Open Community Development for Science Gateways with Apache Rave

Marlon Pierce
University Information Technology Services
Indiana University
Bloomington, IN 47408
mpierce@cs.indiana.edu

Raminderjeet Singh
University Information Technology Services
Indiana University
Bloomington, IN 47408
ramifnu@indiana.edu

Zhenhua Guo
University Information Technology Services
Indiana University
Bloomington, IN 47408
zhguo@indiana.edu

Suresh Marru
University Information Technology Services
Indiana University
Bloomington, IN 47408
smarru@cs.indiana.edu

Pairoj Rattadilok
University Information Technology Services
Indiana University
Bloomington, IN 47408
prattadi@indiana.edu

Ankur Goyal
University Information Technology Services
Indiana University
Bloomington, IN 47408
asgoyal@umail.iu.edu

ABSTRACT

Science gateways enable researchers and students to use distributed scientific computing infrastructure (cyberinfrastructure) through Web browsers and Web-enabled desktop clients. This paper describes the use of the open source, open community Apache Rave project as the basis for developing science gateways. Building on Apache Shindig (for OpenSocial Gadgets) and Apache Wookie (for W3C Widgets), Rave provides an out-of-the box deployment that can be used to host reusable social Web components. Rave is based on the Spring MVC framework and so can also be extensively customized or extended with (for example) custom database back-ends and authentication modules. In this paper we consider Rave as a development platform for science gateways and discuss how the source code may be extended through three use cases that focus on gateway security requirements. A major consideration of this paper is how to design Rave as a development environment so that developers can make local customizations and extensions freely on both a rapidly changing code base (during Rave's initial development), and (later) between stable code bases during version upgrades. We conclude with a discussion of the implications of developing science gateways and other cyberinfrastructure software within the Apache Software Foundation and present its potential advantages.

Categories and Subject Descriptors

H.5.3 [**Information Storage and Retrieval**]: Online Information Systems – *Web-based services*

General Terms

Design, Documentation, Human Factors, Standardization

Keywords

Science Gateways, Web Computing, Grid Computing, Cloud Computing, OpenSocial, W3C Widget

1. INTRODUCION

Science gateways provide Web-based access to distributed scientific computing infrastructure, ranging from campus clusters to national scale cyberinfrastructure such as the NSF-funded XSEDE [1][2]. Gateways build on general purpose middleware (such as Globus [3] or Condor [4]), distributed security infrastructure, scheduling and queuing systems (such as PBS), wide area file systems, and mass storage, extending these resources to provide user-centric views of the infrastructure, the ability to collaborate online, and higher level functionality such as the ability to create and manage scientific workflows. For a discussion of the gateway software stack (Web portals, Web and REST services, and workflows), see [5].

Considerable effort has gone into the development of science portals and gateways for multiple scientific communities, with the gateway tracks at TeraGrid 2010-2011 [6][7] and the Gateway Computing Environment workshop proceedings [8][9][10] providing overviews. Efforts have also been made to generalize and develop open source software that powers these gateways, including the HUBzero [11] project and the authors' involvement in the Open Gateway Computing Environments project [5]. We present in this paper our efforts to go beyond open source software to an open community model, fostered by the Apache Software Foundation. As discussed below, we believe this is an important missing component of many previous open source cyberinfrastructure efforts to obtain sustainable, diverse developer bases.

This effort is being realized through two Apache Incubators: Apache Rave, which focuses on component-based Web front ends and services for managing user environments and collaborations; and Apache Airavata, which focuses on scientific workflows, science application wrapping, messaging, and registries. This paper describes Apache Rave, a joint effort of Hippo Software, SURFnet, Mitre Corporation, and the Open Gateway Computing Environments project.

Apache Rave's goal is to provide an out-of-the box, Web-based collaborative environment. Rave builds upon Apache Shindig, the reference implementation for the OpenSocial specification that provides the gadget rendering engine. Shindig provides JavaScript and Web service (REST and RPC) programming interfaces and basic implementations for developing Web-based gadgets and server-to-server interactions. Rave's goal is to extend these basic implementations to provide production-quality user and group management implementations. Rave also provides gadget layout management and its own set of REST/RPC services for interacting with gadgets. In addition to OpenSocial, Rave also is developing support for W3C widgets [12] to be co-deployed in the container. We use the terms "widget" and "gadget" interchangeably in this paper (since this it is a Rave goal to make this so), but the two have precise definitions.

As we have discussed previously [13] [23], we believe OpenSocial is a good model for building science gateways out of reusable, social network-enabled gadget components that can be extended to support interoperable collaboration. This was the motivation for the development of the OGCE Gadget Container. With Apache Rave, however, we have the opportunity to jointly develop the concepts of the Gadget Container openly with an international community of developers. Unlike the Gadget Container, Rave must go beyond the requirements of science gateways to support a diverse set of deployments, such as corporate intranet portals. Correctly handled, these pressures will lead to well-designed software that is flexible and extensible.

This paper provides an overview of Apache Rave and uses three simple usage scenarios to illustrate how developers can extend the software. Rave is being designed from the beginning to be a development environment as well as an out-of-the-box tool. We conclude with a discussion of the Apache Software Foundation's open community model for software development and the potential benefits of this model to the general cyberinfrastructure community. Science gadget development itself is not in the scope of this paper; please refer to [14] for more discussion on this topic.

2. RAVE SOFTWARE ARCHITECTURE

Rave is developed in Java using the Spring MVC framework (version 3) and is packaged with Apache Maven for compiling, testing, local installation, and distribution. Rave user interface components depend upon initial server-side rendering through Spring but use project-developed and third-party JavaScript libraries for richer client-side interactivity. Communications between the browser and server are handled with AJAX-style invocations of REST services. Example browser-to-server communications are state changes such as adding new pages to a display, adding gadgets to a page, and moving gadgets within the displayed area. Early support for mobile views, including automatic detection of different devices, is also available.

In addition to the user interface managing components, Rave also includes a "widget store" service that allows users to select widgets to add to their displays. A user and gadget administration interface and underlying user role capability are currently in development. Rave is co-deployed with Apache Shindig in the same servlet container (such as Apache Tomcat) and using the same backing database (although Rave and Shindig are separate web applications), but these may be made separable in future releases.

We next review major features of the Spring framework in order to discuss Rave's implementation and (more importantly for this paper) support for developer extensions. Spring is based on the Inversion of Control design pattern, in which the developer creates Java Beans and specifies their initial property values and dependencies in XML configuration files. The Spring MVC container is then responsible for instantiating the beans and handling inter-dependencies such as the order of instantiation. Web pages can be developed as Java Server Pages (JSP) with optional additional tag libraries but are rendered through a dispatcher servlet rather than loaded directly. The primary components are listed below. Figure 1 illustrates the components' relationships. Gateway developers may want to extend or override several of these as discussed in the next section.

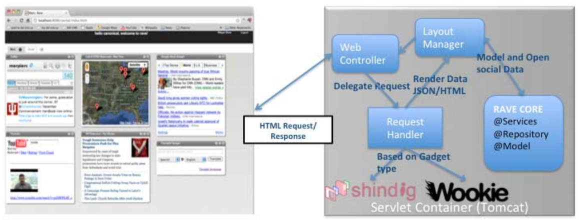

Figure 1 Apache Rave's user interface (left) and server-side components.

Models: these are JavaBeans that provide the basic data object structures for Rave. Example models include User, Page (modeling a page containing gadgets or widgets), Region (modeling a region, typically a column of a page, that contains widgets), and RegionWidget (modeling a widget instance on a page). Models are serialized into repositories for persistence. A developer may wish to extend the User object, for example, to provide custom attributes specific to his/her science gateway.

Controllers: these are associated with specific Java Server Pages and REST and RPC services. If a user requests a specific URL, this is handled by a controller. The response may be HTML to be rendered by a browser or a JSON message to be consumed by another service. Controllers are annotated with *@Controller*. Rave places all JSP user interface components in *WEB-INF/views/,* where they are inaccessible except through the Spring dispatcher servlet. A gateway developer may wish to add to or extend the existing controllers to customize the account creation process, add additional, gateway-specific view pages, or expose additional REST services, for example.

Services: the processing associated with specific controllers, such as adding a new user to the system after an HTML FORM action request, is handled by (internal) services. Services are annotated with *@Service.* Note these services are not directly exposed as network services. A developer may wish to extend Rave's services to support new controllers. A gateway developer will typically need to implement a new service to support the extended Rave model object and its associated controller.

Repositories: access to the data management system (for users, pages, user views, and other models) goes through repository classes using the Java Persistence API (JPA). Rave currently uses Apache OpenJPA to handle the object-relational mappings with backing relational databases. Modifications to a model object (such as User object extensions made by a gateway developer) may also require changes to the associated Repository classes. In principal, this layer could also provide access to non-relational ("NOSQL") databases, but this has not yet been investigated.

The Spring MVC container is responsible for instantiating all instances of these classes. Dependencies are annotated with *@Autowired* and specified in configuration files. This annotation can be used for example to associate one or more Services with the Controller that needs them: the Service dependencies are passed to the Controller through its constructor.

Rave configuration files are specified in the XML files listed in Table 1. Developers may need to modify one or more of these files when making extensions.

Table 1 Rave configuration files

Configuration File	Description
applicationContext.xml	Used to create instances of all beans, controllers, and services, and to pass in any initial parameters. Developers may need to add additional Java Beans here to support their extensions.
applicationContext-security.xml	This configuration information is used to specify which URL patterns Spring security will intercept and handle or not, to enable OpenID support, and to specify the authentication provider. Developers of science gateways may want to change the default authentication module.
dispatcher-servlet.xml	This is to describe controller servlet and its properties. Web component and its configuration properties. Data marshaling with JAXB to handle requests is described here.
dataContext.xml	Used to set up the in-memory H2 database and to populate it with demo accounts. Deployers may wish to override these settings to populate with their own initial set of users.

Finally, we note that Spring MVC provides a number of built-in modules that are interesting for gateways, particularly for security issues. Hashing and salting of passwords, for example, are handled by Spring's SaltSource and PasswordEncoder interfaces. Specific implementation beans (ShaPasswordEncoder, ReflectionSaltSource) and their initial parameters (using the *username* parameter as the salt source, for example) are specified in the *applicationContext.xml* file, which a developer may wish to modify. The Rave service that manages user accounts and local passwords accesses the hashing and salting beans through auto-wiring. For a fuller review of Spring security, see [14].

Rave provides an environment that supports both deployment and extensibility. The Rave portal is built using the single command "mvn clean install", which will produce both Java JAR files and a comprehensive WAR application, which can be used both to run Rave and to develop extensions. Rave development with Eclipse and JRebel IDEs is described at http://incubator.apache.org/rave/ide-settings-and-debugging.html; these guidelines also apply to "sandbox" Rave extension projects described in the next section. Running "mvn deploy" will create a comprehensive deployment directory that includes Apache Tomcat with both Rave portal and custom Shindig WAR files correctly deployed. Developers who wish to distribute extended versions of Rave can thus use Rave's Maven setup to create their own distribution releases.

In summary, Rave's Spring-based framework and specific implementations provide a systematic way to make enhancements and customizations to the code base. The downside is that a developer (rather than a deployer) must modify or replace several program and configuration files. This introduces significant management problems for developers working from Rave's trunk (where changes are frequent) rather than a stable release. Developers who base their gateways on a specific stable release version may also experience difficulties integrating their changes into newer releases and so get "stuck" on an obsolete release. This is commonly handled in many projects by branching. However, in Rave we have chosen a different approach.

3. EXTENDING RAVE: SCIENCE GATEWAY SCENARIOS

Apache Rave's goals are to be both a useful out-of-the-box Web environment and an extensible platform for further development. This section focuses on the latter and is illustrated with three short science gateway scenarios, but the approach is general and can be applied to more complicated scenarios such as integrating with an LDAP directory server. Rave's "sandbox" extension [15] build provides the basic model for directory layout and jar dependencies. In particular, developers will minimally need the Rave components listed in Table 2. See [16] for further details on Rave's project structure.

To support development, Rave's maven build produces Java JARS, Maven POM files, and Java WAR files that are deposited in the user's local Maven repository as well released WAR files to be run in a Tomcat server. Rave deployers only need the WARs, but developers program to Rave's APIs through Maven-specified dependencies. Thus a developer does not need to modify or add files for overriding (for example) the User model within Rave's code base. The developer does not even need Rave's source code. Instead the developer creates a new Maven project, adds dependencies on Rave's JARs, WARs, and POMs as needed, and includes only locally modified or added files in his/her project directory.

Examples of these development patterns are available from Rave's sandbox. The sandbox examples show how to set up a Maven project that depends on Rave components and which correctly overrides the default Rave WARs with overlay customizations. Following the sandbox's model, a developer can establish dependencies on a specific version of Rave (drawn from the developer's local repository or online from Apache) and then isolate all local changes to a developer directory.

Running "mvn clean install" will then create new WARs that incorporates both Rave and the user's local modifications.

Table 2 Rave components for extensions.

Component	**Description**
rave-portal-dependencies	A Maven POM file listing all Rave-produced JARs and third party dependencies.
rave-portal-resources	A Java WAR file containing all Rave web resources.
rave-shindig	A Java WAR containing Rave modifications and extensions to Apache Shindig.

We next consider three example scenarios in which a science gateway developer would extend Rave: a) replacing the default JSP view pages with customized ones; b) providing GridShib and community credential support; and c) integrating with the CILogon online authentication service. Although simple, they help validate our extension approach before we consider more complicated gateway development.

3.1 Local Customizations to View Pages

Rave view pages (such as *login.jsp*, *home.jsp*, *newaccount.jsp*, and *store.jsp*) provide browser-renderable user interfaces that handle self-evident tasks. If a developer only wishes to make cosmetic changes to these pages (changing HTML layouts, adding banners, and modifying CSS, for example), we recommend capturing these changes in a project that follows the Rave sandbox template rather than making changes directly to the Rave source code. That is, a developer should create a new Maven project based on the sandbox example and place changes to *home.jsp* in the project's *src/main/webapps/WEB-INF/views* directory. Running "mvn clean install" on the branch will create a new WAR that includes new files and replaces Rave's default files with local overrides.

Developers may also want to modify the *dispatcher-servlet.xml* and *applicationContext.xml* files to, for example, use a default destination other than *home.jsp* and add additional pages that are exempt from Spring's authentication requirements (such as additional script and image directories). These should be placed in the new project's *src/main/resources/* directory. With proper Maven dependency configuration (shown in Rave's sandbox), the Maven build process will override the default XML files with the developer's local configurations.

3.2 Providing Community Credential Support

The next case study focuses on common tasks of Science Gateway developers: getting X.509 community credentials and applying GridShib [19] [20] attributes for users when they log in. GridShib attributes are used to add user-identifying information to a shared community credential; the attributes (such as user email, instance creation time, and source IP address) can be used for accounting and for identifying security problems with specific accounts without disabling the entire community. Sample code for this is also available for the Rave sandbox. Again, it is recommended that the developer create a local Maven project that includes only the code overrides and extensions, rather than making changes directly to Rave's trunk code. We will assume the gateway has access to the community credential through some approved process, such as from a secure local file that can be read by the gateway using JGlobus classes. Better approaches for initially acquiring user and community credentials are currently in development [17].

In any case, we will assume that the community credential has been obtained and must now be associated with a user account when the user logs in. Rave provides a UserService interface that is implemented by default with *DefaultUserInterface.java*. Spring determines the implementation to use through the *applicationContext-security.xml* configuration file. The developer thus should first create the "sandbox" Rave Maven project layout as before. This sandbox should depend upon Rave's repository components (Table 2) and contain the developer's modified *applicationContext-security.xml* file (which will tell Rave to load the developer's replacement for DefaultUserService.java) and the developer's new implementation of the UserService interface.

GridShib-enabling Java portals also requires a number of configuration files: in Rave, these should be deployed in the sandbox Maven project's *src/main/resources* directory. Maven will place these files in the resulting WAR's *WEB-INF/classes* directory and will also optionally filter any variable property values needed for the local deployment. An example of this is available from the Rave sandbox (http://svn.apache.org/viewvc/incubator/rave/sandbox/science-gateways/).

One issue we encountered with GridShib-enabling Rave was its requirement that *xml-apis.jar*, *xercesImpl.jar*, and *xalan.jar* be placed in Apache Tomcat's "endorsed" directory. We did not find a clean way to automate this within Rave's Maven build system in a single step, which uses the *cargo* plugin to deploy the WARs into Tomcat. This only applies to the development phase of projects (deployed versions would not use the *cargo* plugin but would instead deploy WARs into a production Tomcat server), but it does make development awkward. One possible solution is to use Maven's Ant plugin to manage the deployment process more directly, as was done in the OGCE Gadget Container. However, this and other options (such as providing an appropriate plugin) need to be investigated.

3.3 Supporting CILogon Authentication

The CILogon prproject [21] provides an online service that science gateways can use to give users a secure login to the gateway through campus authentication mechanisms (like InCommon) or OpenID. The motivations are a) to support campus bridging with national cyberinfrastructure (such as the NSF's XSEDE), b) to simplify authentication for gateway users, c) to give gateway users a trusted, secure service for authentication instead of relying on the gateway to correctly manage the user's security credentials, and d) to return a generated X.509 credential to the gateway. The gateway can then use this X.509 credential to identify the user to other secure services running externally to the gateway server. CILogon provides open source clients in several programming languages to support gateway integration.

Rave's support for CILogon authentication requires the following steps. First, the developer must create a POM file with Rave dependencies (as in previous examples) and also a dependency on the CILogon client jar. Next, the developer modifies Rave's login.jsp page to direct users to CILogo's online service. The developer should customize CILogon's "welcome" servlet. The developer will also need to customize Rave's configuration file, *applicationContext.xml*, to override Rave's default URL security filtering: the "welcome" servlet and URLs of CILogon's required client callback REST API must be accessible.

Next, the developer must add support for CILogon's REST API methods: *ready, startRequest, success,* and *failure*. The *startRequest* and *failure* methods are implemented in CILogon's client jar and only need to be specified in the developer's *web.xml* (which should replace Rave's default version). The developer's primary coding task is to implement the *success* API callback, which CILogon's service calls to send the X.509 certificate generated during the authentication process to the science gateway. As before, Rave's SpringMVC framework supports REST services through the *@Controller* annotation. The developer must create a new controller that implements this service. The new controller will need also to override the default UserService interface to implement additional actions, such as extracting the user's distinguished name from the X.509 credential and associating it with a specific local portal account before redirecting the browser to *home.jsp*.

3.4 Advanced Gateway Requirements

The previous three examples were chosen to validate Rave's extensibility approach for simple gateway scenarios. These extensions are available outside Rave's source tree trunk, in the project sandbox. An important consideration is how to release these specialized pieces to the gateway community. One approach is to simply deem them generally required by the larger Rave community. This would take place if enough science gateway developers become involved in Rave and would vote for gateway-specific pieces to be included in each release, accompanied by the usual test requirements and peer-review of releases. A second and perhaps superior solution is to develop an offshoot Apache incubator project from Rave that can be more specialized. Some discussion of this has already taken place within Rave, as we consider how to manage plugin gadgets (such as user interfaces to Airavata services). This would keep Rave's core code focused and generic with optional modular extensions. The Apache HTTPD project has followed this pattern for several years.

We finally note that, although Rave is intended to support gadgets and widgets as its primary user interface component, a developer could develop all user interfaces directly within the SpringMVC framework; that is, the developer could create all interfaces as JSPs and place them in the *WEB-INF/views* directory. This is relevant for many gateways, which have complicated interfaces that require a lot of the browser's real estate rather than dashboard views composed of many small components. Gadgets support full-browser views ("canvas" mode), but making this the default view complicates Rave's current assumptions about layout management.

4. SCIENCE GATEWAYS AND THE APACHE COMMUNITY PROCESS

Although the National Science Foundation Office of Cyberinfrastructure has taken important steps in requiring open source licensing for the code that it funds in its SDCI and SI2 programs, we believe there are further steps to be taken in sustainability and governance of cyberinfrastructure software and are participating in Rave to investigate these issues. We believe the Apache Software Foundation is an important model for building sustainable software since it is concerned with the more comprehensive issue of fostering open software communities, not just open source licensing.

Access to a project's source code is irrelevant if there are no well-defined processes for contributing patches, holding the developers accountable for bugs and features, and contributing to or officially joining the project that creates the software. Apache projects have well-defined mechanisms and incentives for addressing these problems, and new incubator projects (such as Rave) are required to demonstrate healthy growth such as developer diversity before graduation. New members ("committers") are elected by the other project members based on contributions to the project. Contributions can be software patches and extensions, but committers can also be testers, documentation-writers, Web designers, and insightful mailing list critics.

Open discussions and design processes are also required in open communities. Decisions on software design and project direction must be made in the open. Apache requires all project discussions to take place on open (and publically archived) developer mailing lists. All project committers have a vote. Apache thus represents a neutral ground for software projects that allow competitors to contribute to the same baseline software.

We believe this is an important model that needs to be more thoroughly investigated by cyberinfrastructure projects as a path towards sustainability, better project governance, and as a bridge to other software development communities. Successful Apache projects have diverse development teams that don't rely upon the same sources of funding, so third-party developers can have greater confidence that the project will live beyond the involvement of any particular developer. Apache membership also has obvious workforce development implications, as successful student developers on research projects will become members of Apache, and science gateway projects can attract proven developers from the Apache community. Finally, Apache provides infrastructure and governance models for software development communities.

The primary disadvantage of Apache to cyberinfrastructure software development projects is the potential loss of control of the project as new members from the larger Apache community join. This is a relatively low risk to a project that has been designed for extensibility and takes these requirements into account early in its development. As we have discussed in Section 3, Apache Rave considers extensibility as a first-class requirement.

Another potential disadvantage to cyberinfrastructure projects is Apache's assumed developer mobility. Apache members are encouraged to get involved in multiple projects, which assumes that an intelligent developer with general knowledge and skills can quickly come up to speed on a project. This may not be true for highly specialized projects, particularly computational science communities that assume at least a graduate-level background to understand the science behind the algorithms. Another risk is the failure of a project to graduate from the Incubator. This is not a risk for Rave, which already meets the developer diversity minimum requirements (developers from three institutions). There is a potential risk however for cyberinfrastructure projects that may be too specialized to attract broader interest. Choosing good Apache champions and mentors with prominent standing in the community and detailed knowledge of Apache processes for the incubators is essential.

We hope with Apache Rave and Airavata to not only produce successful projects but to become good mentors and champions for future projects.

Finally, proposals and funding must be carefully considered within the NSF community. It is likely that the Apache model for academic software development should follow an open-source business model: Cyberinfrastructure developers would not be funded specifically to develop the core code (since this would involve non-academic and possibly international institutions that are ineligible for funding), but rather academic teams would be funded to provide extensions on the general-purpose core software to support the specific needs of a scientific or scholarly community. With proper design for extensibility (as Rave is trying to provide), these extensions can be made even if the Apache community deems them inappropriate or too specialized for the core code.

5. RELATED WORK

The Grid/Gateway Computing Environments workshop series [8][9][10] provides an overview of science portals and gateways. The TeraGrid and XSEDE Science Gateways program has played a key role in driving the development of gateways for the US research community. Gateways may be built on a number of frameworks. The XSEDE (formerly TeraGrid) User Portal builds on the Liferay portal framework. Liferay itself includes support for both Java portlets and OpenSocial gadgets. The NanoHUB project's front end significantly extends the Joomla content management system (CMS); this is generalized, repackaged, and released as HUBzero [11] for general gateway development. The FutureGrid portal uses the Drupal CMS for its portal. The use of the open source social networking software Elgg for science gateways is described in [17]. Prominent open source portal implementations based on the OpenSocial framework include SURFNet's COIN portal, Mitre Corporation's Open Source Enterprise Container (OSEC), and the Open Gateway Computing Environments project's Gadget Container. These three groups, along with open source CMS vendor OneHippo Software, are the primary initial contributors to Apache Rave.

Compared to prominent CMS frameworks (Drupal, Liferay, and Joomla), Rave is first and foremost a social networking and gadget rendering platform. It lacks CMS capabilities. On the other hand, by following the W3C Widget and OpenSocial gadget specifications, it is possible to develop Rave components using many different programming languages and frameworks (JSP, PHP, and others). This does require a different programming model, however, as pages must rely on the OpenSocial JavaScript libraries and more generally on AJAX-style programming rather than HTML FORM actions: the server-side of the gadget should support REST invocations from JavaScript.

An important issue is whether science gateway developers will want to develop applications as gadgets (with the associated JavaScript libraries, assuming the developer wants to go beyond using IFrames). It is possible that Rave will need to become more of a backend service, concentrating on implementing OpenSocial's REST and RPC services rather than its JavaScript libraries, in order to support the user interface requirements of gateways.

Rave is a new project and still in many ways immature, although actively developed. These present tradeoffs to gateway developers. Developers wanting a stable product that can be used to provide substantial functionality will not be satisfied with Rave in its current (incubating) form. On the other hand, developers who are interested in actively developing and contributing to a new project, and who are interested in learning more abut the Apache processes, are good fits for Rave.

Cyberinfrastructure software sustainability issues are surveyed at length in [22]. Other relevant gateway-related models for sustaining software include the Sakai Foundation and the HUBzero Consortium. The Sakai Foundation is a non-profit organization with operations directly funded by member universities and commercial partners. This approach obviously requires a large scale of commitment at high levels within the university that is not easily obtainable by more typically OCI-funded projects. In contrast, the Apache Software Foundation is a community that exists to foster open community development but is not geared towards a single software product or even "platform ecosystem" as is the case for Eclipse.

Sustainability and governance are important issues for science gateways. Many first generation gateways were based on the portlet model and used GridSphere [24] as the portlet framework. GridSphere had substantial success (it was used by the TeraGrid User Portal for several years, for example), but ultimately it could not sustain this success within the gateway community after the key developers left the project. Project governance (as with many other e-science projects) was not thought through from the beginning in order to address the inevitable turnover of developers. Governance issues such as the process for adding new code committers, making peer-reviewed releases, and publically discussing design decisions were not sufficiently considered. We hope with Apache Rave to learn from both GridSphere's successes and its shortcomings.

6. CONCLUSIONS AND FUTURE WORK

We have described the Apache Rave project and its extensions to support science gateways, including both code extensions and display customizations. The goal of our involvement in Apache Rave is more than just developing and packaging open source software. Sustainability and integration with diverse teams outside the normal gateway community are also crucial. Sustainability of small and medium sized projects has not been addressed, and the non-profit foundation approaches used by larger projects are not viable for the typical gateway.

7. ACKNOWLEDGEMENTS

Apache Rave is a community-developed, incubator project of the Apache Software Foundation. Rave's project mentors and contributors are listed at http://incubator.apache.org/projects/rave.html, and specific contributions are tracked through the project's mailing list, Jira tracking system, and SVN code repository.

This work is partially supported by the NSF Office of Cyberinfrastructure's SDCI program, award # 1032742.

8. REFERENCES

[1] Nancy Wilkins-Diehr, Dennis Gannon, Gerhard Klimeck, Scott Oster, Sudhakar Pamidighantam: TeraGrid Science Gateways and Their Impact on Science. IEEE Computer 41(11): 32-41 (2008)

[2] Nancy Wilkins-Diehr: Special Issue: Science Gateways - Common Community Interfaces to Grid Resources. Concurrency and Computation: Practice and Experience 19(6): 743-749 (2007)

[3] Ian T. Foster: Globus Toolkit Version 4: Software for Service-Oriented Systems. J. Comput. Sci. Technol. 21(4): 513-520 (2006)

[4] Douglas Thain, Todd Tannenbaum, Miron Livny: Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience 17(2-4): 323-356 (2005)

[5] Marlon Pierce, Suresh Marru, Raminder Singh, Archit Kulshrestha, and Karthik Muthuraman. 2010. Open grid computing environments: advanced gateway support activities. In *Proceedings of the 2010 TeraGrid Conference* (TG '10). ACM, New York, NY, USA, , Article 16 , 9 pages. DOI=10.1145/1838574.1838590 http://doi.acm.org/10.1145/1838574.1838590

[6] TeraGrid 2010 Proceedings (Gateway Track): http://dl.acm.org/citation.cfm?id=1838590

[7] TeraGrid 2011 Proceedings (Gateway Track): http://dl.acm.org/citation.cfm?id=2016741

[8] GCE 2008 Proceedings: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4738437&isYear=2008

[9] GCE 2009 Proceedings: http://dl.acm.org/citation.cfm?id=1658260&picked=prox&CFID=41670638&CFTOKEN=26444986

[10] GCE 2010 Proceedings: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5668967

[11] Michael McLennan and Rick Kennell. 2010. HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering. *IEEE Des. Test* 12, 2 (March 2010), 48-53. DOI=10.1109/MCSE.2010.41 http://dx.doi.org/10.1109/MCSE.2010.41.

[12] Marcos Cáceres (ed), Widget Packaging and XML Configuration, W3C Proposed Recommendation 11 August 2011.

[13] Zhenhua Guo, Raminderjeet Singh, and Marlon Pierce. 2009. Building the PolarGrid portal using web 2.0 and OpenSocial. In *Proceedings of the 5th Grid Computing Environments Workshop* (GCE '09). ACM, New York, NY, USA, , Article 5 , 8 pages. DOI=10.1145/1658260.1658267 http://doi.acm.org/10.1145/1658260.1658267

[14] Peter Mularien, *Spring Security 3,* Packt Publishing, 2010.

[15] Extend Rave to Build Your Own Portal: http://incubator.apache.org/rave/documentation/rave-extensions.html

[16] The current Rave project organization and rationale are described at https://issues.apache.org/jira/browse/RAVE-171.

[17] Roger Curry, Cameron Kiddle, and Rob Simmonds. 2009. Social networking and scientific gateways. In *Proceedings of the 5th Grid Computing Environments Workshop* (GCE '09). ACM, New York, NY, USA, , Article 4 , 10 pages. DOI=10.1145/1658260.1658266 http://doi.acm.org/10.1145/1658260.1658266

[18] Jim Basney, Terry Fleury, Von Welch: Federated login to TeraGrid. IDtrust 2010: 1-11

[19] Von Welch, Jim Barlow, Jim Basney, Doru Marcusiu, Nancy Wilkins-Diehr: A AAAA model to support science gateways with community accounts. Concurrency and Computation: Practice and Experience 19(6): 893-904 (2007).

[20] GridShib Project Site: http://gridshib.globus.org/

[21] CILogon Project Site: http://www.cilogon.org/

[22] Stewart, Craig A.; Almes, Guy T.; Wheeler, Bradley C. (eds.), Cyberinfrastructure Software Sustainability and Reusability: Report from an NSF-funded workshop. Available from http://hdl.handle.net/2022/6701

[23] Zhenhua Guo, Raminderjeet Singh, Marlon Pierce, and Yan Liu. Investigating the Use of Gadgets, Widgets, and OpenSocial to Build Science Gateways. In *Proceedings of the 7th e-Science*, Dec 2011, Stockholm, Sweden.

[24] Jason Novotny, Michael Russell, Oliver Wehrens: GridSphere: a portal framework for building collaborations. Concurrency - Practice and Experience 16(5): 503-513 (2004)