Developing Standards Based Cloud Clients

Michael Behrens, R2AD, LLC
David Moolenaar, R2AD, LLC

ABSTRACT

The computing industry is experiencing a shift in the way we use computers at home, at work, and especially in-between as we have handheld mobile computing devices which also work as telephones. This paper explores how client devices can use cloud oriented software. One area covered is cloud management and the other is cloud based applications or Software as a Service (SaaS). Recent specifications have been produced in the cloud standards space, namely Open Cloud Computing Infrastructure (OCCI) and Cloud Data Management Interface (CDMI). This paper explores how cloud based clients and also servers can be built using emerging open cloud standards. OCCI is a trademark of the Open Grid Forum (OGF).

INTRODUCTION

Cloud management and storage specifications can be used by clients such as web applications, phone, and other mobile computing devices. This chapter explores how client devices can use standards to interact with cloud services.

This chapter examines in detail a JavaFX client which interacts with OCCI and CDMI servers in order to manage cloud resources (computers, memory, etc.) and also storage (raw storage as well as objects).

The project code produced by R2AD, LLC is available as open source, released under the BSD license model. The code was developed with interactions with demo OCCI and CMDI servers which were implemented to process requests compliant with those specifications. A Google project is available with details on the demonstration systems: http://groups.google.com/group/cloud-demo. Visit this group for the latest information about the available code and demonstration servers and also to participate!

The R2AD open source cloud client described in the following sections can be used to learn about cloud services in general and to the code can be used to build more extensive projects that could be targeted to end users or administers. R2AD is current working to port some of the functionality to an Android.

BACKGROUND

The computing industry is experiencing a huge shift in the way we use computers at home, at work, and especially in-between as we use hand-held mobile computing devices. The phone is now a computer that is part of the internet and is a first class citizen of the web. The mobile computing potential motivated the team to work with standards to build a client capable of running in the web and on mobile phones.

Traditional models of client software development included stand-alone applications and client-server applications. Following this is the distributed computing model in which there are n-tiers of servers and or clients. Peer-2-Peer is another which could be considered a variation of client-server computing. However with

the concept of a cloud, we now might consider client-cloud based applications as a form of distributed computing. In this case, the client is interacting with an endpoint which represents a large server side potential. The "one to many" cloud connections enable scalability through parallelism.

Starting with Cloud Standards

Our team worked within the Open Grid Forum (OGF) and the Storage Networking Industry Association (SNIA) standards organizations to contribute to the specifications. This provided:

- An understanding of open cloud specifications
- Team building toward a common demonstration

We shall continue to work with these and other organizations such as the Distributed Management Task Force (DMTF).

A number of attempts are being made to standardize access to the cloud by many groups. Various efforts are underway by standard development organizations such as the Open Grid Forum (OGF), Storage Networking Industry Association (SNIA), Distributed Management Task Force (DMTF), National Institute of Standards and Technology (NIST), and others. In our case, we have focused on two specifications: Open Cloud Computing Infrastructure (OCCI) and Cloud Data Management Interface (CDMI).

The OCCI specification is developed by the Open Grid Forum (OGF) Standards Development Organization (SDO). OCCI is a boundary protocol/API that acts as a service front-end to your current internal infrastructure management framework (IMF). The following diagram shows OCCI's place in the communications chain.

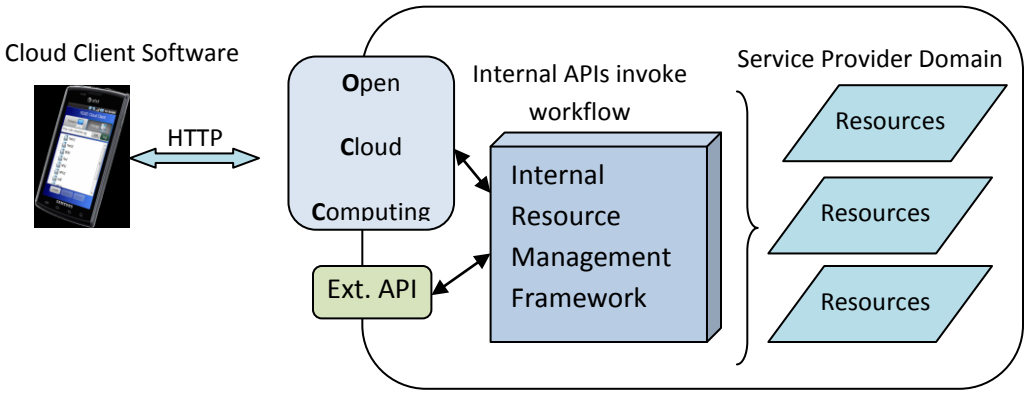


Figure 1 OCCI as a Boundary API – from the OCCI Core Specification from the OGF

OCCI consists of a set of specifications (Core and models, HTTP header rendering, Infrastructure models, and other rendering). These documents, while written to address a standard interface for cloud computing, are in fact directly applicable to any distributed computing architecture. They represent the "State of the Art" in terms of internet computing today. They support Rich Internet Application Development, Service Oriented computing, and offer a scalable and dynamic approach to creating semantic oriented services.

The OCCI specifications refer to this architecture as a Resource Oriented Architecture (ROA). It defines RESTful interfaces based on the Hypertext Transfer Protocol (HTTP). Each resource (a computer, or storage element, etc.) is identified by a Uniform Resource Identifier (URI). One or more "representations" of that resource can be

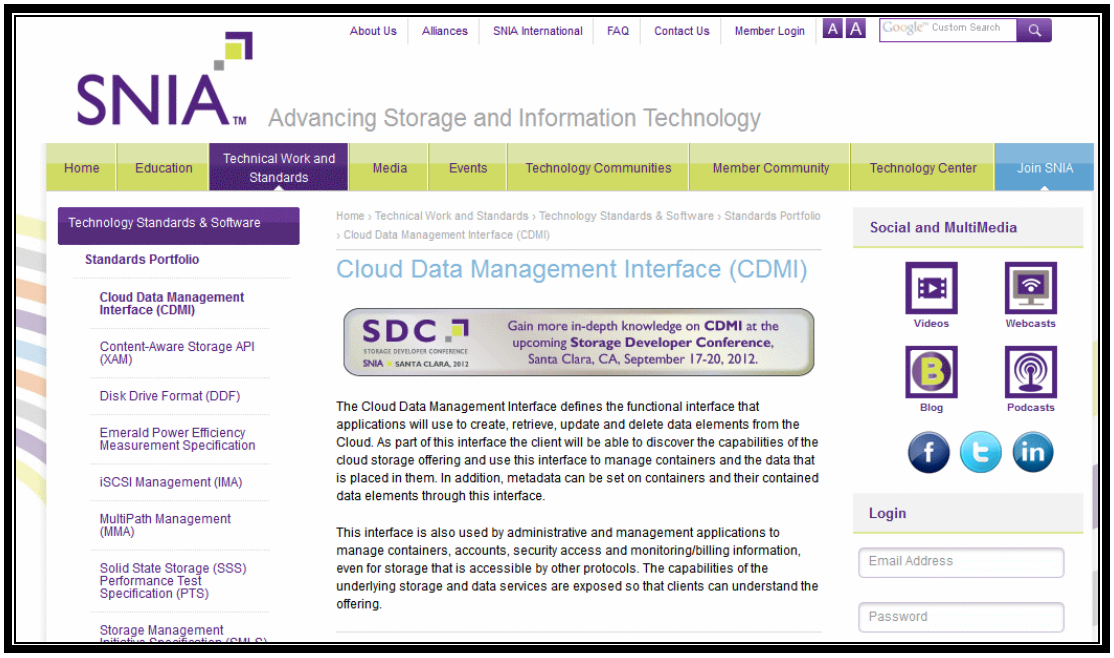
requested using the HTTP "GET" verb. The PUT and POST verbs in HTTP can be used to create and update resources. Associations of resources can be conveyed in the HTTP headers, using the link tags.

OCCI defines a core and foundation specification upon which a complete Infrastructure as a Service (IaaS) services can be built. Following this pattern, additional layers of the specification can be added to provide support for Platform as a Service (PaaS) and Software as a Service (SaaS).

The Cloud Data Management Interface (CDMI) is developed by Storage Network Industry Association (SNIA). It also provides a RESTful API set to manage storage in the cloud. CDMI provides cloud data management in support of both public and private storage. In addition to providing management of storage, it also provides a means to create objects within containers, providing the building blocks necessary for any basic application that needs to store data. CDMI is the functional interface that applications may use to create, retrieve, update, and delete data elements from the cloud. It also supports the addition of metadata which can be used to tag data within containers. CDMI uses many different types of metadata, including HTTP metadata, data system metadata, user metadata, and storage system metadata.

From a storage access perspective, CDMI also supports various standard protocols such as CIFS, NFS, iSCSI, and its own standardized RESTful HTTP data path. From a client perspective, basic HTTP GET and PUT requests can be issued and the resultant JavaScript Object Notation (JSON) message bodies can be used within a browser context or parsed by other clients. Together with client side software, the CDMI specification can be used to provide cloud services such as cloud2cloud (C2C) migration, data backup services, searches, and application storage.

CDMI became an ISO standard in October of 2012.



Cloud Client Implementation

Using both specifications together, our team has developed a basic open source cloud management client which demonstrations the core functionality of both the OCCI and the CDMI specification.

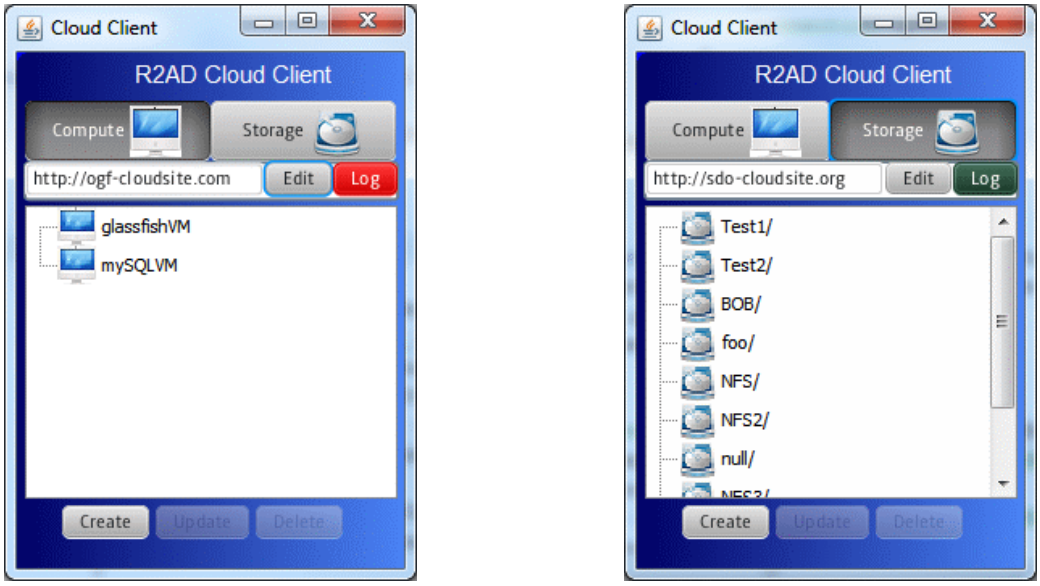


Figure 2 - Compute Resources and Storage Resources

Following a use-case driven development process, we first used OCCI to access a list of cloud compute nodes. Then, using CDMI, we access a set of storage resources. Using both together, we've essentially created a mash-up of the two specifications that allow the creation of compute resources that reference the CDMI storage. The figure shows the compute tab with the sample virtual machines that are residing on the OCCI controlled servers.

Also shown is the storage tab which is listing the available storage containers that the CDMI cloud service exposes. These storage end points can be used when defining a virtual machine. Each of these can contain file objects, metadata, or other elements which can be independently managed through the RESTful services.

This example illustrates a simple cloud management client that utilizes specifications developed by standards organizations. Ultimately, if the major public and private cloud producers included support for these standards, client applications could worry less about interoperability and more about value added features for administrators and users.

The overall design of the application included a login component, a presentation layer, and parsers for both XML and JSON documents that are received from the cloud based services.

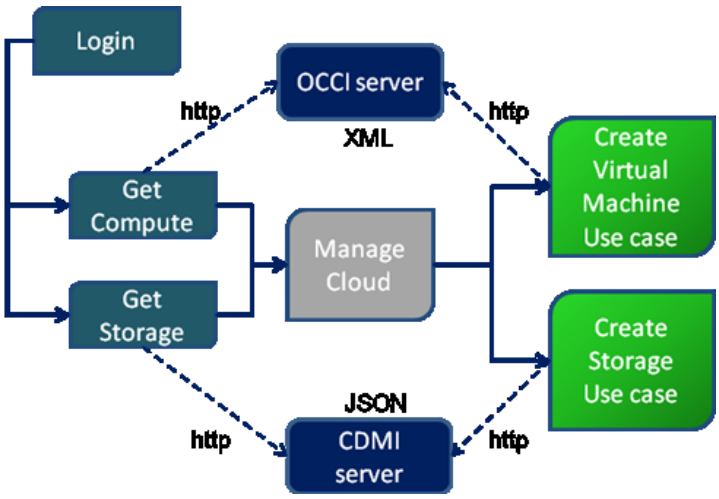


Figure 3 - Cloud Management Design

The figure above shows the service interactions. Before the GET requests can be made, the login credentials are required. Note that in our current version, only the OCCI required credentials. Each request passed the credentials in the header, as per the specification. Results were returned in either JSON or XML and were parsed accordingly. Keep in mind that this client acts as a front-end to the cloud and the back-end processing which could be implemented in a number of ways. In our demonstration, an Open-Nebula server created the VMs and the CDMI Reference Implementation was used to provide the storage service.

INSTALLATION AND DEPLOYMENT GUIDE

The source code to the client is available from http://github.com/r2ad. The code can be viewed on-line as needed or downloaded using a git client. To download all the code into a current directory, the following command can be used (enter in what is in bold):

> *E:\gitcloudclient>**git clone http://github.com/r2ad/R2AD-Cloud-Client.git***
> *Initialized empty Git repository in /cygdrive/e/behrens/programs/CloudStudy/gitcloudclient/R2AD-Cloud-Client/.git/*
> *remote: Counting objects: 215, done.*
> *remote: Compressing objects: 100% (163/163), done.*
> *remote: Total 215 (delta 92), reused 92 (delta 27)Receiving objects:*
> *Receiving objects: 100% (215/215), 154.04 KiB, done.*
> *Resolving deltas: 100% (92/92), done.*

NOTE: In the example case above, git is a Windows executable from the Cygwin library of tools available from http://www.cygwin.com. The git executable is also available for Linux and other operating systems. For more information about git, please visit: http://git-scm.com. For those who are new to git, the following URL provides a good Git user's manual: http://www.kernel.org/pub/software/scm/git/docs/user-manual.html. Note that for

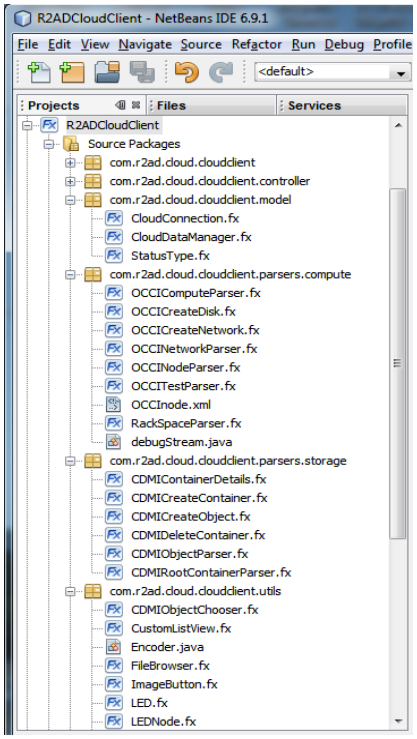
first time git users, the "**git init**" and "**git config –global user.name**" commands should be used to initialize and configure git. Your e-mail can also be provided, if needed using the config command.

Once the code is checked out, install the NetBeans development environment. This is required mostly since the JavaFX Framework used for this client is well supported with NetBeans. As of the writing of this chapter, according the NetBeans web site (http://netbeans.org), the latest version is 6.9.1. There are several bundles that are available, be sure to choose one that supports JavaFX or choose the complete bundle which also includes support for Web Server development.

While JavaFX and Java are related, there are enough differences that it is recommended to carefully study JavaFX to be aware of its key script-like features such as timelines, styles, functions, and variable declarations. Visit http://javafx.com for more information. Some of our developers took the on-line JavaFX course available from http://www.javapassion.com/portal/ and have also taken the Android course as well.

Run the Code using NetBeans

Launch NetBeans and import the project by selecting **File** | **Open Project**. Navigate to the R2AD-Cloud-Client directory where the source is checked out and the click the **Open Project** button. Once the project is open, use the tree on the left hand side to expand each of the source code packages and become familiarized with the layout of the packages. The main packages are as follows:



com.r2ad.cloud.cloudclient.parsers.compute

Provides XML parsers for the sample beta OCCI server. Note that while this code and the parsers are working with XML, the newer OCCI specification (described later) have multiple renderings including HTML. Updated OCCI servers would not be expected to work with the parsers provided at this time, however future work is planned to update the client code.

com.r2ad.cloud.cloudclient.parsers.storage

Provides the Java Script Object Notation (JSON) parsers for a CDMI server (described later). These parser handle basic container manipulation and can be expanded to support the rich set of CDMI capabilities.

com.r2ad.cloud.cloudclient.view

Contains the presentation code for the various dialogs presented to the user.

com.r2ad.cloud.cloudclient.controller

Manages all access to the data and in particular handles the CRUD operations.

To execute the code, right click on the FX project in the tree and select **Run Project**. When the application first starts up, two tabs are presented, one for Storage and one for Compute. For each tab, the Edit button must be used to enter in the URL and credentials for the server.

Initial Startup Dialog Login Screen

Figure 5 – Starting dialog when executing from NetBeans

The checkbox on the login screen can be used to save the information and the blur info checkbox can be used to obfuscate the values. The latter is useful when giving demos where the login information is not safe to share.

When running in this mode, the application is executing as a regular Java[1] program. Using NetBeans, it is possible to change the mode to make the program run as a web application, that is as an Applet, or as a Java Web Start application using the Java Network Launching Protocol (JNLP) protocol. To see a live example of the project running, visit the http://clouds.r2ad.net web site and click the Launch button. *NOTE: The ability to launch JavaFX modules may be temporarily disabled until the code is updated.*

As an example, select the Storage tab and follow the steps below to define a new CDMI container. Note that since we are not really connected to a server at this time, it will simulate the creation of the container.

- Click the **Create** button.
- Enter in a name, for example, "store-a"
- Click the **Upload** button and select a file such as Object.txt.
- Check the **CDMI Content Type** check box. The CDMI spec supports the CDMI content type and also other MIME types. This version of the client only supports the CDMI Content type as this time
- Click the **Create** button.
- Observe the debug output in NetBeans. Look for the line that states:
  - CDMICreateObject: file content: This is an ASCII Object.
  - CDMICreateObject: Object Request Aborted...zero content size!

Testing and Debugging the Client

[1] Java is a trademark Oracle, previously Sun Microsystems.

REST based services can be easily monitored when developing a client against them. In our case we used a number of different tools, to include the FireFox plug RESTTest and the Java TCP monitoring tool tcpmon. The latter tool can be run from within the NetBeans IDE or separately as a Java command line tool or as a Java WebStart application directly from its homepage. An example of using tcpmon to monitor communication with the CDMI server is shown in figure 6.

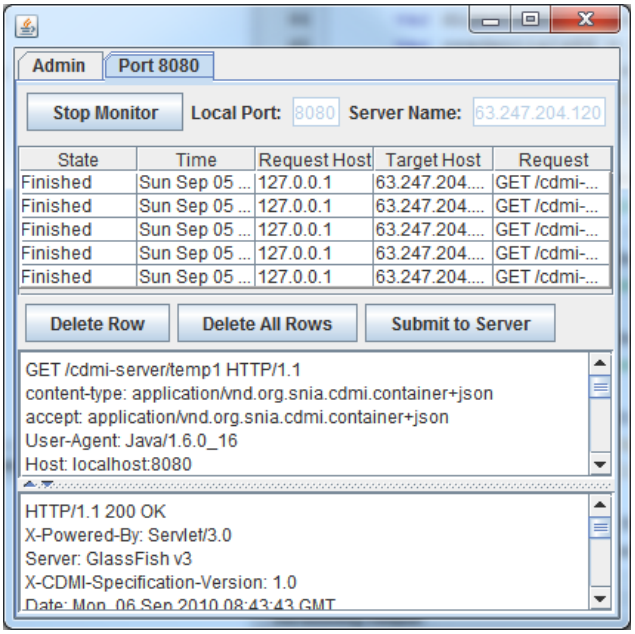


Figure 3 - Using tcpmon to capture traffic

To monitor the transactions, start the tcpmon program. This will relay the requests to localhost to the actual CDMI server IP for the real server.

- Open a command window and startup the tcpmon program and configure it to connect to the CDMI server IP.
- Alternatively, use the provided tcpmonstart.cmd and pass in the IP address and port number (default port is 8080 if not provided)

Use tcpmon to follow the web service PUTs and GETs when trying to upload a document.

The parsers are probably of the most interest. Let's examine the CDMICreateContainer class in detail. The *createContainer* method is invoked when the create button is pressed from the Storage Editor View. It starts by building a JSON object which it then sends to the server using the HTTP PUT verb. The method then processes the response, which is also a JSON object. JavaFX (and Java as well), provides an HttpRequest class which is used to post the request.

The mime type and version information are placed into the header, in compliance with the specification:

```
headers: [contentHeader, acceptHeader, versionHeader]
method: HttpRequest.PUT
```

The actual message being sent is stored as a StringBuffer and its bytes are sent to the server by overloading the output variable as shown below:

```
override var output on replace {
  if (output != null) {
    try {
      output.write(content.toString().getBytes());
      output.close();
    } catch (e: java.io.IOException) {
      println("{myName}: Unable to write JSON content");
    }
  }
}
```

The response that comes back is processed by the internal function *processResults*. It expects a JSON object to be returned, and it uses a PullParser to analyze the message. Once an END_ARRAY_ELEMENT is received, the newly created CDMI container is then stored in the model with a call to the controller which manages all transactions to the internal data cache:

```
controller.dataManager.addStorageType(result);
```

Each of the user facing views extend the AppView class. AppView is a custom node in JavaFX and provides convenience functions for all views add to the scene such as the showProgress, showMessage, and deleteConfirmation methods.

Lessons Learned

A number of lessons were learned during the development and testing of the client, namely:

1. The network is not always present. Developing a cache of recently access data would be useful. A recommendation for cloud storage standards would be to add support for cache synchronization.
2. Remote administration tools were valuable. These included the obvious tools such as SSH access, but also included tools with provided full screen remote desktop access such as Oracle's Secure Global Desktop as implemented as part of R2AD's Teleclient® solution.
3. Developing within a cloud environment should take advantage of snapshots. Also, developing in the cloud meant that the development experience could be shared with other network participants.
4. Deciding to make the code open source before the development cycle would have saved some code editing time later on.
5. Working with the standards group is a great way to both participate in the process and to learn the specification details.

Security of the cloud, and for that matter any application, is always important to consider. The client we developed used basic authentication for each request. For the OCCI interface, a base 64 encoded MD5 hash on a SHA1 signed value provided to the server for validation. Since our implementation was only a demonstration, it did not use HTTPS, however that would be used in production systems. OCCI and CDMI require the credentials to be sent and validated for every request. For added security, other cloud based APIs have implemented different methods which include the use of a session key and a URI path which is unique to the session.

Figure 4 - Obfuscation of Credentials

Authorization is performed on the cloud side for each request. How this is managed and implemented is left up the implementer of the specification.

On the client side, credentials were cached, however not in an encrypted manner as they would be in a final product. We did however implement a credential obfuscation mechanism which allowed the system to be demonstrated while not revealing any account information.

The prototype code was developed using JavaFX using Netbeans 6.9.1. However in retrospect, using the Android or iPhone platform would have been a better choice as it would be easier to make available to a growing marketplace.

FUTURE RESEARCH DIRECTIONS

Consider the mobile computer user. The small footprint of modern phones today is able to execute a variety of applications and access the Internet. They have the necessary tooling which enables them to access the internet, store data locally, and interact with a number of external devices. They are great candidates for cloud clients since their storage and computational powers are limited. They are helping to push the envelope of what cloud computing means to everyday users. The mobile application market is already taking advantage of the cloud services offered by companies such as Google, Amazon, and others.

The client device is acting as a cloud terminal in many cases. For example, instead of storing addresses only locally on the phone, some applications let the cloud store that information, using the local storage as a data cache. Someday, these applications may well use specifications like CDMI, allowing the client code to remain relatively constant, regardless of the actual service provider.

Another example is how imagery (video and photos) can be stored and accessed. Also, from a High Performance Computing (HPC) viewpoint, image processing is a great service that enables tools like bar-code lookup or image recognition. Soon there will be many other client applications that use the cloud services such as voice translators and on-line dictation services.

The OCCI specification is considering the standardization of the console to the virtual computing resource. During the development of the cloud client, we used remote desktop access mechanism, as previously mentioned. While this is important for administrators, this capability is equally valuable to end-users. Many users are using the mobile computing devices more often than their home PCs. Providing these users with screen access to virtual PCs is a natural extension. These cloud based PCs shall dramatically change the role of the heavy desktop. Devices to access Cloud PCs would instead dominate the end-user marketplace and would allow users to access their desktops from many places to include their home HDTV televisions.

More and more users are mobile computing base and therefore the number of client clients shall increase substantially over time. We shall continue our research using the Android platform since it has excellent support for application development and internet access. It acts as a great cloud client and through its internal storage mechanisms, can cache data locally for off-line viewing. In turn, the client will need to interact with many more cloud services to include data storage and request processing.

We look forward to porting our application to the Android and publishing it in the marketplace!

CONCLUSION

Cloud standards are available now for use by integrators, major software vendors, academia, and government institutions. While the space is always going to evolve, there are clear indications that the specifications used in this section are ready to for wide scale adoption. This will help maximize the return on investment that we all make, to include the energy it takes to become familiar and productive with an Application Programmers Interface (API).

The CDMI and OCCI specifications shall help standardized the cloud computing world. Based on the recent level of activity in the standards organizations centered on cloud computing, the time seems right to rally the industry around standards that will help reduce the proliferation of APIs and give a common ground for all providers to implement, adding their own value differentiation.

Overall, cloud computing changes the assumptions made earlier about client-server applications in that the connections from the client are “one to many” instead of one client to one server. This facet alone offers a level of scalability which is changing the perception of what a PC is.

Lastly, it is important to support the standards in manner in order to ensure the best of breed become adopted by to promote adoption

REFERENCES

OCCI

Open Cloud Computing Infrastructure (OCCI), *http://forge.ogf.org/sf/go/projects.occi-wg/wiki*.

SNIA CDMI

Storage Networking Industry Association, "Cloud Data Management Interface (CDMI)", 2010 [Online]. Available: *http://snia.org/cloud*
http://www.snia.org/cdmi

REST

Roy Fielding, Representational State Transfer (REST), http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

R2AD Source Web Site:

R2AD Cloud Client Source Code, Available: *http://clouds.r2ad.net*.

REST Test

FireFox RESTtest plugin, Version 1.4, https://addons.mozilla.org/en-US/firefox/addon/5946/

TCP Monitor

Java TCP Monitoring tool, tcpmon version 1.1, https://tcpmon.dev.java.net/