An Infrastructure for Teaching CS1 in the Cloud

Michael Woods[1], Godmar Back[2], and Stephen Edwards[3]

Abstract – A key goal of entry level computer science classes is the recruitment and retention of potential computer science graduates. Many entry level classes struggle to show students the real world context and social and societal impact of computer science. Modern Web 2.0 applications provide an attractive way of putting computer science in a realistic context. However, efforts to teach such applications in a first CS course are hampered by the fact that they require a deep understanding of multiple and complex concepts and technologies. Rather than focusing on learning core computer science concepts, students would need to learn transient and technology-specific knowledge.

This paper presents the initial design of CloudSpace, a teaching environment that allows students to understand core computer science concepts in the context of modern web applications. CloudSpace presents entry-level CS students with a virtual environment to develop web applications in a manner similar to the development process of desktop user interface applications. For example, CloudSpace allows students to work with abstractions such as a console and or a file system that mimic the environment in which traditional programs run. This virtual environment also shields students from the distributed nature of the web technologies with which they work.

To populate these virtual environments with real-world data, CloudSpace will be equipped with API's to access data from the Internet. CloudSpace will feature a rich library for accessing RSS feeds, on-line video clips, on-line CSV data sources, and generic web page content. Rather than working with dull datasets, this library allows students to integrate real-world and life data sources into their applications.

Keywords: CloudSpace, CS1, Cloud Computing, and Virtual Machines

INTRODUCTION

CloudSpace provides students with an easily accessible web development environment. Students can develop applications, deploy and manage them and share them with classmates and friends, as shown in Figure 1. CloudSpace provides virtual machines, virtual file systems and a persistence framework for student applications. A CloudSpace prototype has been in use in CS1 courses at Virginia Tech for two semesters. This paper describes the model underlying CloudSpace and its ongoing implementation.

[1] Virginia Tech, 114 McBryde Hall (0106), Blacksburg VA 24061, mjw87@vt.edu

[2] Virginia Tech, 114 McBryde Hall (0106), Blacksburg VA 24061, gback@cs.vt.edu

[3] Virginia Tech, 114 McBryde Hall (0106), Blacksburg VA 24061, edwards@cs.vt.edu

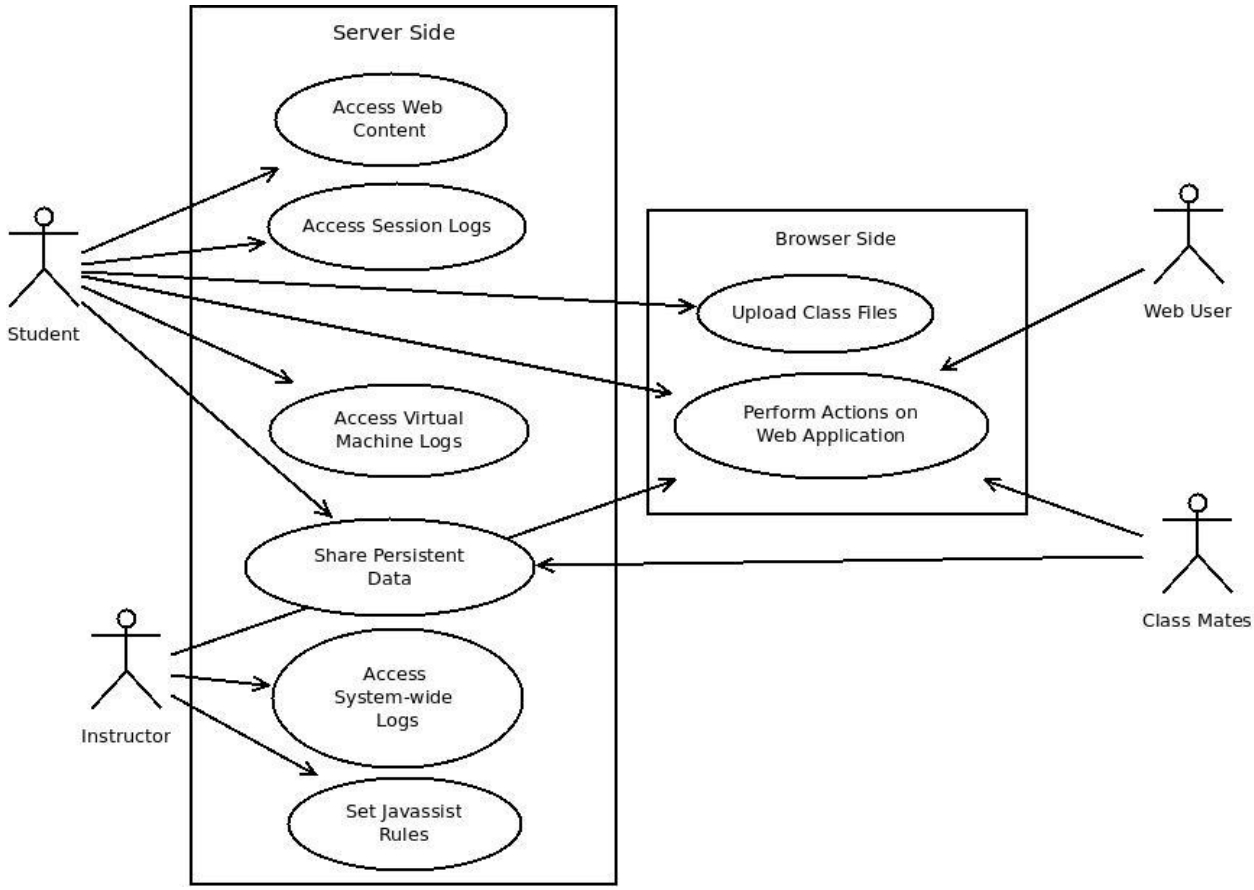

Figure 1: CloudSpace Roles

MOTIVATION

As novice computer science students take their initial computing courses, they are introduced to such basic software concepts as data and control flow structures. However, the stark difference between what is taught in the initial curriculum and what they experience when using modern web-based software leads many students to doubt the practicality of their education [Beaubouef and Mason, 8]. CloudSpace aims to bridge this divide by providing a toolset and environment for teaching Computer Science in a web development that is accessible to novice computer science students.

Distributed Nature of the Web

The World Wide Web is a distributed system based on the client-server paradigm. Most web development tools make no effort to hide its distributed nature from the developer. Web applications must be able to serve multiple clients simultaneously, which requires synchronization when concurrent accesses to shared data occur. Most tools require the developer to ensure this synchronization, but techniques for synchronization are not and should not be part of the curriculum of most CS1 courses. Instead, tools provided to CS1 students must protect their data from concurrent access. If concurrent access is unavoidable, the tools must resolve conflicts in a predictable way to prevent confusion and potentially educate the student about the concurrent access that is occurring. If a tool handles concurrent access to data, students can focus on core computer science concepts first.

Providing an Environment

As classroom experience has shown, students want their work to be usable and shareable. If their work merely passes unit tests and never impacts anyone's life, it is difficult for them to understand the impact of the field of computer science. With application servers backing the projects that students create, their web applications can be immediately shared and directly used by family members and friends. This type of positive reinforcement shows students that the skills they can learn through computer science have real-world impact.

TECHNICAL BACKGROUND

ZK Framework

CloudSpace is based on a heavily modified distribution of ZK [7]. The ZK framework offers a professional level web development environment to students. One advantage in using ZK is its support for dynamic data binding in HTML pages. This allows students to bind data stored in classes they have written to the HTML page that is displayed upon request. If the data stored in their class is updated, ZK will automatically communicate the new data to the client and the HTML page will be updated to reflect the changed data. Providing dynamic data binding does not require students to learn technologies such as AJAX and JavaScript.

The ZK lifecycle begins with HTTP requests initiated from a client's browser. An HTTP request is associated with both a request id and a URL. The URL is then translated into a page located on the web server and the page is delivered to the client. In the standard ZK distribution, dynamically bound data elements are updated only when specified browser side events such as button pushes occur. CloudSpace automatically sends updates to the client upon any change in the bound data.

ZK also allows Java code to be embedded into HTML pages, which is interpreted using a bean shell interpreter. The bean shell interpreter creates a namespace for each HTTP request. Using ZK, Java objects can be defined an initialized and Java statements can be associated with browser side events, such as pushing a button or checking a checkbox. Embedding Java statements allows students to interact with Java classes using the techniques they are learning.

THE CLOUDSPACE MODEL

Virtual Machine

CloudSpace adds a virtual machine framework to ZK. Each HTTP request handled by ZK is associated with a virtual machine that corresponds to the directory structure in the URL. Virtual machines are responsible for recording information printed to the system's console and managing the lifecycle of student Java classes. They are intended to mimic a traditional Java execution environment as closely as possible.

Java provides developers with two console print streams for displaying text to the system's console, System.out and System.err. Both of these streams' output is usually written to the system's console. In a traditional web environment, write operations to these streams are captured and written to shared log files. In many shared hosting environments, access to these shared logs is not possible, or the logs would present a confusing combination of output created by the web application server and other student's web applications.

To emulate a console that resembles the ones provided in Java IDE's such as Blue J[3] and Eclipse[4], CloudSpace provides a special HTML tag for embedding a console window into their web application, as seen in Figure 2. Students can choose to see output resulting from only the current web application instance, or from all currently active instances, each of which corresponds to a unique visitor to their application. The console is dynamically updated to contain all lines printed to System.out or System.err in all active web application instances.

Figure 2: Example of the Embedded Console

The virtual machine also manages the classes used when handling HTTP requests. Students upload their classes to the server using a minimalistic web interface. The virtual machine mapped to that student's directory is responsible for detecting new uploaded class files. When new HTTP requests are received, the virtual machine coordinates with ZK to ensure that new classes are loaded if required.

File System

CloudSpace provides a virtual file system on the web server in which students can create files using absolute and relative paths. The virtual file system provides students with access to a virtual Unix like root directory in which to store their web application resources. Figure 3 shows an example of a CloudSpace virtual file system call compared to a file system call in a traditional environment.

```
//Virtual file system call
File foo = new File("/bar.txt");
assert(foo.exists(), true);
```

```
//Traditional file system call
File foo = new File("/tomcat/webapps/CloudSpace/bar.txt");
assert(foo.exists(), true);
```

Figure 3: CloudSpace virtual file system avoids the use of absolute paths that would be required in a standard J2EE environment

Persistence Framework

Many web applications work with data that is persisted in databases between visits. Since database concepts and query languages such as SQL are not usually part of the CS1 curriculum, CloudSpace provides an object-based approach to persistence.

The CloudSpace persistence model provides students with a layer to persist objects they have created for later retrieval. All save operations requested by the student are saved to the persistence layer when the web application is closed. All objects loaded from the persistence layer are frozen for the current web application. By controlling the save and load operations, CloudSpace protects the student from unexpected race conditions when accessing shared objects.

The persistence framework operates using check-in and check-out method calls. When a student wishes to commit data to the persistence layer, they perform a check-in command that uses a key for identification and stores the current data populating the fields in the object. When they wish to retrieve the value later, they request a copy of the data stored in the persistence layer. All data that the student commits to the persistence framework is committed in

serialized form. This way there is no restrictions on the object design they use. Figure 4 shows the use of CloudSpace's persistence API to access persisted objects.

```
public class WebApplication extends Application {
	public class FullName {
		private void firstName;
		private void lastName;
		//constructor getters and setters……..
	}
	public void test() {
		FullName name = new FullName(“John”,”Doe”);
		setSharedObject(“name”, name); //Save object
		//Perform operations and retrieve original
		name = getSharedObject(“name”,FullName.class); //Restore object
	}
}
```

Figure 4: Example of Persistence Framework

Through these check-in and check-out commands, a student can store object between HTTP requests. It is also possible to evolve the classes implemented the persisted object. The persistence layer is able to map new classes to existing stored objects as long as the field names and types of the stored information have not changed. To take the persistence layer a step further, CloudSpace can share objects across different virtual machines. If students follow a similar naming convention for the fields in their classes, they can load and export their persisted classes to other CloudSpace virtual machines. This type of collaboration allows for a unique social aspect to be integrated into CS1 projects. In our Introduction to Software Design class [Edwards, 9], students are assigned a project to create a web application that functions similar to Facebook. In this project, students use the persistence framework to access user profiles created by different students' web applications.

DISTRIBUTION CONCERNS

To facilitate deployment, CloudSpace must easily integrate with existing web servers. To allow the deployment of CloudSpace in any J2EE environment such as Tomcat [1], we avoided relying on any modifications to the Java API classes. Such modifications would interfere with other web applications running on the same server and prevent the deployment of CloudSpace in some environments.

At the same time, we do not wish to require students to use environment-specific, non-standard classes, such as wrapper classes for file system access that translate directory path names used by students to the actual path names in the hosting environment. We accomplish this goal by rewriting the students' code and transparently intercepting file system accesses.

IMPLEMENTING CLOUDSPACE

CloudSpace is implemented as a Tomcat [1] web application based on the ZK RIA framework [7]. It is distributed as a war archive that can be placed into the Tomcat web app directory and automatically loaded into a Java application server.

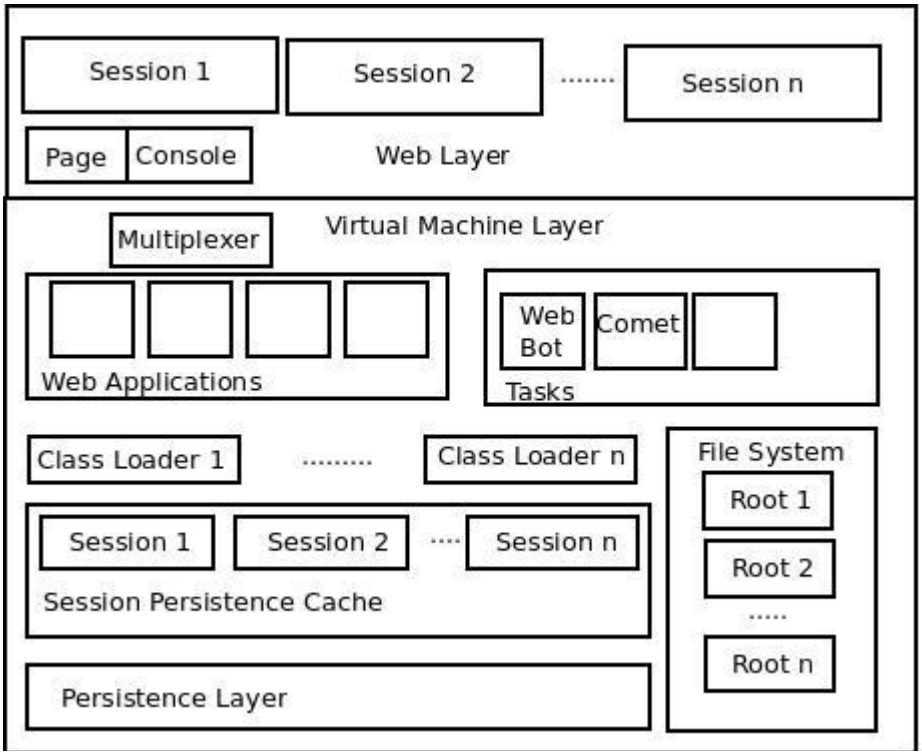

Figure 5: CloudSpace Implementation

The Virtual Machine

The virtual machine forms the backbone of the CloudSpace architecture. It stores state information such as the root directory, the current class loader, all threads associated with the virtual machine, and console output streams. All student code runs in a virtual machine.

A virtual machine is uniquely defined by the root student directory it is operating in. When a page request is received, CloudSpace inspects the URL used to request the page and identifies the root directory. Using the root directory parsed from the URL, CloudSpace performs a lookup in a map to identify the virtual machine associated with the current root directory, creating a new virtual machine if needed. The root directory is used to find student class files and rewrite all paths to files students access. The virtual machine creates an environment for the user in which the root directory parsed from the URL appears as the root of the VM's file system name space.

CloudSpace provides individual class loaders for each web application to separate classes between applications. To update the classes in the class loader, students upload classes to CloudSpace and issue a new HTTP request from their browser. When a new HTTP request is received, the web application's virtual machine will detect changes in the uploaded classes and update the class loader accordingly. Figure 6 shows the class loading scheme implemented by the web application's virtual machine. The Javassist translation step, in which students' bytecode is rewritten, will be discussed in the next section.

All event handling triggered by the web application is routed through a bean shell interpreter[2], which is linked to an immutable class loader by CloudSpace when the page is first requested. When the class loader associated with the virtual machine is updated, already open sessions are not affected. This mimics the class lifecycle in desktop environments, in which class changes are effective only when an application is re-invoked.

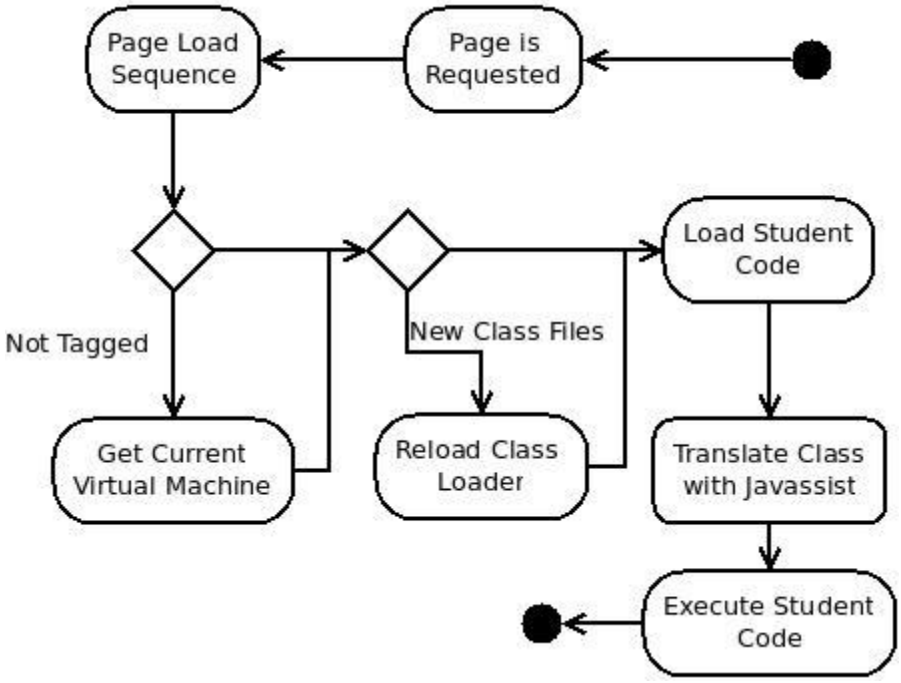

Figure 6: The class loading scheme used in CloudSpace

Adding to the complexity of the virtual machine implementation is ZK's multithreaded nature. Every time an event is created through user interaction with a web page, the server is notified and an event handling thread is assigned the event handling operation. As threads are assigned an event, the events are inspected and threads are assigned a virtual machine according to the page request URL it is operating on. As the event handling thread continues, all code executed will refer to the tagged virtual machine. The thread tag is removed when the thread has completed the execution of the event handler.

Tagging every thread according to its root directory does not allow the identification of individual web sessions. To identify these sessions inside a virtual machine, an additional session identifier is tagged on a thread. The session ID allows for further filtering of events and print operations occurring in the virtual machine based on the executing session.

When the first virtual machine is loaded by CloudSpace, System.out and System.err are replaced with a system multiplexer. The original System.out and System.err print streams are preserved for later use. When a print operation occurs, the multiplexer forwards the output to the current virtual machine's out and err streams, respectively. These messages are stored for shared access by all web applications. Each line captured is tagged with desktop ID, a timestamp, and the line to be printed. After the output is captured by the virtual machine, the original system print stream is passed the printed line, which logs all output for inspection by the instructor.

Environment Transformation

Each student is provided a virtual file system on the CloudSpace server. Since Java does not provide a robust notion of a current working directory, relative paths are resolved relative to this virtual file system's root. This resolution is performed by intercepting file system accesses via wrapper classes, which are invoked in instrumented student code that is dynamically rewritten when it is loaded.

To perform the dynamic rewriting of classes, CloudSpace uses the Javassist library [5]. Javassist is a byte code rewriting framework that allows users to interpret Java class files and rewrite the contained byte code. Bytecode rewriting is used because it can be done transparently after students have successfully compiled their code and does not require any action on the part of the student.

We implemented a configurable instrumentation library based on Javassist for CloudSpace, which allows instructors to define rules to find specific byte code sequences in compiled Java code and specify how the byte code should be rewritten. To configure the byte code rewriting, instructors create a series of rules that define the expressions to be rewritten and the utility functions that should be used in the translation. After the configuration file has been registered with CloudSpace, all classes loaded by the virtual machine class loader will be rewritten. For example, in CloudSpace's default installation, a configuration is included to rewrite all calls to java.io.File methods. Each call to any method of java.io.File is intercepted to change all paths passed from the user to java.io.File to absolute server paths. The configuration file also wraps methods such as getPath() and getAbsolute() path so that the student receives return values that match the expected values.

Configuration rules consist of three parts. The first part, the `define` keyword, indicates the beginning of a configuration line. The second identifier indicates the Javassist configuration class to load using reflection. All Javassist configuration classes must implement a Java interface (Command) that allows them to evaluate an expression and perform a translation. The remaining terms are arguments passed to the new command. The initial arguments are passed as single strings while the last argument may be multiple strings surrounded by quotes.

//Before Translation
File foo = new File("/bar.txt");
	//After Translation
	{
	String rewritePath = toGlobalName("/bar.txt");
	//rewritePath = "/<virtual machine root directory>/bar.txt"
	File foo = new File(rewritePath);
	//The original path is stored for later retrieval,
	//e.g., in getPath()
	logUserPassedString("/bar.txt");
	}

```
define rewritePathAndLog(resultingObject, passedString) << EOM
{
        //Packages removed for brevity
        java.lang.String rewritePath = toGlobalName(%passedString));
        $_ = $proceed(rewritePath);
        logUserPassedString((java.io.File)%resultingObject, %passedString);
}
EOM

translate Replace NewExpr java.io.File(java.lang.String) with << EOC
{
        %rewritePathAndLog($_,$1)
}
EOC
```

Figure 7: Typical Student Code Rewritten by a CloudSpace Rule.

The instrumentation library allows users to define commonly used macros for reuse. Macro names begin with the '%' character while parameters in the macro begin with '$' characters. Macros are substituted in replacement text before being passed to command constructors. The instrumentation library can be reconfigured at run time. With the ability to reconfigure the instrumentation rules while the web server is still running, libraries can be integrated into the CloudSpace virtual machine blueprint without impacting other students using CloudSpace. Figure 7 shows an example of a Javassist configuration command and the resulting code differences.

While the Javassist library is used extensively to translate calls to classes contained in the java.io package, it can be used to rewrite calls to other packages. This allows professors to include other Java libraries that were not originally designed for use in a virtual environment into their projects.

Console Implementation

All output created in a thread tagged with a virtual machine is captured by virtual machine specific data structures. To view captured console output, students add a <Console> HTML tag to their web application.

Each time a student prints to the console, the output is captured by an InfoStream data structure. As lines are appended to InfoStreams, they are tagged with their web session id, or unique web application id, an InfoStream specific line number and a time stamp, which together determine the order in which lines are displayed. If two lines occur in the same InfoStream, their line numbers determine the order of the lines. If two lines occur in different InfoStreams, then their associated time timestamp is used to determine their relative location.

In addition to capturing System.out and System.err write operations, the console supports registering additional user defined log streams. For example, in the current CloudSpace implementation, a dedicated CloudSpace log InfoStream is available for the CloudSpace library to communicate messages to the student about various events, such as when the console was flushed or a class loader was updated.

Persistence Framework

The persistence framework is based on the xstream libraries [6]. When a student commits an object, CloudSpace inspects the object and passes it to the xstream libraries to be converted into XML format. When a student requests the persisted object later in execution, CloudSpace obtains the stored XML object and recovers all of the fields in a new object. Thus, CloudSpace is creating a copy of the persisted object for a student to use.

If concurrent access to persisted objects occurs, the last write to the object is saved. However, if a student has checked out an object, it is stored in the current web session's cache. Therefore, until the current session has finished, the student is working with a cached copy immutable to other web sessions. Delaying updates to the persistence layer prevents concurrency conflicts from occurring while the student is using the web application.

Web Robot Framework

Stevenson and Wagner emphasized the importance of including information obtained from real world data source in student projects [Stevenson and Wagner, 10]. CloudSpace provides students with a Web Robot Framework to access content from the World Wide Web, which streamlines this process by providing students with a simple fetch or download command. CloudSpace's web robot implementation parses HTML and other formats and presents it in an easily accessible form to the students' applications. A dedicated web robot thread executes all page requests and caches them, reducing the load on external servers.

FUTURE WORK AND CONCLUSION

As our development continues, we are focusing on providing a complete CS1 curriculum based on CloudSpace. We plan to provide a full set of laboratory assignments and programming assignments that will be made available to other instructors who want to use CloudSpace in their courses. CloudSpace will include example assignments where students build their own social networking web sites with the modern features that students expect from their own real-world experience. The CloudSpace curriculum will also include textbook recommendations for instructors who want to incorporate CloudSpace into their courses. The CloudSpace team will continue to revise and refine the infrastructure so that a simple, easily understood architectural approach to designing web applications can be conveyed to students.

To create a more accessible distribution, CloudSpace will provide a one-file WAR distribution of CloudSpace on-line, so that instructors who wish to evaluate it or use in a course can do so easily. We also plan to provide a remote-hosting server for CloudSpace, so that institutions that do not have the manpower or resources to set up their own server infrastructure can still try out CloudSpace or use it in their classes.

REFERENCES

[1] *Apache Tomcat*. Available from: http://tomcat.apache.org/.

[2] *BeanShell - Lightweight Scripting for Java*. 2009; Available from: http://www.beanshell.org/.

[3] *Blue J - Teaching Java - Learing Java*. Available from: http://www.bluej.org/.

[4] *Eclipse.org home*. Available from: http://www.eclipse.org/.

[5] *Javassist*. 2009; Available from: http://www.csg.is.titech.ac.jp/~chiba/javassist/.

[6] *Xstream Library*. Available from: http://xstream.codehaus.org/.

[7] *ZK Direct RIA*. 2009; Available from: http://www.zkoss.org/.

[8] Beaubouef, T. and J. Mason, *Why the high attrition rate for computer science students: some thoughts and observations.* ACM SIGCSE Bulletin, 2005. **37**(2): p. 103-106.

[9] Edwards, S. *CS1114 Virginia Tech*. 2009; Available from: https://moodle.cs.vt.edu/mod/resource/view.php?id=10770.

[10] Stevenson, D.E. and P.J. Wagner, *Developing real-world programming assignments for CS1*. ACM SIGCSE Bulletin, 2006. **38**(3): p. 158-162.