Building the PolarGrid Portal Using Web 2.0 and OpenSocial

Zhenhua Guo, Raminderjeet Singh, Marlon Pierce

Community Grids Laboratory, Pervasive Technology Institute
Indiana University, Bloomington

2719 East 10th Street, Bloomington, Indiana 47408

{zhguo, ramifnu, marpierc}@indiana.edu

ABSTRACT

Science requires collaboration. In this paper, we investigate the feasibility of coupling current social networking techniques to science gateways to provide a scientific collaboration model. We are particularly interested in the integration of local and third party services, since we believe the latter provide more long-term sustainability than gateway-provided service instances alone. Our prototype use case for this study is the PolarGrid portal, in which we combine typical science portal functionality with widely used collaboration tools. Our goal is to determine the feasibility of rapidly developing a collaborative science gateway that incorporates third-party collaborative services with more typical science gateway capabilities. We specifically investigate Google Gadget, OpenSocial, and related standards.

Categories and Subject Descriptors

H.3.5 [Information Storage and Retrieval]: Online Information Services – Web-based services. D.2.13 [Software Engineering]: Reusable Software – Reusable Models

General Terms

Design, Management

Keywords

Collaboration tools, Web 2.0, REST, Gadget, OpenSocial, OpenID, OAuth

1. INTRODUCTION

Science gateways [1, 2] are composed of user interface components supported by back-end services and capabilities. This approach has several advantages: common components can be shared between projects, adopting a service architecture provide a distributed version of the model-view-controller design pattern, and service instances can support multiple front-ends. A common approach in many gateways of the previous generation was to adopt the JSR 168 portlet component model and WSDL/SOAP style web services. The TeraGrid User Portal [3] and the LEAD science gateway [4] typify this approach.

We believe that while the fundamental concepts of a component-service gateway are still useful, it is time to revisit some of the software and standards used to actually build gateways. Two important candidates are the Google Gadget component model and the REST service development style for building gateways. Gadgets are attractive for three reasons. First, they are much easier to write than portlets and are to some degree framework-agnostic. Second, they can be integrated into both iGoogle (Google's Start Page portal) and user-developed containers. Finally, gadgets are actually a subset of the OpenSocial specification [5], which enables developers to provide social networking capabilities. Standardization is useful but more importantly one can plug directly into pre-existing social networks with millions of users without trying to establish a new network from scratch. RESTful services for gateways have been reviewed elsewhere and are appropriate for building information services. As we discuss below, REST-style services are an important part of the OpenSocial framework and are supported by new security specifications.

The PolarGrid project [6] is an NSF-funded MRI project that provides computing support for the Center for the Remote Sensing of Ice Sheets (CReSIS, https://www.cresis.ku.edu/). CReSIS is primarily concerned with using Synthetic Aperture Radar (SAR) techniques to obtain information on the depth of the Greenland and Antarctic ice sheets and their underlying rock beds. PolarGrid provides both in-the-field computing clusters for initial image processing (useful for finding problems with radar equipment, for example) and larger clusters at Indiana University for full-scale image processing needed to make community data products. Image processing is needed to produce data products of multiple levels. The initial products result from raw image processing and have little need for interactive job submission. However, higher-level products need human interaction and judgment.

In this pilot project, we implement the web services that give users the access to testing the three basic filters (Table 1). The basic scenario we consider here is applying data filters to clean up lower-level data products obtained by initial data processing. We developed three sample filters: Wiener, Median, and FIR1. Filters are implemented using Matlab and then wrapped as Web Services using the OGCE's [7] GFAC tool [8].

Table 1. Testing dataset and filters

Data and Filters	Parameters
Helheim dataset	size: 17023 (w) x 970 (h), ground track: 67 km
Medium filter	horizontal and vertical length (h, v)

SC '09, November 14-20, 2009 Portland, Oregon, USA

Wiener filter	horizontal and vertical length(h, v)
Fir1 filter	cut off frequency (f)

The PolarGrid OpenSocial portal is an effort to provide a collaborative environment for the scientist to work together. We reiterate here that our goal for this study was to evaluate Google and related APIs, not to design the final interface. Thus we adopt generic requirements for collaborative Science Gateways. In summary, scientists must be allowed to work with different data sets, run their experiments on TeraGrid and share their results with collaborating scientists. OpenSocial portals provide an interface for people to connect and work in a collaborative environment. We want to build upon these collaboration tools to make it easy for users to keep track of their work, share the same dataset and avoid duplication of work. Scientists can still maintain their privacy if they want. Desirable features include:

- View CReSIS data sets, run filters, and view results through Web map interfaces;
- See/Share user's events in a Calendar;
- Update results to a common repository with appropriate access controls;
- Post the status of computational experiments.
- Support collaboration and information exchange by interfacing to blogs and discussion areas;

2. SOLUTION APPROACHES

There are two general modes for meeting the requirements listed above: a) providing all services in-house and b) providing a lightweight coupling over third-party collaboration services. Both options can involve standards, best practices, community contributed components, etc. The first option means running all collaboration services on resources (servers, databases, etc) specifically maintained by the Gateway provider. This is suitable for closed, intranet gateways and portals, especially if there are privacy and trust concerns with third party service providers. The second option means relying upon third party collaboration services as much as possible, with the gateway developers building only the minimum integration components. Obviously any real gateway will be a combination of these two modes. We briefly review some existing tools to support each option.

Existing frameworks like Sakai, Moodle, Liferay, Exo and Drupal: All of these are open source portal, CMS, web publishing, collaboration, and social networking frameworks widely used to develop portals for different domains. Both Sakai and Moodle are designed to help instructors, researchers and students collaborate online. Collaboration tools include both general-purpose tools like calendars, notice boards and courseware-specific components like course content and pages. They provide easy interfaces to build customized portals and add modules based on the need. They have developed modules for calendars, discussion boards, and standards-based content repositories.

Building open system using OpenSocial Gadget and using Google services or social network services like Facebook, Twitter etc. for collaboration. In this approach, the developer attempts to integrate in as many third party collaboration services as possible, removing the need to run these directly within the portal. Thus for example instead of using a built-in calendar system provided by a downloadable framework, the gateway uses a pre-existing calendar service such as Google Calendar. This is the approach that we adopt and investigate.

Most of the existing frameworks are also examining OpenSocial and trying to build their components accordingly. Sakai3, for example, is a new architecture expected to be released in 2011 and will be OpenSocial compliant with other Web 2.0 features. The Moodle team is also building OpenSocial framework called Wookie that is currently in Apache Foundation's Incubator at initial development stage.

Liferay, Exo and Drupal already have at least prototyped support for OpenSocial. These consist of a large set of plug-ins available that a portal developer can use to easily bring up a social site using these modules. The question here is what will happen when one needs to move to a different framework later: how much effort it will take? Another question is how far these community frameworks will go and can evolve with the change happening in modern software architectures. We believe the important long-term issue is sustainability of deployed specific gateway instances, not standard compatibility.

We believe the lightweight approach has several inherent advantages if it can be made to work. First, we will be able to take advantage of tools such as Gmail, Google calendars, blog and Twitter feeds, etc that users are already using and familiar with. Second, we believe building on top of third-party systems is more sustainable in the long term. A problem of the purely in-house solutions is that the intellectual content (discussions, documents, project timelines, generated data, and job execution metadata) can be lost when the project ends, the servers are decommissioned, the maintainers take new jobs, and so on. By using prominent third party services, we can offload some of the long term archiving of collaboration data from that gateway itself. Third, many of these services can be integrated into huge existing social networks, which we believe is much easier at this point than tempting users to join yet another startup network.

We note that we make no judgments on the sustainability of community frameworks (Sakai, Liferay, etc) versus third party services: we believe both models can succeed. Our concern is the sustainability of specific gateways built from these frameworks. Sustaining a gateway in the long term requires a significant amount of effort. Large teams can afford to provide most services in-house, but smaller teams cannot. Furthermore, we doubt users will be tempted to join yet another social network at this point. We concede these are our personal opinions. Our goal in this work is to build a system to test those opinions.

We note also that there are subtle problems regarding trust that are beyond the scope of this paper. Users need to be able to trust that the gateway is sustainable or else they won't use it. However, they must also trust a third-party service provider to maintain privacy and ownership of intellectual content. We believe these choices are worth further study.

To support our lightweight approach we have tried different open source tools and API's. We looked into Google Data API's [9] to provide framework to connect to different Google tools to share the data. This API provides a simple protocol to read and write data to various Google tools like Blogger, Calendar, Picasa, Google docs, YouTube etc. It is very easy to use these API's to read/write data. Google Friend Connect [10] is another tool we explored to provide social support to the community. We will explain in detail about how they helped us in our solution. Table 2

summaries design and technology choices we made in current implementation.

Table 2. Summary of design and technology choices

Tech/Design choices	Reason Summary
Web 2.0	Improves usability and responsiveness
Gadget	Makes developers possible to write reusable web components that can be deployed to any Gadget container.
OpenSocial	Makes portal possible to interact with existing large social networks instead of building our own.
REST	Makes applications able to access PolarGrid services using simple HTTP requests.
OpenID	Makes portal able to interact with external OpenID-compliant identity management systems.
OAuth	Makes portal able to interact with external OAuth-protected services.
MyProxy	Makes portal able to interact with security infrastructure of Grid systems.

3. ARCHITECTURE

The PolarGrid portal is designed using Web 2.0 concepts [11] , Google Gadget components [12], OpenSocial collaborative user networks, and RESTful services. We will start by introducing all the components briefly and then go into detail.

Gadgets are a frontend user interaction layer and provide Web based user experience like other web applications. Web Services (RESTful services) wrap backend capabilities and provide flexibility to the client applications to make synchronous/asynchronous calls. Apache Shindig, an open source implementation of OpenSocial specification, provides social networking features to the portal. In our solution we use Shindig as gadget renderer that understands the gadget XML and creates HTML content to be displayed in web browser. We use the OGCE gadget layout manager [13] to arrange both social and non-social gadgets in a layout. The gadget layout manager provides internal user management and also support user authentication using OpenID [14]. User can add additional gadgets to a single layout and also create multiple tabs in same screen. Google Friend Connect provides authentication support using Google, Yahoo, AOL accounts and using those accounts to provide collaborative environment.

In our architecture, the PolarGrid User portal has four main components (see Figure 1). We will explain them one by one and how each component is designed.

The basic processing flow is as follows:

1. A user visits his/her gadget home page, which is served by gadget layout manager
2. The gadget layout manager constructs the user's custom gadget layout in browser and makes use of a gadget renderer (Shindig in our case) to render each gadget XML to HTML/JavaScript. Then the generated HTML/JavaScript code is displayed in browser.

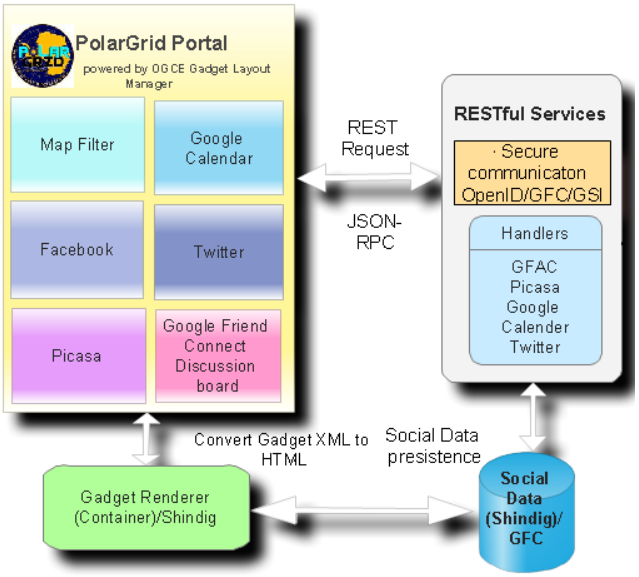

Figure 1. Architecture diagram of PolarGrid Portal

3. Different gadgets may interact with different backend RESTful services to generate output. A JSON response is sent back to the gadget to display the results.
4. Gadgets and RESTful services also query social data using OpenSocial API's by sending requests to Shindig server.

In the processing flow described above, the discussion of authentication and authorization are omitted for simplicity. They will be covered in a separate section. We will describe the components one by one in detail.

3.1 Gadgets (Web widgets)

Gadgets are mini applications that can be developed and rendered by any standard-compliant gadget renderer. They are based on many existing technologies including HTML, JavaScript and CSS, which removes (we believe) many barriers to entry for application web developers; that is, developers can build gadgets using almost any Web language and framework. The gadget renderer is responsible for converting gadget source files to HTML code that can be displayed in most of modern browsers. With the popularity of Web 2.0 in newly developed portals, gadgets are also widely used in these sites. Gadgets are defined by XML files that include both metadata and HTML/JavaScript code. Basically there are two types of gadgets - legacy gadgets and OpenSocial gadgets. Gadgets cannot run in regular HTTP web servers because they need more runtime support including rendering and OpenSocial data service. They are typically served by gadget rendering engines such as Apache Shindig (see below).

We have created several different gadgets for the portal to fit PolarGrid project needs (Table 3).

Table 3. PolarGrid gadgets

Filter Gadget	User can select different parameters to run a filter to create image. Result image will be displayed on Google map.
Blog Gadget	To display the feeds from PolarGrid blog site.

Discussion Board	Google Friend Connect (GFC) gadget to discuss on certain topic.
Filter Images	Picasa gadget to display all the filter images with filter description.
FAQ Gadget	GFC gadget for Question/Answer. Moderator can always control the topics and can block people from the list.
Google Calendar	Calendar gadget to display public PolarGrid-specific activities and tasks.
Twitter Gadget	To read filter execution updates from twitter related to PolarGrid.
Facebook Gadget	User can update status of task directly to Facebook from here.

Currently all the gadgets are deployed in the local shindig server and added in OGCE layout manager. These gadgets may be deployed into an online gadget container like iGoogle. This is useful for development and testing of individual gadgets and also a means to deliver general interesting gadgets to a wider audience.

3.2 Gadget layout manager

Having frontend portal gadgets, we need to put them into a container that manages layout of gadgets and displays them to end-users. We have developed our own gadget layout manager, which makes brings customization of user interface under our control. Gadgets are organized into different groups in arbitrary way. When a gadget group is rendered, all of its contained gadgets are displayed.

Two built-in layouts are provided. One is tab layout that is shown in Figure 2. Each tab corresponds to a gadget group and the user can switch between different tabs by clicking tab title in tab bar. The other is tree layout that is shown in Figure 3. On the left of user interface is a navigation tree. Each internal tree node represents a gadget group. When a user clicks an internal tree node, corresponding gadgets are rendered and displayed in the main panel.

We support two gadget views: home and canvas. In the home view, a gadget shares space with other gadgets within the same group. The advantage of this view is that multiple gadgets are simultaneously displayed on the same page, which is convenient for users who are interested in output of more than one gadget. In the canvas view, a gadget does not share space with other gadgets and occupies all of space. This view is useful for gadgets that need large space for better interactivity. One example of this is a scientific workflow composition gadget. A large editing space helps improve editing efficiency of workflows. For each gadget, the user can easily switch between home view and canvas view.

On-demand gadget rendering is used to improve performance. The user may have several gadgets that are organized into different groups. At any moment, just one gadget group is displayed and the other groups are invisible. Gadget layout manager does not render all of the gadget groups at one time. Gadgets belonging to a group are rendered only when the user switches to that gadget group for the first time. This improves performance by avoiding unnecessary rendering of gadget groups that may never be used by user in a session.

Figure 2. Gadget layout manager - Tab layout

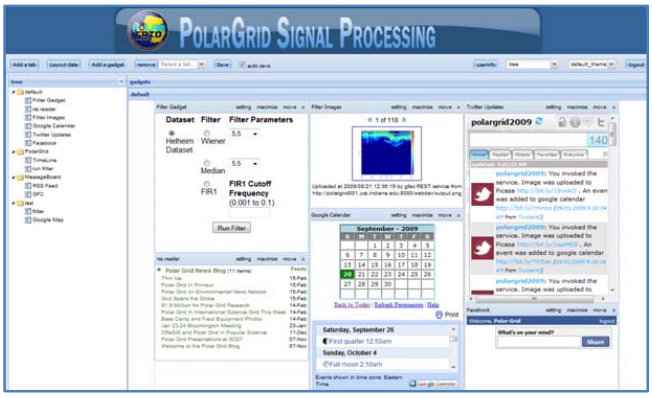

Figure 3. Gadget layout manager - Tree layout

Users can easily export and import their layout data using our container. The exported data includes all information needed to rebuild users' gadget configuration and layout. If the user wants to switch to another instance of our gadget layout manager, he/she just needs to export his/her data and import it to the new instance. Currently we use a custom JSON format to describe layouts. The reason we chose JSON is that it is lightweight and natively supported by JavaScript. There are no foreseeable obstacles that prevent us from supporting more data formats (e.g. XML).

The OGCE gadget container supports the following personalization features.

Drag-and-Drop support: Within a rendered gadget group, the user can drag and drop any gadget to move it to a different location. This is useful because usually the user wants to put similar and related gadgets close to each other.

Dynamic addition and removal of gadgets and gadget groups: To add a new gadget, the user just needs address of the gadget specification file which can be hosted anywhere. When a gadget group is removed, all of its contained gadgets are moved as well.

Theme customization: Several built-in themes are provided. Also developers can customize theme by writing their own CSS files.

Our goal is not to reproduce iGoogle and compete with Google, of course. Our goal is to give communities and developers an alternative by building an open source gadget layout manager that provides rich features, can be run in closed environments, and is under the developer's direct control. So any community that wants to build their own gadget servers instead of using iGoogle can

download and install our gadget container, which has an Apache Maven-based build process. The OGCE gadget layout manager is appropriate for private gateways. OGCE gadget layout manager is almost as fully featured as iGoogle, as is shown in Table 4.

Table 4. Features of OGCE layout manager and iGoogle

Features	OGCE Layout Manager	iGoogle
Gadget Repository	In Development	Yes
Gadget User Preference	Yes	Yes
Dynamic Height Adjustment	Yes	Yes
Drag and Drop	Yes	Yes
Home/Canvas view	Yes	Yes
Theme Customization	Yes	Yes
Layout data export and import	Yes	No

3.3 Gadget Renderer (Shindig)

Shindig is a reference implementation of Gadget and OpenSocial specification. Firstly, it supports gadget rendering. Given a gadget specification file, HTML output is generated that can be displayed in most modern browsers. In the generated HTML, all required gadget JavaScript is served automatically. Secondly, Shindig supports gadget metadata service by which third-party applications can query the metadata of gadgets. For example, developer can query title, description and preferred height of a specific gadget. Thirdly, it supports OpenSocial data serving in a primitive way. Shindig's built-in implementation supports read-only OpenSocial operations applied to in-memory JSON social data representation. Based on design requirements, we have two choices about how to integrate real backend social data. These choices match our prior discussion on open vs. closed approach in section 2.

We are currently using Apache Shindig as our gadget renderer. In the future, we plan to implement our own data service and make it connect to Shindig so that Shindig container can manipulate our social data. In other words, we build our own service to host social data. This is an in-house solution which is appropriate for private gateways or gateways having specific concerns about security. If we delegate identity management to an external party like Google Friend Connect we can also store users' social data in the external party. In this case, all social data are served by an external party used by our system. This solution makes use of existing third-party services, which may improve data sustainability and reduce development cost and time. One important thing is that standard-compliant external services are preferred over proprietary services. The reason is that it is easier for OpenSocial client applications to switch to a different social data service.

3.4 RESTful Web Service for PolarGrid

RESTful Web services provide an interface to expose features of backend services in REST/RPC calls and respond as JSON messages, which can easily be converted to JavaScript in gadgets.

We have developed four different handlers for RESTful services. Each handler plays its own role, and all the handlers are executed in a sequential manner. Error handling is done at each individual component level to report correct error to users.

Table 5. RESTful service handlers

Handler	Function
GFAC Handler	GFAC is a Service toolkit from OGCE to wrap command line applications as Web services and run them on distributed computational resources. Here GFAC acts as a wrapper of MatLab binaries to filter data and also will help in data movement from different resources. This also takes care of GSI security to make use of TeraGrid resources.
Picasa Handler	This handler reads the image URL response from GFAC, extracts the image and uploads it to Picasa using a given Google account and to a particular folder. It also adds description of input parameters to the image.
Google Calendar Handler	This records the activity to the given Google calendar about image data processing. This records the processing time taken to process one data filer.
Twitter Handler	When all the processing is done and the image is uploaded to Picasa, this handler updates the twitter feed with image URL and parameters. Portal users can directly look at the feed and see what processing is already done and what are the results.

Currently, our RESTful service is not protected by any security mechanism. How to protect it with mechanisms described in Section 4 is part of our future work.

4. SECURITY

When building any portal or a gateway, security plays an important role in the architecture and sustainability of any framework or solution. Two fundaemntal aspects of security are authentication and authorization. A traditional GSI credential is used by backend components like GFAC to communicate with TeraGrid. As discussed below, we must go beyond this and investigate more options. For OpenSocial sites and gadgets, OpenID and OAuth are the solutions available for those two aspects.

4.1 Authentication

Authentication is the hallmark of any Web portal. The problem with existing authentication frameworks is that users' profile data stored at different identity management systems is isolated and disconnected. As a result, the user must remember usernames and passwords for many accounts and create lot of redundant profile data. One proposed solution to this problem is OpenID, resulting from community effort. The way OpenID tackles the problem is to define an interaction protocol between providers and relying parties. Providers act as authentication services. Relying parties that want to verify user's identity rely on providers to accomplish this. Users can use a single OpenID to access various services provided by relying parties. Currently, the OGCE gadget layout manager provides built-in user management system and it also supports OpenID as relying party. Considering the number of public OpenID providers, including Google Blogger, Yahoo!, and MySpace, it is highly possible that the user already has an OpenID. Users can use their existing OpenIDs to log in OGCE gadget layout manager. The user can bind their OpenID to an OGCE

gadget layout manager local account. During binding establishment, OpenID Simple Registration Extension [15] and Attribute Exchange [16] are used to retrieve user's profile data as much as possible from provider, which is fed into fields of local account. As a result, the user does not need to type profile information (e.g. name, gender) repetitively.

To maximize richness and flexibility, we propose an identity management system integration architecture (shown in Figure 4) in which both OpenID providers and non-OpenID authentication services are supported.

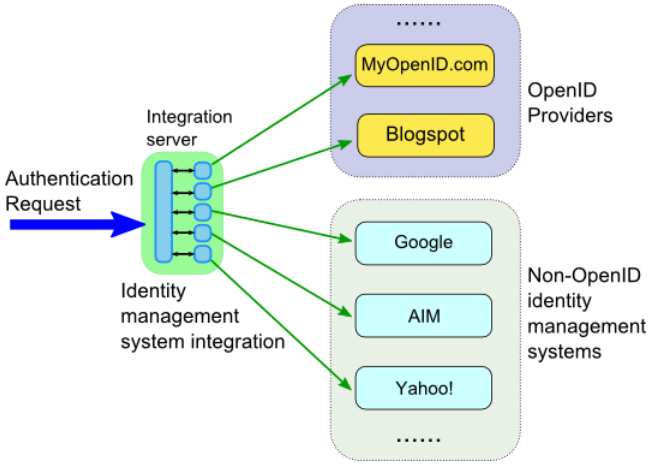

Figure 4. OGCE authentication integration

In the architecture, all authentication communications go through the integration server. After the user types his or her credential information (usually username and password), it is sent to integration server. Then integration server takes over and handles communication details with the backend authentication systems. After authentication is done, the integration server returns authentication results to the end user. The implementation of integration server should be extensible so that developers can develop their own authentication modules to interact with specific backend authentication systems.

Google Friend Connect (GFC) implements integration of multiple authentication systems. Google Friend Connect is not bound to Google Applications. It can provide an authentication service to any website. It enables us to use all common Identity Management Systems and store the OpenSocial data on the server. Besides authentication, Google Friend Connect maintains social data, such as members and messages, associated to each registered websites. We are currently using friend connect collaborations features. We want to integrate GFC to the backend REST services level to validate the user token is correct. GFC has API's to support server-side integration besides client-side integration. We are still in the process of investigation to use maximum features of friend connect.

4.2 Authorization

Authorization is another hallmark of science gateways. For grid systems, usually Grid Security Infrastructure [17] is used. MyProxy [18, 19] server acts as certificate repository for science gateways and manage authentication. One problem is how to combine a gateway user management system with MyProxy certificates. Community accounts [20] are a simple solution in which all gateway users share the same backend grid account, which is used to access grid resources. Some authorization frameworks, including Community Authorization Service [21, 22], Akenti [23] and PERMIS [24], have been proposed to improve original GSI. They work well for centrally coordinated Grids like NSF's TeraGrid. However, in the PolarGrid portal design we are trying to have an open architecture for services. The portal may need to access some third-party services that are not under our control. As a result, GSI and GSI derivatives cannot be used directly although they still can be used to access some centralized Grid systems.

We investigated OAuth [25], which is an open standard tackling the problem of cross-domain authorization. OAuth allows third-party applications to access users' data stored at service providers. This enables our applications to access OAuth-protected external services including Twitter, Google, and Yahoo!. The official OAuth specification only covers three-legged authorization in which the user must be involved to grant or deny resource access requests. For science gateways, this requirement is sometimes too tight considering that in scientific computation batch processing is a common occurrence. One solution is to use three-legged OAuth to start up a complete workflow that consists of multiple jobs. After OAuth processing, an access token is granted that makes the program able to access resources. This is acceptable if all accessed resources are within the same domain as the program. However, if OAuth-protected external resources, such as Google Picasa, are accessed, additional OAuth authorization processes are needed so that the user must manually grant access requests to all external OAuth-protected resources. The reason is that the access token obtained from one domain cannot be used to access resources belonging to another domain. So we have extended OAuth to support two-legged authorization. After the initial delegation of authority, client application can access resources on user's behalf without user involvement. In summary, digital signature using public key cryptography is utilized to prove identity of message sender. As a result, a key management mechanism is necessary. Two-legged OAuth process flow is shown in Figure 5.

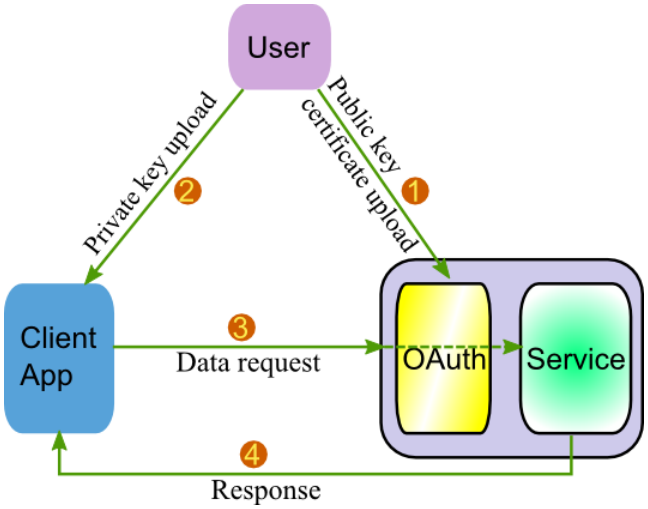

Figure 5. Two-legged OAuth

At first the user uploads public key certificate to OAuth server that protects the backend service. Then the user uploads the corresponding private key to client application. This step actually delegates user's privileges to the client application. After that, the

client application can access data of the OAuth-protected service on behalf of the user. Sometimes the user may not be willing to expose their long-term private keys to a client application because of trust issues. We proposed a solution that combines MyProxy. It is shown in Figure 6.

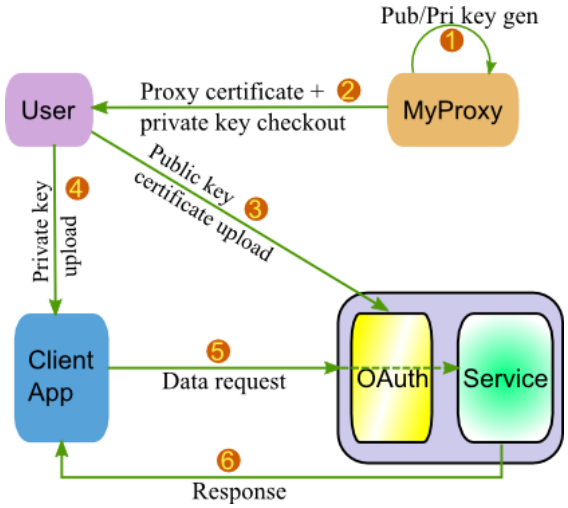

Figure 6. Two-legged OAuth and MyProxy

At first, the user uploads his/her long-term public key certificate and private key to MyProxy server. Then when the user wants to access an OAuth-protected service, he/she first retrieves a short-lived proxy certificate and private key from MyProxy server. The following process matches what is described in Figure 5. Length of validity period of proxy certificate checked out in step 2 can be used to limit how long the privilege delegation will be. The problem is that, in Step 2, the private key cannot be checked out in the current MyProxy implementation. Additional work is needed to support checkout of private key plus proxy certificate. In both of the two OAuth-based authorization architectures described above, the initial privilege delegation is cumbersome and not user-friendly. Further study is needed on how to simplify and automate the process.

All of our OAuth investigation was done separately from PolarGrid Portal. It will be integrated into PolarGrid portal in future work. Additionally, authorization granularity and delegation revocation are two problems we are working on.

Finally, integrating OpenID and OAuth is helpful to improve usability. Users can log in portal using their existing OpenID and authorize the portal to access their data stored at external services using OAuth-based approach. This has not been implemented in our current code base and is part of future work.

5. CONCLUSIONS

Science gateways have long been built out of reusable components and services, but we believe the standards used need to be reevaluated. We also believe that previous science gateways have not provided enough collaboration and social networking capabilities. We further have asserted here that collaborative service sustainability (not to be confused with software sustainability) is a key issue when building these systems. Our goal is to build a system to test some of these ideas.

Our methodology is to utilize lightweight open standards and make use of existing third-party services instead of building our own. Use of open standards improves interoperability so that we are not locked to a specific implementation. Delegating some components to third-party services makes agile development possible and improves sustainability. We have chosen to investigate REST, Web 2.0, Gadget and OpenSocial. REST makes our gadgets able to access backend services easily without incurring complex message manipulation operations (compared with SOAP). The gadget specification is used to build modular web components (gadgets) that can be deployed to any standard-compliant gadget container. Numerous existing gadgets in iGoogle and Orkut developed by programmers across the world can be used by our gateway easily. This is a notable improvement over the older portlet component model, which required server-side (rather than client-side) integration and which never developed the extensive library of standard and portable component that was anticipated. Also OpenSocial makes it possible for our portal to access users' social data in existing social networking services. The third-party services used in our current implementation include Google Picasa, Google Calendar and Twitter. They are used to save filter execution results, filter events and filter execution status notifications.

We reiterate that our goal is to investigate applications of existing technologies and to build a generic open portal platform instead of a closed one. Considering development time of PolarGrid portal and the accomplishments, we believe our investigation and trial is successful. However, there are problems, especially security integration that will require further work.

6. FUTURE WORK

The system described in this paper was an evaluation prototype to explore different technologies and how they work together. The current portal illustrates capabilities, but we must align them more carefully with specific PolarGrid user requirements. We will work to enhance user experience with the Layout Manager and improve gadget directory support using Sling [26] or Jackrabbit [27] to leverage the versioning and comments capabilities supported by these API's.

For authentication, we will build generic identity integration service described in Figure 4, which will provide an authentication service to both the portal and other external applications. For authorization, we will first integrate two-legged and three-legged OAuth into the portal and integrate it with GSI security. OAuth will be used to secure both portal data, such as layout data and user information, and RESTful services. Currently, our PolarGrid RESTful service does not have any built-in security mechanism. We will evaluate OAuth for protecting our developed RESTful service and investigate how to use it to protect arbitrary RESTful services in a general and noninvasive way. Further, authorization granularity and delegation revocation will be studied carefully, and OAuth extensions will be proposed to solve these problems in a flexible and scalable way. Finally, integration of OpenID and OAuth will be implemented to make the whole process more user-friendly.

Inter-gadget communication is also part of our future work. There exist different intercommunication patterns. One pattern is direct communication in which a logical communication channel between two gadgets is established so that one gadget can send messages to and receives messages from the other gadget directly. Another pattern is publish/subscribe. A 'publisher' gadget publishes its changes, and 'subscriber' gadgets that declare

interest in those changes get notified. One possible application is workflow submission and monitoring. A 'publisher' gadget can be used to submit workflows and the corresponding 'subscriber' workflow status-monitoring gadget is notified when the user submits a new workflow through the 'publisher' gadget. One foreseeable obstacle is unique identification of gadgets. Simply UUID can be generated automatically for each gadget. However, UUID is not human-friendly. Trade-off between rigorousness and convenience needs to be made.

7. ACKNOWLEDGMENTS

The National Science Foundation supports this work through awards 0721656, "SDCI NMI Improvement: Open Grid Computing Environments Software for Science Gateways" and 0504075, "SCI: TeraGrid Resource Partners: Indiana University." The Polar Grid project is funded by NSF award 0723054, "MRI: Acquisition of PolarGrid: Cyberinfrastructure for Polar Science". We thank Jun Wang for developing Matlab image processing filters that are used by some of our frontend gadgets.

8. REFERENCES

[1] Wilkins-Diehr, N., Gannon, D., Klimeck, G., Oster, S., and Pamidighantam, S. 2008. TeraGrid Science Gateways and Their Impact on Science. IEEE Computer 41(11): 32-41

[2] TeraGrid Science Gateways [Online] http://www.TeraGrid.org/gateways/gateway_list.php

[3] TeraGrid user portal [Online] https://portal.TeraGrid.org/gridsphere/gridsphere

[4] Lead Portal [Online] https://portal.leadproject.org/gridsphere/gridsphere

[5] OpenSocial Specification [Online] http://www.opensocial.org/Technical-Resources

[6] PolarGrid Project [Online] http://www.PolarGrid.org/PolarGrid/index.php/Main_Page

[7] OGCE Project [Online] http://www.collab-ogce.org/ogce/index.php/Main_Page

[8] Generic Factory toolkit [Online] http://www.collab-ogce.org/ogce/index.php/GFAC

[9] Google Data Protocol [Online] http://code.Google.com/apis/gdata/

[10] Google Friend Connect [Online] http://www.Google.com/friendconnect/

[11] Pierce, M. E., Fox, G., Yuan, H., and Deng, Y. 2006. Cyberinfrastructure and Web 2.0. "High Performance Computing and Grids in Action" (L. Grandinetti Editor) published by IOS Press, Amsterdam, as the volume no 16 in the series "Advances in Parallel Computing", Proceedings of HPC2006 July 4 2006 Cetraro Italy

[12] Gadget Specification [Online] http://code.Google.com/apis/gadgets/docs/spec.html

[13] OGCE Gadget Container [Online] http://www.collab-ogce.org/ogce/index.php/OGCE_Gadget_Container

[14] OpenID Authentication 1.1 [Online] http://openid.net/specs/openid-authentication-1_1.html

[15] OpenID Simple Registration Extension 1.0 [Online] http://openid.net/specs/openid-simple-registration-extension-1_0.html.

[16] OpenID Attribute Exchange 1.0 - Final [Online] http://openid.net/specs/openid-attribute-exchange-1_0.html.

[17] Butler, R., Welch, V., Engert, D., Foster, I., Tuecke, S., Volmer, J., and Kesselman, C. 2000. A National-Scale Authentication Infrastructure. IEEE Computer 33(12): 60-66 (2000)

[18] Novotny, J., Tuecke, S., and Welch, V. 2001. An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001, pages 104-111.

[19] Basney, J., Humphrey, M., and Welch, V. 2005. The MyProxy Online Credential Repository. Software: Practice and Experience, Volume 35, Issue 9, July 2005, pages 801-816.

[20] Welch, V., Barlow, J., Basney, J., Marcusiu, D., and Wilkins-Diehr, N. 2007. A AAAA model to support science gateways with community accounts. Concurrency and Computation: Practice and Experience 19(6): 893-904 (2007)

[21] Pearlman, L., Welch, V., Foster, I., Kesselman, C., and Tuecke, S. 2002. A Community Authorization Service for Group Collaboration. Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.

[22] Pearlman, L., Kesselman, C., Welch, V., Foster, I., and Tuecke, S. 2003. The Community Authorization Service: Status and Future, CHEP03, March 24-28, 2003, La Jolla, California

[23] Thompson, M., et al. 1999. Certificate-based Access Control for Widely Distributed Resources. In Proc. 8th UsenixSecurity Symposium. 1999.

[24] Chadwick, D. W. and Otenko, A. 2003. The PERMIS X.509 role based privilege management infrastructure. Future Generation Comp. Syst. 19(2), pp. 27-289, 2003.

[25] OAuth Core 1.0. [Online] http://oauth.net/core/1.0

[26] Sling [Online] http://sling.apache.org/site/index.html

[27] Jackrabbit [Online] http://jackrabbit.apache.org/