Apache Airavata: A framework for Distributed Applications and Computational Workflows

Suresh Marru
Indiana University
smarru@indiana.edu

Lahiru Gunathilake
Indiana University
lginnali@indiana.edu

Chathura Herath
Indiana University
cherath@indiana.edu

Patanachai Tangchaisin
Indiana University
ptangcha@indiana.edu

Marlon Pierce
Indiana University
mpierce@indiana.edu

Chris Mattmann
NASA JPL
chris.a.mattmann@jpl.nasa.gov

Raminder Singh
Indiana University
ramifnu@indiana.edu

Thilina Gunarathne
Indiana University
tgunarat@indiana.edu

Eran Chinthaka
Indiana University
eran.chinthaka@gmail.com

Ross Gardler
Open Directive
rgardler@opendirective.com

Aleksander Slominski
IBM Research
aslom@us.ibm.com

Ate Douma
Hippo B.V.
a.douma@onehippo.com

Srinath Perera
Lanka Software Foundation
hemapani@opensource.lk

Sanjiva Weerawarana
Lanka Software Foundation
sanjiva@opensource.lk

ABSTRACT

In this paper, we introduce Apache Airavata, a software framework to compose, manage, execute, and monitor distributed applications and workflows on computational resources ranging from local resources to computational grids and clouds. Airavata builds on general concepts of service-oriented computing, distributed messaging, and workflow composition and orchestration. This paper discusses the architecture of Airavata and its modules, and illustrates how the software can be used as individual components or as an integrated solution to build science gateways or general-purpose distributed application and workflow management systems.

Categories and Subject Descriptors

K.6 [**Management of Computing and Information Systems**]: Software Management—*development*

General Terms

Design, Standardization, Management, Experimentation

Keywords

Scientific Workflow, Science Gateways, Grid Applications

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GCE'11, November 18, 2011, Seattle, Washington, USA.
Copyright 2011 ACM 978-1-4503-1123-6/11/11 ...$10.00.

1. INTRODUCTION

The nature of computational problems has evolved from simple desktop calculations to complex, multidisciplinary activities that require the monitoring and analysis of remote data streams, database and web searched and large ensembles of simulations. Various frameworks have emerged to address these needs, along with software platforms that provide a community of users with the ability to easily solve computational problems. Particularly, science gateways [35] have harnessed the vast amount of computational resources from grid [14] and cloud [2] computing while reducing the learning curve and barriers to entry by introducing simpler abstractions. Gateways offer a convenient interface for accessing data, collaborating with colleagues, running simulations, and performing data analysis without the need to understand the intricacies of the underlying infrastructure.

Workflow systems continue to be the dominant abstractions; prominent examples include Taverna [26], Triana [33], Kepler [23], and Pegasus [10]. User communities have rallied around these tools to build collaborative infrastructures and science gateways including MyExperiments [17], nanoHUB [22], the LEAD gateway, [7], CaGrid [30], GridChem [12], Earth System Grid [4], and CIPRES [25]. These represent a very small sub-set of the landscape and the majority are open source academic research software. However, due to lack of cohesive open community and meritocratic contribution models and a neutral venue, core features are continually reinvented. These core layers can and should be normalized through technological evolutions.

Apache Airavata, inherited from the Open Gateway Computing Environments (OGCE) workflow system [27] attempts to mitigate this continual reinvention by fostering an open development *community*, not just open source software. Airavata focuses on the baseline tools of application, workflow, job and data management systems, while synergis-

tically working with external portal, user management and security frameworks. The Airavata project will align with the highly successful Apache Software Foundation's [3] meritocratic philosophies. Alongside the technical goals, Airavata aims to engage committed user base throughout the software life cycle, building upon existing tools where possible, developing reproducible builds and tests, and, more importantly steering the project by meritocracy.

The Airavata framework uses general-purpose industry standards and can incorporate emerging distributed and scalable infrastructures like Apache Hadoop [5]. In this paper we discuss the flexibility and extendibility of the Airavata architecture, and provide a brief roadmap. Section 2 discusses the evolution of the project, while Sections 3 and 4 present the over all architecture of Airavata, its modules, and the salient features of using the components individually or as a integrated solution. Section 7 discusses the Apache way and community building aspects and Section 6 discusses related work. The paper concludes with a summary and discussion of the roadmap in Section 8.

2. AIRAVATA EVOLUTION

Apache Airavata originated from the Extreme Computing Lab [13] at Indiana University then led by Dr. Dennis Gannon. The concepts and initial versions of the code are a byproduct of more than a dozen PhD dissertations and years of research and development efforts. The software was initially developed to meet the challenging goals of the Linked Environments for Atmospheric Discovery (LEAD) [15] project. LEAD has pioneered new approaches for integrating, modeling, and mining complex weather data and cyberinfrastructure systems to enable faster-than-real-time severe weather forecasts. LEAD goals required creating a dynamically adaptive, on-demand, grid-enabled workflow system supporting long running applications and on-demand computing. The infrastructure led to wide usage in various educational and research efforts including the National Collegiate Weather Forecasting Competition[8], the Storm Prediction Center's Hazardous Weather Test bed [6] and Vortex2 Tornado Tracking Experiments. LEAD has subsequently produced close to 450 research publications combined in all disciplines.

The OGCE project [29] has subsequently generalized, enhanced, tested, and maintained the LEAD workflow suite. OGCE has promoted collaborative software development through SourceForge and has worked with various institutions. The OGCE project and the TeraGrid Science Gateways group [28] have integrated, the OGCE workflow suite [27] into a wide spectrum of science disciplines including GridChem [12], UltraScan [11], EST Pipeline, BioVLab [36] and ParamChem [16].

Sustaining middleware through evolution of the underlying resources is a daunting problem. Sustaining open source software beyond the interest and funding from initial developers and funding agencies is a bigger challenge. To address these issues, Suresh Marru and Marlon Pierce have pioneered the idea of a science gateway software-based Apache project since late 2008. Many Apache members have guided further generalization of the software to foster a broader Apache Airavata community.

3. ARCHITECTURE

Airavata's primary goal is to support long running applications and workflows on distributed computational resources. The architecture is built on the concept of service oriented architectures (SOA) and is designed to be a software framework comprising modular components as illustrated in Figure 1. The goal of the Airavata framework is minimalist architectural design (i.e., a thin layer), a conceptually simple to understand architecture; and an architecture that is easy to install, maintain, and use. Its most important feature includes is the ability to use components separately or as an integrated solution.

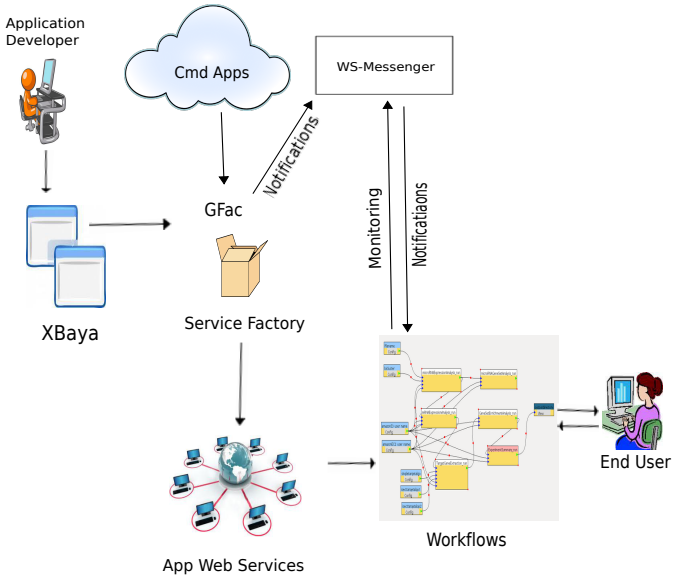

Figure 1: Airavata Overview

Airavata focuses on the following features:

- Desktop tools and browser-based web interface components for managing applications, workflows and generated data.
- Sophisticated server-side tools for registering and managing scientific applications on computational resources.
- Graphical user interfaces to construct, execute, control, manage and reuse scientific workflows.
- Interfacing and interoperability with various third party data, workflow and provenance management tools.

Airavata includes modules for: Generic Application Factory Service - GFac, XBaya Workflow Suite, an interpreted workflow enactment server, information and data registry API, and the WS-Messenger publish/subscribe-based messaging system. Each of these modules is developed, built and packaged as stand alone component as well as integrated into the Airavata suite. The modules can be mixed matched and integrated into existing infrastructures with very minimal effort. The paper [28] details some uses of these components:

- XBaya can be used to orchestrate any standard compliant WSDL web services [9]. It is a general purpose BPEL graphical client.
- GFac generated axis2 application services can be used by any generic web service clients.

- XBaya, GFac and JCR Registry can be used to meet the above two use cases and have a persistent information repository.
- XBaya, GFac, JCR Registry can be integrated with WS Messenger to provide the above features including asynchronous monitoring.
- XBaya, GFac, JCR Registry, WS Messenger can be integrated with Workflow Interpreter Service to provide fire and forget workflow executions with deferred monitoring.
- The entire system can be used to build end-to-end gateways and a full-featured dynamically interactive workflow system.

4. AIRAVATA MODULES

4.1 Application management

Application developers write in various programming languages, and programs typically are designed for use by a single user. Since most science problems are inter-connected, scientists need to share applications, combine multiple application logics in a flexible but planned order, and orchestrate them as workflows. Airavata enables scientists to share and compose their applications to create workflows and to execute them while hiding most of the computer science related details from the end user.

Airavata GFac provides a generic framework to wrap an application as a service interface in Java. This service layer separates the invocation from the communication layer supporting multiple protocols like SOAP, REST, or JSON. Thus GFac can generate a SOAP, REST or native Java interface to any command line application. Currently, Apache Airavata focuses on the Axis2 and core java implementations, but is well architected to make it possible to incorporate the GFac-Core into any service frameworks.

The application provider first describes the application through input, outputs, deployment information, temporary working directories, and remote access mechanisms for file transfers and job submissions and registers this information with a registry service. Once applications are registered, GFac Distributed Application Management handles the file staging, job submission, and security protocols associated with executions. Furthermore, the service acts as the extensible runtime around which extensions such as sharing, auditing and resource scheduling.

During execution, the application schedule is determined and the input data files specified by input parameters are staged to the computational host. The underlying application is then executed using a job submission mechanisms currently, grid, cloud, SSH and local submissions are supported. The framework monitors the status of the remote application, and periodically publishes activity information to the event bus. Once the invocation is complete, the application service tries to determine the results of the application invocation by searching the standard output for user-defined patterns or by listed a pre-specified location for generated data products. The application service runtime is implemented using a processing pipeline based on the Chain of Responsibility pattern [34], where the pipeline can be altered by inserting interceptors. The resulting architecture is highly flexible and extensible, and provides the ideal architectural basis for a system that supports a wide range of requirements.

4.2 Workflow management

The Airavata XBaya workflow system provides a programming model that allows the scientist to program experiments using service-oriented architecture that abstracts the complexities of the underlying middleware. Airavata XBaya provides interfaces for composition, execution, and monitoring of the workflows as illustrated in Figure 2.

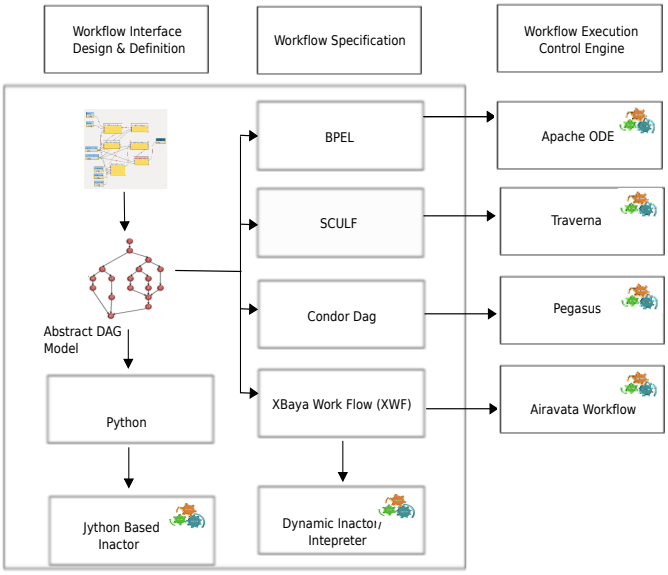

Figure 2: Airavata Workflow System

Workflow composition: Apache Airavata XBaya consists of a convenient GUI interface for workflow composition, a workflow enactment engine/interface, and a workflow monitoring module. XBaya by design decouples composition and monitoring from the orchestration of the workflow although it does provide an embedded workflow enactment engine integrated with the workbench. As a scientific workflow suite XBaya is often expected to run long running computations. It often delegates the workflow execution to a persistent orchestration engine. while the XBaya workbench can monitor the progress of the workflow asynchronously. The workbench provides a convenient drag and drop GUI for SOA based service composition along with other functionalities like service discovery, registry lookup and workflow experiment management.

Workflow orchestration: XBaya provides a unique pluggable architecture for selecting the orchestration engine. When a user composes a workflow using the XBaya workbench, it builds an abstract directed acyclic graph (DAG) which is independent of any workflow runtime. There are pluggable compiler modules that are capable of producing workflow execution scripts for target runtimes. Figure 2 illustrates how the DAG can compiled into different runtimes. Currently XBaya supports BPEL 2.0 and Apache ODE [18]; SCUFL and Taverna [26]; DAGMAN and Pegasus [10], Jython scripts and the XBaya Interpreter Service.

Each of these workflow runtimes has its own strengths; for example, Apache ODE is a robust SOA based workflow orchestration engine well suited for long-running applications. The Pegasus workflow system is well suited for parametric sweeps whereas the XBaya Interpreter engine is strong in dynamic and user interactive workflow execution. It is also

capable of generating a Jython script that can be executed in any Jython runtime, independent of workflow infrastructure.

Workflow interpreter: XBaya provides a built-in workflow enactment engine that provides extremely dynamic and interactive workflow execution. As the name suggests, the workflow interpreter provides an interpreted workflow execution framework rather than a compiled workflow execution environment. Once the user composes and launches the workflow the workflow interpreter will start executing the workflow DAG. Topological sort of the DAG provides execution ordering of the tasks, which allows the identification of the initial ready task set and the incremental addition of new tasks to the ready list as previous tasks finish execution. What is unique about this type of execution model is it allows the user to pause the execution of the workflow as necessary and update the DAGÕs execution states – or even the DAG itself – and then resume execution, with changes immediately picked up by the workflow interpreter.

The workflow interpreter can be used as an embedded workflow enactment engine within the XBaya GUI or as an interpreter service that may run as a persistent service. The functionality provided by the workflow interpreter allows fine-grained control over the workflow execution, which translates into the following functionalities.

- Static workflow interactions
- Dynamic change to workflow inputs, workflow rerun.
- Stop, pause, and resume execution of the workflow. Pause workflow at a particular (debug) point.
- Dynamic change in point of execution; workflow smart rerun.
- Fault handling and exception models.
- Dynamic workflow interactions.
- Reconfiguration of workflow activity.
- Dynamic addition of activities to the DAG.
- Dynamic removal or replacement of activity in the DAG.

Workflow monitoring: The XBaya workbench allows the user to monitor the progress of the workflow in real-time. Monitoring includes the services which are currently executing, and those that are done. It also provides the state of the job submissions to the batch job queues, and the data transfer status. Workflow monitoring in the XBaya workbench functions in two modes, synchronous and asynchronous. The synchronous mode applies only when the XBaya interpreter runs embedded in the workbench. This allows the execution engine to be aware of which tasks are ready, running, finished, or waiting and thus accordingly to update the DAG execution state. However, when the workflow execution is delegated to a persistent service like the Apache ODE server or XBaya interpreter service, asynchronous monitoring is necessary. This is achieved using the WS-Eventing based notification messages published by different components of the workflow system. If a user launches a workflow instance a unique topic will be associated with that instance and all the workflow components such as the workflow engine, and factory service will publish status notifications about the workflow to the event bus using the topic assigned to the workflow instance. This asynchronous monitoring is truly asynchronous; for example the user can launch the workflow, start monitoring, and then lose internet connectivity. Once reconnected the XBaya workbench will bring back the monitoring to the current execution state of the workflow.

4.3 WS-Messenger

WS-Messenger is a web services based messaging system which guarantees scalable, reliable and efficient message delivery. WS-Messenger contains message broker, message box, and WS-Messenger client libraries. WS-Messenger acts as the nervous system within Airavata infrastructure, but can very well be integrated into any WS-Eventing/WS-Notification standard compliant infrastructures. WS-Messenger is deployed as an Axis2 service providing inherent support for different transports including HTTP 1.0, HTTP 1.1 and JMS. Custom transport protocols can be added using the Axis2 framework's transport plug-ins [32]. WS-Messenger is integrated with MySQL and Apache Derby database implementations. The embedded Derby database is deployed within the same Java process. The system also supports persistent subscriptions and notifications and therefore tolerates failures and restarts of the application server. The Figure 3 illustrates the asynchronous capabilities further described in this section.

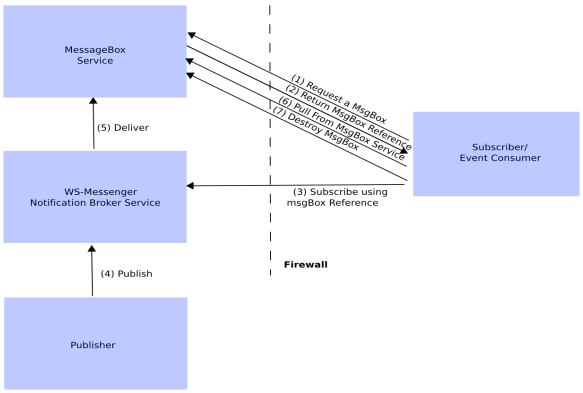

Figure 3: Airavata WS Messenger System

Message broker: The message broker service within WS-Messenger system handles filtering and delivering of messages to event sinks and subscription management. WS-Messenger supports filtering of messages based on XPath expression and topic expression. Message broker is implemented on web services specifications and provides the mediation between WS-Eventing specification and WS-Notification specifications. WS-Eventing and WS-Notification specifications are two competing specifications defining the interfaces for web services-based publish/subscribe systems. The web services interfaces and the notification message formats defined in these two specifications are fundamentally different. WS-Messenger provides a mediation approach facilitating interoperability with supported services. WS-Messenger can automatically convert the notification message formats to make sure that the message "on the wire" can be understood by the event consumers. For example, a WS-Notification consumer can receive notification messages published by a WS-Eventing publisher and vice versa. The event consumer type is determined by the subscription request message type. Here we assume that the subscribers and the event consumers use the same specification. This is a valid assumption for most cases since the subscribers needs to know the loca-

tion of the event consumers. Further details are discussed in [19]

The Airavata WS-Messenger is significantly improved over the predecessor OGCE version in regard to improving efficient message delivering schema implemented by consumer blacklisting to minimize the overhead of message delivering failures. Message delivering performance has been improved in this version by introducing a parallel message delivering sub system. The blocking I/O overhead is reduced in configuring the NIO transport. WS-Messenger used the NIO transport provided by Apache Synapse project to enable non-blocking I/O in Axis2. These improvements are discussed in detail in [21].

Message Box for asynchronous pull: WS-Messenger decouples the consumer tolerating network failures and firewalls as illustrated in Figure 3. In this setting, the message box acts as an intermediate notification storage between the broker and the event sink. Users can create a message box and provide its URL as the event sink URL during the subscription. When messages are published to the message broker, notifications will be sent to the message box. The message box stores these incoming messages in the database or keeps them in memory, depending on the WS-Messenger configuration. The message box service includes operations to pull the messages from the message box, so whenever the event sink is ready it needs only invoke the message pulling API and receive the notifications. When the message box is no longer needed, the subscriber can send another web service request to destroy the message box.This is a useful feature in a scenario where an event source cannot send notifications directly to an event sink running behind a firewall. This deployment is a very common firewall configuration between the De Militarized Zone (DMZ) and the green zone in an enterprise deployment.

Airavata Messenger usage: Within Airavata, WS-Messenger is used for user and system communications between GFac, XBaya and the workflow interpreter. XBaya in monitoring mode subscribes to a pre-specified topic to which the workflow engine and GFac publish status notifications. XBaya receives notifications and visualizes the workflow progress by means of color-coding of tasks being executed.

The following steps illustrate the usage in detail:

- XBaya creates a message box, subscribes to it and then invokes a workflow, with the created message box endpoint reference (EPR) passed in context.
- XBaya then sends a subscription request to WS-Messenger with a particular topic and message box URL received as the event sink.
- Then workflow instrumentations in XBaya sends the notifications using the same topic to the message broker.
- Individual application services created by GFac are instrumented to publish detailed statuses and send progress to the EPR received in the workflow execution context.
- XBaya translates the messages into graphical colors and input and output extractions.

4.4 Information and data registry

Airavata provides capabilities for an integrated workflow and data management system. To supper these capabilities, the information and data must be made persistent in a registry service. To adhere to the philosophies of Airavata for layering over existing stable third party components, Airavata usage is consolidated into a thick-client API. This API allows for the portability of the infrastructure and avoids being locked in into any specific registry architecture. This section describes the usage of the registry through a standard API and summarizes Airavata usage as illustrated in Figure 4

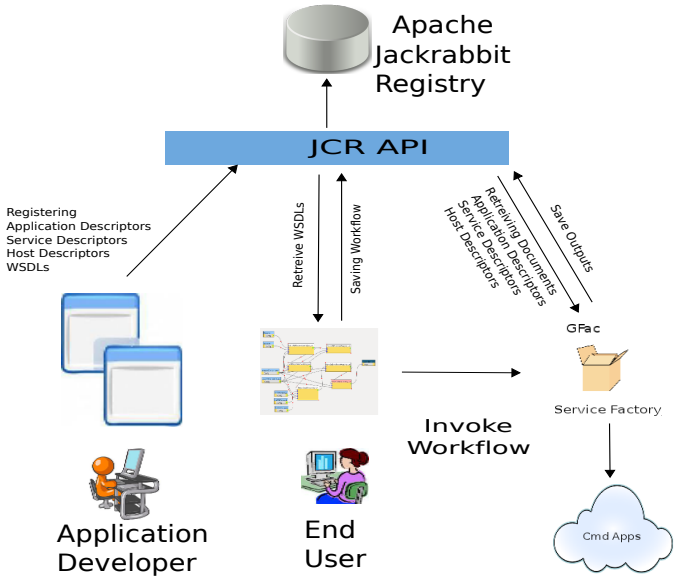

Figure 4: Airavata Registry Usage

JCR specification: The content repository for the Java Technology specification, developed under the Java Community Process JSR-170 [20], aims to standardize the Java API to repositories. The specification provides a unified API under the javax.jcr namespace that allows a user to access any specification-compliant repository implementation in a vendor-neutral manner. A major advantage of JSR-170 is that it is not tied to any particular underlying architecture. The back-end data storage for a JSR-170 implementation, for instance, may be a filesystem, a WebDAV repository, an XML-backed system, or even an SQL database. Furthermore, the export and import facilities of JSR-170 allow an integrator to switch seamlessly between content back ends and JCR implementations. Finally, the JCR provides a straightforward interface that can be layered on top of a wide variety of existing content repositories, while simultaneously standardizing complex functionality such as versioning, access control, and searching. There are open source implementations of JSR-170 such as Apache Jackrabbit, Alfresco, and WSO2 Registry.

Airavata registry API: The Airavata registry functionality to store and retrieve data avoids locking into custom API's to content repository implementations, seamlessly integrates with existing repositories. This is achieved by the JCR-2.0 specification and the JSR-170 implementations. The Airavata-specific API is layered over the generic JCR API and is integrated with the Apache Jackrabbit implementation. The API is modularized as a common component reused by XBaya and GFac components to store and retrieve data. In addition to the service information, the registry is used to catalog workflow consumed inputs and generated output data. The registry is also used to advertise and retrieve persistent service information. The use

of the registry for GFac service registration allows for late service binding and load balance between many application instances.

5. COMMUNITY ENGAGEMENT

Airavata is proposing to adapt and enhance a defined architecture of services and user interface components that can be used in part or in whole by science gateways. These components follow industry standards and may be used by gateways developed with a wide range of programming languages and frameworks.

Apache, as a neutral development venue independent of individual universities and specific funded research projects, has the potential to welcome usage and contributions from emerging science gateways along with industry partners. Airavata will facilitate collaboration among tool developers, gateway deployers and end users. The user community will be actively and openly involved in defining requirements, designing, developing, and maintaining the Airavata software. Apache Airavata is undergoing incubation at the Apache Software Foundation.

Community: Any user is considered to be part of Airavata Community if they participate in the project, including: providing feedback; writing or updating documentation; helping new users; recommending the project to others; testing the code and reporting bugs; fixing bugs; providing feedback on required features; writing and updating the software; creating artwork, or translating to different languages.

Meritocracy: Following the Apache way, the community is encouraged to engage in the project and to actively contribute. If the project management committee feels a person has "earned" the merit to be part of the development community, they grant direct access to the code repository. Thus enlarges the group and therefore enhances the ability of the group to more effectively develop and maintain the program. This principle is called "meritocracy" meaning government by merit. Airavata follows and encourages meritocratic processes in all decision making.

Coding practices: Airavata will follow standard coding practices, design patterns, and an approach to commit early and often. The commits provide an opportunity for the community to subscribe to the mailing list and agree or object to the changes or directions of the project. Silence is considered approval.

Voting and community involvement: All decisions in Airavata will follow the general guidelines of Apache. Discussions other than voting in new committers are public and open to all mailing lists. Decisions include release time lines, features, implementation decisions and general directions of the project. End users can provide feedback by joining the mailing lists and participating in voting. All negative votes and feedback from users are taken into account. The community has opportunity to influence and be part of the governance of the project. In rare circumstances, where the active contributors disagree with non-contributing users or unjustified feedback, the project management committee has the right to veto community votes. The following voting format is followed for all decision making. Typical voting time is 72 hours to allow community members from across the globe to participate.

- +1 means "I agree with this and will help make it happen"
- +0 means "I agree with this but probably won't make it happen, so my opinion is not that important"
- -0 means "I don't agree with this, but I'm offering no alternative so my opinion is not that important"
- -1 means "I don't agree and I am offering an alternative that I am able to help implement"

Lazy consensus: In order to facilitate speedy progress and avoid latencies with a geographically distributed developer base, Airavata supports the concept of lazy consensus. When a developer is convinced that the community would like to see a certain action happen, he or she can assume that there is already consensus and proceed with the work without approval or need to call a vote. The community's support is assumed unless explicitly objected. However, if an objection is raised, the code change or other directional changes must be reverted based on voting decisions.

6. RELATED WORK

The Airavata framework is related to a number of other efforts in the fields of grid computing, software architecture, web-services frameworks for data management (including content detection and analysis), and science gateways. One of Airavata's goals is to establish an open community forum for synergistic efforts. We illustrate here a few representative cross-section's of those efforts, to provide the basic breadth in the area.

At the Apache Software Foundation (ASF), Airavata is most related to the Apache Object Oriented Data Technology (OODT) [24] framework. Apache OODT is a software architecture-based framework for highly distributed and data intensive information systems. The CAS-PGE task wrapper is a highly specialized workflow task that runs an underlying scientific algorithm, providing it information from the file manager (staged files, metadata, options and switches), the workflow manager (the next task in the sequence, rules, conditions, etc.) and the resource manager (disk space, CPU allocation, supporting software libraries, etc.). Airavata's registry and workflow components are related to OODT's file manager (catalog), and workflow services. Both are written in Java. Both efforts leverage web services to expose the underlying function. Whereas Airavata has focused on the SOAP-based web services stack, OODT focuses on XML-RPC and REST-ful web services. Both projects have the opportunity to inform one another and we are working to ensure coordination between the two.

Another related framework is Apache Tika [1], a toolkit for content detection and analysis. Tika is a major component leveraged in the Apache Lucene search engine stack (specifically in SolrÕs ExtractingRequestHandler), and also within the Apache Jackrabbit content repository that Airavata has recently adopted. Tika provides facilities for the detection of file types (by MIME magic; regular expressions and glob patterns), and by XML-schema comparison. Tika is a framework that could be leveraged by Airavata within its registry component to provide file classification, and extracted metadata from underlying parsing libraries that Tika integrates including Apache POI.

As discussed in detail in Section 1, there are many science gateway efforts providing similar capabilities to Airavata. The project's focus is not to introduce yet another work-

flow/gateway software stack, but to open the community for the widely used OGCE gateway building toolkit. The project will interoperate with existing projects which support the open community model in addition to open source software. As an example, initial implementations supported the Grid Process Orchestration Engine (GPEL)[31]. GPEL was specifically designed for eScience usage with requirements for long running workflows and a decoupled client end enactment engine. GPEL provided many experiences to early adopters of the BPEL efforts. However, Airavata was ported to support Apache ODE to ensure sustainability. Airavata is based on the concepts of service oriented architecture and all services run within the Tomcat container. The web services are based on Axis2. The web interfaces within the Airavata software are synergistically developed with another Apache incubator project (Rave).

7. ROADMAP AND POTENTIAL COMMUNITY

The Apache Airavata project is in its initial release versions with the following components and features:

- GFac-Axis2: An Axis2 web service which can consume user-defined command line descriptions and generate Axis2 application web services.
- XBaya: A desktop application which lets users construct, execute, and monitor workflows.
- XBaya is also used in the first release as a user management, application management. and data browser. In a release we will develop web gadgets to be deployed into containers like Apache Rave.
- Workflow interpreter: Axis2 wrapper around XBaya dynamic executor. This is a simple and interactive workflow execution engine. Future releases will support Apache ODE in addition to the interpreter service.
- WS-Messenger: WS-Eventing/WS-Notification based messaging system.
- Registry-API: A thick client registry API for Airavata to put and get documents. Current JCR implementation is supported by Jack-rabbit.

Users will be able to build and deploy the entire suite or selected components with one-click Apache Maven installation and Axis2 deployment. The toolkit will enable user management in Jack-rabbit, constructing and executing third party and GFac generated services as workflows, monitoring progress through events and cataloging and browsing workflow inputs and outputs.

Through the open community process and user engagement through the mailing lists, science gateway developers currently using the OGCE framework have expressed interest in porting to the Apache Airavata software stack. Key initial users include GridChem, the UltraScan Gateway, BioVLab project, the Ocean Land Atmospheric System and Nuclear Physics LCCI project. We expect developers in these projects to participate in the mailing lists as users and in a meritocratic way will be encouraged to become committers.

8. CONCLUSIONS

In this paper we introduced the general purpose Apache Airavata software framework and discussed the architecture and the various components. The framework will facilitate users from a wide variety of environments to selectively and securely share applications as web services, and to construct, execute and monitor workflows with these services. The suite also includes features of on-demand service creation, workflow orchestration and monitoring. Furthermore, these components can be used individually or collectively to build small- to large-scale gateways.

9. ACKNOWLEDGMENTS

This work is partially funded by the National Science Foundation through the award: OCI-1032742 (SDCI NMI Improvement: Open Gateway Computing Environments - Tools for Cyberinfrastructure-Enabled Science and Education). The initial code donation to Airavata is developed by the Extreme Computing Lab at Indiana University funded by the Linked Environments for Atmospheric Discovery project under NSF award ATM-0331480.

10. REFERENCES

[1] Apache. Tika. `http://tika.apache.org/`.

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[3] ASF. Apache software foundation. `http://incubator.apache.org/airavata/`.

[4] D. Bernholdt, S. Bharathi, D. Brown, K. Chanchio, M. Chen, A. Chervenak, L. Cinquini, B. Drach, I. Foster, P. Fox, et al. The earth system grid: Supporting the next generation of climate modeling research. *Proceedings of the IEEE*, 93(3):485–495, 2005.

[5] D. Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 2007.

[6] K. Brewster, D. Weber, K. Thomas, K. Droegemeier, Y. Wang, M. Xue, S. Marru, D. Gannon, J. Alameda, B. Jewett, et al. Use of the lead portal for on-demand severe weather prediction. In *Sixth Conference on Artificial Intelligence Applications to Environmental Science, 88th Annual Meeting of the American Meteorological Society, New Orleans*, 2008.

[7] M. Christie and S. Marru. The lead portal: a teragrid gateway and application service architecture: Research articles. *Concurr. Comput. : Pract. Exper.*, April 2007.

[8] R. Clark, S. Marru, M. Christie, D. Gannon, B. Illston, K. Droegemeier, and T. Baltzer. The lead-wxchallenge pilot project: enabling the community. In *24th Conference on IIPS*, 2008.

[9] F. Curbera, F. Leymann, T. Storey, D. Ferguson, and S. Weerawarana. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR, 2005.

[10] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, pages 131–140. Springer, 2004.

[11] B. Demeler. UltraScan: a comprehensive data analysis software package for analytical ultracentrifugation experiments. *Analytical Ultracentrifugation: Techniques And Methods*, pages 210–229, 2005.

[12] R. Dooley, K. Milfeld, C. Guiang, S. Pamidighantam, and G. Allen. From proposal to production: Lessons learned developing the computational chemistry grid cyberinfrastructure. *Journal of Grid Computing*, 2006.

[13] Extreme. Computing lab. `http://extreme.indiana.edu`.

[14] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 2001.

[15] D. Gannon, B. Plale, S. Marru, G. Kandaswamy, Y. Simmhan, and S. Shirasuna. Dynamic, adaptive workflows for mesoscale meteorology. *Workflows for e-Science*, pages 126–142, 2007.

[16] J. Ghosh, N. Singh, Y. Fan, S. Marru, K. Vanomesslaeghe, and S. Pamidighantam. Molecular Parameter Optimization Gateway (ParamChem). In *Proceedings of the 2011 TeraGrid Conference*. ACM, 2011.

[17] C. Goble and D. De Roure. myexperiment: social networking for workflow-using e-scientists. In *Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 1–2. ACM, 2007.

[18] T. Gunarathne, C. Herath, E. Chinthaka, and S. Marru. Experience with adapting a ws-bpel runtime for escience workflows. In *Proceedings of the 5th Grid Computing Environments Workshop*, pages 7:1–7:10. ACM, 2009.

[19] Y. Huang, A. Slominski, C. Herath, and D. Gannon. Ws-messenger: A web services-based messaging system for service-oriented grid computing. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 8–pp. IEEE, 2006.

[20] Java. Content repository. `http://jcp.org/aboutJava/communityprocess/review/jsr170/`.

[21] R. Jayasinghe, D. Gamage, and S. Perera. Towards improved data dissemination of publish-subscribe systems. In *2010 IEEE International Conference on Web Services*, pages 520–525. IEEE, 2010.

[22] G. Klimeck, M. McLennan, S. Brophy, G. Adams III, and M. Lundstrom. nanohub. org: Advancing education and research in nanotechnology. | *Computing in Science & Engineering*, 2008.

[23] B. Lud scher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 2006.

[24] C. Mattmann, D. Crichton, N. Medvidovic, and S. Hughes. A software architecture-based framework for highly distributed and data intensive scientific applications. In *Proceedings of the 28th international conference on Software engineering*, pages 721–730. ACM, 2006.

[25] M. Miller, W. Pfeiffer, and T. Schwartz. Creating the cipres science gateway for inference of large phylogenetic trees. In *Gateway Computing Environments Workshop (GCE), 2010*, pages 1–8. IEEE, 2010.

[26] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045, 2004.

[27] S. Perera, S. Marru, and C. Herath. Workflow Infrastructure for Multi-scale Science Gateways. In *TeraGrid Conference, June*, 2008.

[28] M. Pierce, S. Marru, R. Singh, A. Kulshrestha, and K. Muthuraman. Open grid computing environments: advanced gateway support activities. In *Proceedings of the 2010 TeraGrid Conference*. ACM, 2010.

[29] M. Pierce, S. Marru, W. Wu, G. Kandaswami, G. von Laszewski, R. Dooley, M. Dahan, N. Wilkins-Diehr, and M. Thomas. Open grid computing environments. In *Proceedings of the Fourth Annual TeraGrid Conference*. Citeseer, 2009.

[30] J. Saltz, S. Oster, S. Hastings, S. Langella, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz. cagrid: design and implementation of the core architecture of the cancer biomedical informatics grid. *Bioinformatics*, 2006.

[31] A. Slominski. Adapting bpel to scientific workflows. *Workflows for e-Science*, pages 208–226, 2007.

[32] P. Srinath, H. Chathura, E. Jaliya, C. Eran, R. Ajith, J. Deepal, W. Sanjiva, and D. Glen. Axis2, middleware for next generation web services. In *Web Services, 2006. ICWS'06. International Conference on*, pages 833–840, 2006.

[33] I. Taylor, M. Shields, I. Wang, and A. Harrison. The triana workflow environment: Architecture and applications. *Workflows for e-Science*, pages 320–339, 2007.

[34] S. Vinoski. Chain of responsibility. *Internet Computing, IEEE*, 6(6):80–83, 2002.

[35] N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam. Teragrid science gateways and their impact on science. *Computer*, 41(11):32–41, 2008.

[36] Y. Yang, J. Choi, C. Herath, S. Marru, and S. Kim. Biovlab: Bioinformatics data analysis using cloud computing and graphical workflow composers. *Cloud Computing and Software Services*, page 309, 2010.