2014. Knoxville, USA. WORKSHOP

[11] Anjum, N. 2014. GSoC: *GenApp Integration with Apache Airavata* http://www.google-melange.com/gsoc/proposal /public/google/gsoc2014/nadeemanjum/5632763709358080

[12] Apache Software Foundation, *Thrift*, http://thrift.apache.org/