

CO-OPERATIVE EDUCATION PROGRAMS



uOttawa

L'Université canadienne
Canada's university

FACULTY OF ENGINEERING SOFTWARE ENGINEERING

Third Work-Term Report Examples

Title	Page no.	Term
LiveCycle and Flash Development	2 – 29	Fall 2009

Disclaimer

Copyright © All rights reserved. The work-term reports are the property of the author or authors and they are subject to copyright, trademark, and other laws. Your use of the Web site does not transfer to you any ownership or other rights in the contents of the work-term reports. The work-term reports are posted courtesy of the CO-OP Office, students and employers and with their explicit permission. All students are subject to the University policy on fraud, which prohibits the copying or use of any parts of these reports for use in other reports.

SEG 3902

CO-OP WORK-TERM REPORT (Fall 2009)

LIVECYCLE AND FLASH DEVELOPMENT

Adobe Systems Canada

By

XXXXXXXXXX
XXXXXXXXXX

Presented to XXXXXXXXXX

Software Engineering Academic CO-OP Coordinator

University of Ottawa

January 15, 2010

TABLE OF CONTENTS

Abstract	5
I Introduction	5
II Environment	6
A. Adobe Systems	6
B. Learning Resources	7
III LiveCycle	7
A. What is LiveCycle?	7
B. How to use LiveCycle	8
IV Docmap	9
A. Flash, ActionScript and AIR	9
B. Design	10
C. Papervision3D	11
D. Development and problems	11
V Checklist	13
A. Design	13
B. LiveCycle integration	14
C. Problems	15
VI Other Projects	16
A. Documentation update	16
B. LiveCycle self-installer	17
VII Recommendations	18
A. Adobe	18
B. CO-OP program	18
VIII Conclusion	19
Bibliography	20
Appendices	21

LIST OF APPENDICES

A	LiveCycle Workspace ES	21
B	LiveCycle Workbench ES	22
C	Docmap	23
D	Viewport manipulation	24
E	Checklist app	25
F	Invoking a LiveCycle process – Sample	26

Abstract

As web pages become more complex every day, the need for powerful service-oriented applications is at the utmost. LiveCycle allows services and data to be accessed from anywhere using many methods. The main purpose of a Flash developer is to deliver rich and interactive applications to users. The popularity of Flash speaks miles to its advantages and you can find it on most web pages today. My main duties during my work term at Adobe Systems Canada revolved around the development of a new interactive document catalogue and the implementation of a writer's Checklist application. I also contributed to the Learning Resources team by updating help topics and creating a script to automatically update LiveCycle. I was committed to delivering great applications that would impress users and encourage them to pursue Flash. I had the opportunity to see how a large enterprise application, such as LiveCycle, is documented and is used to deliver user-centric services. I utilized many skills learnt in my engineering studies while developing new technical knowledge and valuable know-how in certain aspects of web programming. Also, I gained interpersonal and team skills by working with strong individuals who allowed me to learn from their experience. This work term further amplified the power of web technologies and the applications that can be built around them.

I Introduction

For my third work term in Software Engineering at the University of Ottawa, I worked for Adobe Systems Canada from September 8 through December 23, 2009. I was employed as a Documentation/Samples Development Intern where I reported to XXXXXXXX, XXXXXXXX. Adobe is one of the world's leading software companies and prides itself on innovation and customer satisfaction. Adobe Systems Canada is based in Ottawa, and is just one of many global locations for the industry giant. Its headquarters are located in San Jose, California.

Early in the work term, I met with my supervisor to lay out my goals and objectives for the coming months. We discussed what Adobe strives for and how my skills could benefit its progress. I planned to learn many new programming languages, including Flash, while executing my day-to-day tasks. I was also interested in seeing how a large company functions with the never ending supply of products and expectations. More importantly, how it goes about developing new ideas and creating an organized workflow. Throughout my work term, I had the opportunity to learn new areas of programming and Adobe's wide array of applications. From a software engineer's point of view, I wanted to experience complex software development in a large group and see how it differs from smaller applications.

Overall, my main duties included application development, documentation and software maintenance. My primary role was to complete an online documentation catalogue which required ActionScript programming. I was also tasked with developing and maintaining an in-house AIR utility used by a writing team in San Jose. Finally, I assumed a smaller role with the local SDK team to update and validate existing help guides. Through all these tasks, I had the

opportunity to work with many types of people and experience parallel software development. I was also fortunate to be present during the release of the latest version of Adobe LiveCycle ES2, a product which eclipses the size of any project I have ever been part of.

In this report, I will provide an overview of the tasks required to develop and maintain Flash software. While covering the process, I will give a brief explanation of projects and problems encountered. I will provide observations and lessons that I took away from my work term. Finally, after my work term at Adobe, I can attest that software, in general, has endless possibilities and Web 2.0 is only a stepping-stone to the future of the web.

II Environment

A. Adobe Systems

For most parts of the world, Adobe is synonymous with Adobe Photoshop; however, Adobe has developed a large library of products ranging from consumer creative multimedia applications to business back-end servers. With over 7000 employees worldwide, Adobe is well diversified and is one of the most recognized names when it comes to large software companies. XXXXXXXX is the company's CEO and, for the past couple of years, has been the face of Adobe.

Adobe's line of products is vast and offers many solutions to different fields of interests. Products such as Adobe Photoshop, Adobe Premiere and Adobe Dreamweaver are flagship applications that have become Adobe's bread and butter. Currently in Creative Suite 4, Adobe continues to be the leader and standard in multimedia editing. The addition of stripped down versions such as Adobe Photoshop Elements allow the company to cater to a wider audience and offer its services to more individuals. A proper introduction to Adobe would not be complete without one of its biggest contribution to the tech world: PDF. The Portable Document Format (PDF) has become the universal way of transferring documents from person to person and was recognized by the International Organization of Standardization as ISO32000. PDF began in the early 1990's and has helped make Adobe the household name it is today. Through its free distribution of Adobe Reader, PDF has grown into a very strong file format and is continuously evolving to accommodate new technologies and standards.

In recent years, Adobe has been concentrating on delivering a richer and colourful experience to web users. Its proprietary Flash technology (acquired from Macromedia) and Flex frameworks are changing the way web pages are imagined and developed. Popular web crazes such as YouTube and MySpace are highly dependent on the Flash technology and offer great examples of how Flash can improve a web user's experience. As most companies are heading towards web-based and cloud computing, Adobe has been following suite and developed Adobe AIR which allows web-enabled applications to run on your desktop as if it were a regular installed program.

B. Learning Resources

Being such a large company with such a variety of products, Adobe must provide adequate documentation and community assistance to its users. The Learning Resources (LR) team is responsible for the development and upkeep of documentation catalogues pertaining to certain applications, like Adobe LiveCycle. The team is continuously releasing new help guides and attending public conventions such as Adobe MAX where users have the opportunity to talk directly with developers and other Adobe employees. Earlier in my work term, a couple of members of the Ottawa LR team had the opportunity to go to MAX 2009 and talk with LiveCycle customers. There, they were able to gauge how customers use the help guides, what are the biggest problems and what the LR team can do to improve it in the future.

During my stay with the LR team, I had the chance to witness firsthand the release of a major version of a program. Adobe LiveCycle Enterprise Suite 2 was released in November 2009 and with it came a whole new documentation catalogue. The sheer amount of information contained in the new help docs and the synchronization of the team are staggering. Although I only played a small role in developing documents for the new version, I worked closely with the team to help create utilities and SDK related articles.

III LiveCycle

The Ottawa Branch of Adobe is primarily focused on business productivity applications. One of the biggest players in this field is LiveCycle. Although it is not as widely popular as many other products, LiveCycle occupies a key role with many businesses looking to automate and organize their workflows.

A. What is LiveCycle?

During my first week with the LR team, many people tried to explain what LiveCycle is exactly, and the best description I can say is: LiveCycle is a web service container. LiveCycle encapsulates many technologies and is used to automate processes and general workflow. For example, imagine a new employee is starting a job at Adobe. A contract has to be written and accepted for this person, an employee record has to be created, a badge has to be made, an IT account has to be activated, a computer has to be installed, pay cheques have to be prepared, ... All these steps involve information being passed from person to person to follow specific guidelines. Not only are these steps long tedious work for HR employees, the possibility of introducing mistakes grows with every handoff. When you factor in the amount of companies that follow this type of workflow for new employees, the time lost due to inefficiency is astounding. This and many other scenarios, like tax returns, bank loans and car rentals, happen thousands of times a year. LiveCycle can help with this type of process that follows a systematic series of steps to accomplish a goal.

Having the capability to map out a workflow and have decisive steps and checkpoints for a task makes the process a lot more convenient and safe. LiveCycle ES2 allows the creation of processes which can be used dozens, thousands or millions of times over and over without having to worry about users forgetting to make a crucial step. Having this sort of template makes tracking and logging a lot simpler. For example, instead of calling your HR representative to find out why your claim has not been accepted yet, you can simply view the process and see that it is currently being held by this person and waiting for a credit check.

LiveCycle offers many features and access points. The Workspace is the primary user interface for users. It allows LiveCycle users to start new processes or track previously created processes. It is the simplest way for end-users to start processes out-of-the-box. For example, when it comes time for me to complete timesheets for Adobe, I open up the Workspace from my Internet browser, choose to start a new “TimeCard Canada” process and fill my hours. Once I complete the PDF form, LiveCycle takes over and sends the completed form to my manager. My manager receives notification that I have filled out a timecard and it requires approval. At that point, my manager has the option to accept or reject my hours. Depending on the choice, LiveCycle will alert me that the timecard has been rejected or start another process for the billing to occur. Appendix A shows the general Workspace layout and a list of available processes. From the Workspace, a user can view the state of open process or even review old process. Similar to a package going through the mail; at each step, the process is tagged and routed to the next step.

As we will discuss later on, LiveCycle processes are not only available from the Workspace, but can be launched from programs and/or applications. This means that LiveCycle is accessible from stand-alone desktop applications, web pages and even mobile devices. LiveCycle is clearly a large piece of software and revolves around many technologies which allow it to facilitate workflows and automate any process.

B. How to use LiveCycle

A LiveCycle server implementation is only as good as its processes. So when it comes time to create those processes, Adobe has included another piece of software in the LiveCycle ES package, LiveCycle Workbench ES. This tool lets developers, administrators and HR representatives create workflows and really map out the journey from start to finish. The Workbench was designed with ease in mind. A drag-and-drop interface is the simplest way for users to drop activities in the process’s timeline and link them with arrows. When creating this timeline, users can create multiple routes (equivalent to an if-else statement in a program) which allows for powerful processes that do not rely on human intervention to make routing decisions.

The best way to describe the Workbench is to call it programming with a graphical user interface. Users have the ability to create variables which can be used as input, output or just for logic. The Workbench provides a start point, just like a main method in Java. From there users can link activities in a sequential matter or use conditional statements to create multiple routes. An activity dropped in a process is comparable to calling a function in programming. The process will halt until the activity is completed and then the output variables of the activity can be mapped to variables in the process, just like assigning the return value of a function to a local variable.

When a process is completed, it can either be used as a self-contained process to be invoked from the Workspace or it can be re-used in another process as an activity. Having the capability of creating re-usable processes like this can prove to be a big time-saver because multiple processes can then be linked to create a very complicated process that does not require much input from the user. LiveCycle ES also comes pre-packaged with processes that are typically used in business environments, such as encryption, rights-management and many others. These built in processes can be used to make a custom process even more powerful. See Appendix B for a Workbench screenshot and basic explanations of the GUI.

LiveCycle also offers different ways to programmatically invoke processes, not just from the Workspace. Using the LiveCycle SDK, application developers can dynamically invoke processes from Web services (ASP.NET), Invocation API (Java) or even Remoting (Flex). These types of invocation were instrumental to the Checklist application and will be discussed later on.

IV Docmap

The first project I worked on was a documentation catalogue (docmap). The idea behind this tool was to make the LiveCycle help page more interesting and interactive. Instead of having a plain HTML page with a topic tree on the left and articles on the right, Adobe wanted to create a fun new interactive webpage where users could go through doc nodes and search topics. The previous COOP student had started this and it was my job to get it ready for the ES2 release.

A. Flash, ActionScript and AIR

To fully understand the different components that went into my projects, it's important to distinguish the difference between Flash, ActionScript and the AIR framework. Flash, also known as the Flash Player, is the technology that allows .swf files to be loaded into the browser and played correctly independently of the system running the Flash Player. The .swf file can be created using many different techniques and different applications, such as Adobe Flash CS4. ActionScript is an object-oriented programming language. It can be embedded within .MXML files or as a separate .AS file, where it can be used like any other scripting language. An ActionScript project can be compiled into a .swf file which can be viewed from a browser using the Flash Player. This is similar to the Java environment where a JVM (Java Virtual Machine) is required to run Java files. Finally, the AIR framework is basically Adobe's way of getting web apps onto the desktop. The AIR client allows users to install and use .air apps just like any other program on the machine. The major advantage of AIR is the capability of having web-based apps running offline and allowing the user to have a richer experience. Because AIR apps run as a stand-alone program, they are not limited like browser applications and can easily access the user's local file system.

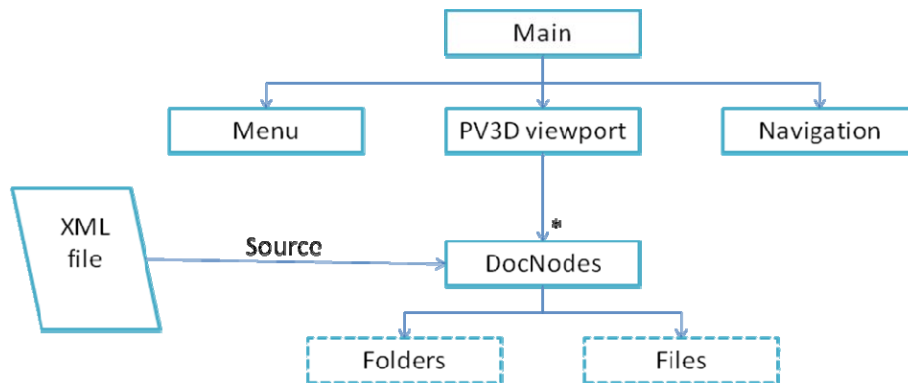
B. Design

The docmap was created to make the LiveCycle help files more accessible. Instead of browsing through endless titles and sections, the docmap breaks the help articles into categories and lets the user navigate freely between them. When the user reaches a file (a leaf in the tree structure), they are presented with multiple links to topics concerning this subject. Currently, the docmap links are pointed to the articles in the HTML help guide, however, in the future, these links will be enriched with videos, PDFs, zip files and samples. At the time of writing this report, the docmap was still being tested internally and not released to the public; therefore I cannot provide a link to the actual application. Please refer to Appendix C for a screenshot of the application and a sample leaf that has an HTML link, a PDF link and a video link.

The docmap was built using ActionScript, which was an entirely new programming language that I had never even heard of. My co-worker assured me that if I knew object-oriented programming, like Java, that I would pick it up easily. As he mentioned, the language is very similar to Java and C++. The idea of classes, packages, variables and functions are all present in ActionScript. The only anomaly that I found was when declaring variables you have to follow this format: “*var name:Object = new Object();*” which appears to be a mix of Java and Visual Basic. Other than small hiccups like those, I found myself learning the language very quickly and digging directly into the docmap source code to learn how it works. Adobe Flex Builder 3 was used as the development environment; it is by far the easiest and most popular Flash IDE.

The docmap is a very interesting application from a software engineering point of view because it has many sides and components interacting with each other. The information contained in the docmap (the links and titles) are all generated from an XML file. This was obviously chosen to allow for dynamic content and new nodes to be added or removed as needed. Once the XML is parsed and DocNode objects are created, the application begins to build the 3D environment and the surrounding elements, such as the menu bar and navigational controls at the lower-right corner. Once these elements are on screen, the application is ready for use and waits for users to navigate the folders. Whenever a folder is selected, all the children of this node (from the xml) are created and added to the screen. It's this kind of interactivity and liveliness that makes the docmap a much more enjoyable experience over a traditional HTML help guide.

Immediately after introducing me to the docmap, the team described the features they wanted to add to the application. Generally, the features included navigation and interface tweaks because the core of the application was already in place, but not very easy to use. We had weekly meetings to review the latest changes and brainstorm new ideas and suggestions that would make the application more accessible. I will not be discussing the software's architecture as it was mostly the past CO-OP student's work. However, a simplistic view of the architecture has been provided. This is by no means a standardized class diagram; its purpose is to show the relation of components and a general idea of the workflow.



C. Papervision3D

Papervision3D is the power behind the docmap's 3D environment. It is a popular framework used by many flash developers to create simple 3D scenes and venture into the world of 3D programming. Before embarking into the docmap's code, I followed many Papervision tutorials on the web. These simple but effective tutorials allowed me to fully understand the intricacies of the 3D world. Most people are familiar with the XYZ coordinate system; however, when programming 3D scenes, the camera is often the most complicated aspect. Not only do you have to manage where objects are in the space, but you also have to control where the user's point of view is located and oriented. The best way to describe this phenomenon is to imagine a user standing inside a house looking outside through a window. The outside world is endless and can move in and out of the window's view path, but the user can only see what is directly in front of the window.

In addition to having 3D objects in the environment, the docmap includes a menu bar and a navigation panel. These extra features are overlaid in front of the Papervision3D on the 2D plane. This 2D layer is called the "viewport" because it is essentially what will be viewable on screen. Again this can be compared to having stickers on the window of the house or even the HUD (heads-up display) in augmented reality binoculars. Managing these different layers and understanding the visibility of each object in 3D space is a skill that I had to quickly learn if I was going to improve the docmap.

D. Development and Problems

This section will give an overview of the upgrades made to the docmap and some of the problems encountered during the development. I will be focusing on three problematic areas that required significant amount of work to resolve: the font, the window size and the performance.

Early in the development, I was making small changes that made navigation and general use simpler. Improvements included making the folders clickable instead of having to click a plus/minus button to open it, making folders and files more distinguishable from each other, making upper level nodes smaller to create less visual distraction for the user, etc. These small changes were much appreciated and for the most part simple. However, when the team asked to use a special font for the text, I was caught off guard by how much work was needed to include custom fonts in a .swf file. Adobe Clean is a custom created font used by Adobe for most publications. It is a TrueType Font available to the Adobe internal employees. Naturally the

first step was to change the font for all text in the application to the Adobe Clean family name. And it worked! Well, it worked from my local computer. After testing the docmap on multiple computers, we realized that the font was defaulting back to Times New Roman when using a computer that didn't have the font installed. This is because .swf files are executed from the client end, not the server's, so when the .swf file looks for the Adobe Clean font, it cannot find it and goes back to the default. This was my first mistake, thinking that the processing happens on the server-side when, in fact, it is on the client-side. After a lot of Google searches and tests, I found out that the true-type font can be converted to an open-type font (.otf) that can then be embedded into the docmap .swf file. When the application loads up, the .otf file is read and recognized as a font file. This puts all the font glyphs in memory and makes them available for the application. This shows the versatility of .swf files and the Flash Player.

When development began, some settings were hard-coded to create a working prototype of the docmap. This helped the team see a visual representation of the application and give comments and suggestions which became requirements. The size of the .swf file was one of the settings that was hard-coded, because when creating a .swf file, you can manually set a pixel size or give a percentage of the window size. The previous CO-OP student and I both used a native resolution of 1280x1024 for our development environment so it made sense to make the .swf file of the same dimensions. This caused a problem when a browser window was not maximized or a user opened the application with a different native resolution. Because Papervision3D is not a simple element or container it cannot be stretched and manipulated like other components (such as textboxes, images and buttons). Setting the .swf file to "100%" of the browser window was the answer, but the 3D environment needed a little bit of tweaking. As mentioned previously, the viewport is like a window to see outside, so if you re-size this window you will see less of the outside world. Essentially, the 3D scene has to be manoeuvred and zoomed to match the browser window. Whenever the Flash Player detects a window resize, the entire 3D scene coordinates are re-calculated and re-positioned to make the user believe that the 3D scene is moving with the browser, but this is just an illusion. By playing with the Z axis of the scene's camera, I can make the camera move in and out, essentially making the doc nodes appear smaller or bigger. So the smaller the browser window, the more the camera is moved away from the scene to make the folders seem smaller. Finally, by changing the X and Y axis of the entire 3D scene the action is always kept centered in the browser. See appendix D for more explanations. Of course, all these manipulations can create a lot of overhead and become very complex, which brings us to the next problem.

Performance has never been a big issue in my past projects and software applications; however, this docmap has definitely introduced me to the world of performance and load testing. The docmap can easily have hundreds of items displayed on screen at the same time. With each folder having three distinct geometric planes, and each plane having a background fill, text and/or lines, it's easy to imagine the workload on the processor. Add the 3D effects and viewport re-sizing and you have quite the memory hog. As the development moved along, more and more features were being added to improve the aesthetics and controllability. One of the sources of the slowdown was that children of a given folder were being re-created every time a user would click on the folder. For example, if "Folder A" was opened, it creates three children files "A1", "A2" and "A3". When the folder is closed, these three files would be made invisible and removed from the scene. But when the folder is re-opened, three entirely new files would be created. Luckily I was able to notice this memory leak and rectify it by keeping pointers to the

old files and re-displaying them. This helped some of the performance woes but during some of my testing, the browser was still reaching a peak of 325Mb of usage when all the branches had been visited in one session (the memory usage is directly related to the number of folders opened). It also doesn't help that Papervision3D apparently suffers from a memory leak when creating planes in the 3D environment. Many message boards state that users of the framework are experiencing the same issues as the docmap. I tested this bug by making a 3D plane appear and disappear over a hundred times in the scene. Unfortunately, the message boards were right. The memory usage grew at a constant rate even though I was re-using the same plane. At the time of writing this report, the future of the docmap was in question due to these performance issues.

V Checklist

Being in the LR team, I had the opportunity to see how articles are written, edited, reviewed and published. My second large project was the Checklist app which is used by writers to ensure that all steps are completed before sending a document for production. A writer uses the app to complete a form (checklist), which is sent to the lead writer of the given document and, when approved, is sent to production.

A. Design

The Checklist app was developed in an AIR container which means that it's installable on the user's computer like any other application. Overall I was very impressed by the AIR framework and the tools that it provides to create simple but stunning looking graphical interfaces. Like the docmap, I was continuing the work of a previous CO-OP student. Luckily, with this project, I entered early enough in the development so I was able to provide more input and make design choices.

The app was basically divided into two sections, the Viewer and the Manager. The Viewer was used by writers to fill out checklists and submit them, whereas the Manager was used by administrators to create new checklists or manage the employee/project information. Whenever a writer submits a checklist, an email containing a PDF version of the checklist is sent to the lead writer, who can then review the checklist and approve the production. Both sections were similar but offered different functionalities, and for that reason they were kept as separate .air files. This meant that both applications would have to be installed separately if the user wanted both. Later in the development, they were merged together to make things simpler and a lot cleaner for users.

As mentioned, the app was developed with AIR which means that it uses MXML and ActionScript for the code. MXML provides your standard GUI elements: buttons, labels, grids, etc. Because the checklists are created dynamically, the interface could not be hard-coded like most application that I have worked with in the past. Grids played a large role in the app because they allowed the checklist to grow and have the correct alignment. During a code-review with a

co-worker, I explained how many grids are interwoven to create the desired look of the app. There are sometimes up to 8 grids embedded within each other to create a single question of a checklist. For a screenshot of the Checklist app, please see Appendix E.

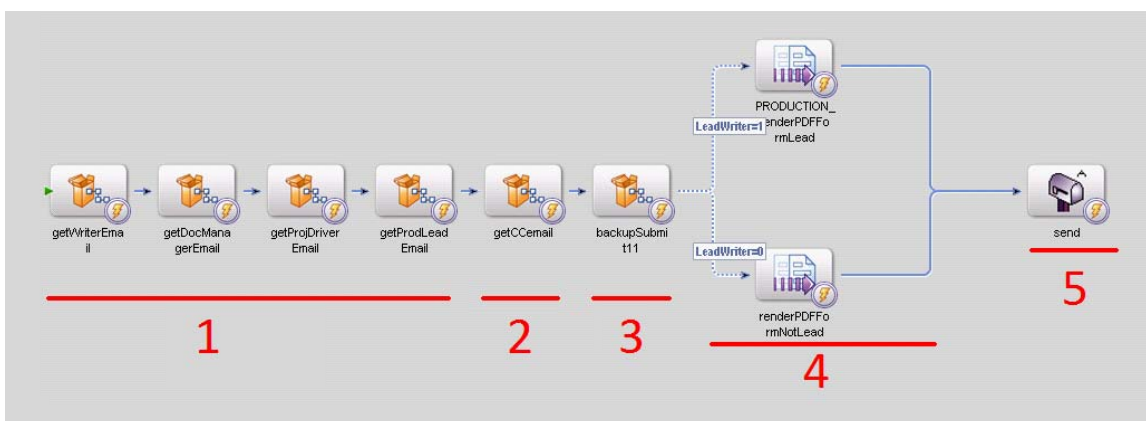
In general, the Checklist app can be divided into three components: the client-side interface, the LiveCycle server and the MySQL repository. The three parts work together to gather the information from the user, process the information and store the information. Having all these components interacting is a double-edged sword, because, individually they offer great services but finding the source of a problem can be a time-consuming task.

B. LiveCycle integration

LiveCycle was developed to automate processes and simplify workflows for the user. This idea of abstracting the business logic with LiveCycle allows a developer to use it as an API. The Checklist app is a great example of LiveCycle being used to communicate with the MySQL database. This helps to create a lightweight client and divide the logic from the interface.

Creating this API is really simple. By using Java, you create application logic that will be mapped to LiveCycle processes. For our purpose, we developed methods that are used to talk to MySQL, for example, methods like `getAllEmployees()` and `deleteProject()` which send queries to the database. Once this application logic is created and tested, a `component.xml` file is created to map the Java methods to LiveCycle process names. These names will become the API methods that will be available from the outside. The whole Java project is compiled to a JAR and uploaded to the LiveCycle server using the “Install Component” feature. And that’s it. If all the correct testing was done, LiveCycle is now ready to accommodate the Checklist app. Appendix F demonstrates the typical way to invoke a LiveCycle process through Flex with Remoting. More information can be found in the LiveCycle SDK help guide.

Although the process may seem ambiguous at this point, the point of this section is to demonstrate how LiveCycle is used within the app, not demonstrate how the app functions. Overall, the checklist uses over twenty custom processes from LiveCycle. The most important and complicated one is the `sendRequest`, which is invoked whenever a user is ready to submit a checklist. A checklist is either directed to the lead writer or to the production which includes a doc manager, a project driver and a production lead. Here is the process as viewed in the Workbench:



1. We call the custom `getEmployeeEmail()` method four times to fetch each person's email address from the database.
2. Fetch the emails of the all the CC'd names with the custom `getEmailString()` method.
3. Write the checklist information to the database using the custom `backupSubmit()` method.
4. Depending on whether they are a lead writer, we generate a PDF using the checklist information. This PDF is generated using an `.xdp` template file and a LiveCycle process.
5. Call the LiveCycle `sendEmail()` process to send the mail to all the emails.

The above workflow is only valid for users who are writers. The Checklist app also contains four more user types which have slightly different workflows. For the most part, only the email destinations will change so the `sendRequest` is re-used. By changing the input names for each of the doc managers, project drivers, etc., we can manipulate where the email will be sent. Again this demonstrates the power of LiveCycle and how one process can be adapted to different situations.

C. Problems

This project had its fair share of hurdles and missteps. Personally, I think the biggest problem was that a requirement list was never properly created. When my manager introduced me to the Checklist app, they warned me that this application was never supposed to become this big. It was meant as a smaller in-house utility that a few people could use to submit checklists to their lead writers. I was told that since then, the app had exploded in size and the customer was looking for even more features for people to use. It was at that point that we should have sat down and created a list of requirements and organize the project deliverables. Instead we adopted an ad-hoc development process which saw the customer email me feature requests and suggestions. This was great when it was minor fixes like a wording change or moving an element slightly to the right. But as the project grew bigger, the changes became more complex and required more work to develop and test. Because I was the only developer working on the project, I had to do the planning, development and testing, so whenever a new feature was added, I had to make regression tests to verify that old code was still working properly. Luckily the app didn't grow too big, so the extreme programming style did work in the end.

One disadvantage of using this style of development is that the code can sometimes become messy. In this particular project, the customer often went back and forth between ideas and features, which meant that the code had to be flexible enough to accommodate the changes. No matter how many times the customer assured us that this was the final change, they asked to revert back the week after. For this reason, a lot of my code is scrambled with commented out sections to allow going back to other features. This makes the code hard to debug and even harder for other developers to understand. Unfortunately, that is the price we have to pay when a project is not properly planned and is continuously changing.

Another skill I learnt on this project was how to work with customers who are not in your geographical area. In this case, the person who I was working with to validate the application lives in San Francisco, which means that we cannot have face-to-face meetings to discuss ideas. This was the first time I worked so closely with people that I have never seen. I learned to make my ideas as clear as possible over the phone and in emails to guarantee that we have the same plan in mind. Adding the time difference between the east and west coast, it became very

apparent that communication would be a key player in getting the correct app to the client. Luckily, the customer contributed a lot of feedback and replied quickly so I never really had to change an entire piece of code because we had a different idea in mind. However, this was a great learning experience and allowed me to see how hard it must be to work with multiple developers in different areas.

Lastly, the Checklist app allowed me to see the implications of upgrading a software application. After a month of development, we were ready to release a version 2.0 to the customer's group. Unfortunately, the new 2.0 features required fields to be added to the database and new processes to be created in LiveCycle. This was clearly a large update and required server downtime for us to get all the changes done. For example, in version 2.0, a login screen was added to authenticate the user and give access to the Submitter and Administrator (now that the apps were merged into one Checklist app). Therefore, a login attribute had to be added to the employee records in the database. For the upgrade to go faster, I created a simple script that retrieved the employee's login from the Adobe LDAP repository, and automatically added it in the MySQL database. Luckily the upgrade coincided with American Thanksgiving which means that the server downtime affected a small number of people. The upgrade went very smoothly and the deadline was met. Since then, the client-side has had a few tweaks but the server and database have not had to change again.

VI *Other projects*

I had the opportunity to work on many different types of projects during my stay with the Ottawa LR team. Although the docmap and the Checklist app were my main areas of focus, I was also part of the LiveCycle SDK team, which allowed me to work on a few smaller projects.

A. Documentation update

Being an engineer in a writing team has allowed me to see both sides of the software world: the development of the application and the supporting documentation. While learning to use LiveCycle, I used the help documentation many times, and I also got the opportunity to contribute some samples to the document. With the release of LiveCycle ES2, the SDK team wanted to improve some weak topics that were not very well written. The team and I decided that it would be a good experience for me to get some writing experience, since I am in a writing team, after all. I updated a few topics but most notably, I contributed to the "Creating Directory Service Providers". This guide shows how to synchronize the LiveCycle user directory with an outside source. For the sample, we used a MySQL database with usernames and passwords. By following the guide, LiveCycle developers can pull users directly from any repository to create dynamic LiveCycle users who can then access the workspace. Although I was hesitant at first, updating this guide turned out to be an immensely helpful way to learn how LiveCycle handles users. But it did not change my optimism when it comes to documenting code...

B. LiveCycle self-installer

One disadvantage about LiveCycle is that it takes a very long time to install. Adobe does provide a turnkey installation which installs everything for the user, including the JBoss Server, the MySQL database, the Java JRE and the LiveCycle application. However, the install still takes over one hour to complete and then the user has to configure the settings which can take anywhere from 30 minutes to one hour depending on which features are enabled.

For this reason, my manager approached me about another side project. He said that the team is always re-installing LiveCycle on servers because the engineering team releases weekly builds of the application. He thought that it would be great if there was a script that could silently fetch the latest version and install it silently. That way the team wouldn't have to waste valuable time running this installation themselves. We discussed the process involved and informally laid out the requirements. The process can be broken down into 5 steps:

1. Copy the new version of the LiveCycle installer to the local drive
2. Uninstall the old copy of LiveCycle
3. Install the new copy of LiveCycle
4. Run the configuration manager
5. Test that LiveCycle is running (ping the Workspace)

LiveCycle comes shipped with GUI and CLI versions of the installer, uninstaller and configuration manager, so by using the CLI version I can launch the different .exe for every step. I used Java for the programming because I feel very comfortable in that environment. Looking back, I'm sure there were much better scripting languages, but this got the job done. Java offers a process controller to launch applications, *Runtime.getRuntime().exec(...)*. By using the Runtime object, we can specify files to execute, just like if we were typing in the command prompt.

The script itself is a runnable JAR file, so the user can set up a Windows Scheduled Task to run it nightly. During the install and configuration, property files are given to set settings and variables. For example, the *installer.properties* file contains a *USER_INSTALL_DIR* which users can change to set the install directory of Livecycle. The same concept is applicable for the configuration manager.

In the end, the entire script takes approximately 1.5 hours to complete and is completely user-independent. This means that the script can be placed on a server, and theoretically, the server will always have the latest version of LiveCycle installed. I say theoretically because the engineering team has not released a new build since the release of ES2. Therefore I have only been able to test the script with a local version of LiveCycle and changing version number manually. My manager has informed me that by January new builds will start to be released and then the script will be put to work.

VII *Recommendations*

Having spent a work term with Adobe and encountering many different situations, I have created a few recommendations which would improve the readiness of a CO-OP student. These are merely observations and do not reflect anybody's opinion but my own.

A. Adobe

Working with LiveCycle for an entire work term has opened my eyes to the endless possibilities that it has. Obviously, the next step for LiveCycle is to get a lucrative deal with large corporations and/or the government and get the LiveCycle name in the public eye. Even though this is an enterprise product, most people have never heard of LiveCycle, I certainly had never heard of it before working at Adobe. Adobe for the most part is doing very well in the software field, competing with the likes of Google, Apple and Microsoft. I commend Adobe for venturing into web-based apps and moving towards cloud-computing.

Judging by the projects I worked on, I would recommend that a more structured approach be used for tool development. I understand that I worked on relatively small projects, but I often found myself wondering what other features would be added in the future. If possible, I think it would be interesting for future CO-OP students to be immersed in group-based development. As a programmer, I think it's important for people to work concurrently on a project and work together to complete an application. Realistically, no large application is ever completed by a single individual.

B. CO-OP program

As with my other CO-OP reports, I would recommend the University and the CO-OP program to enforce more variety in programming languages. The University of Ottawa has a great Java curriculum, but lacks in respect to other programming environments. Flash is a very popular web programming language and could benefit many students. As the web continues to evolve it's important to teach students the new technologies, especially in software engineering.

I would also recommend focusing on web-enabled technologies. It appears that everything is moving online these days, so it would be good to introduce students to web standards earlier in the program. I learned many aspects of web technologies in the class CSI3140 (WWW Structures, Techniques and Standards). I enjoyed this class very much and it helped me understand the inner-workings of the Internet. It was not the most advanced class and could be offered at a lower level.

VIII Conclusion

In conclusion, LiveCycle is undoubtedly a world-leader in service-oriented software. Its ingenious processes and overall controllability make it adaptable to most businesses. As we create stronger and more complex applications, LiveCycle and Flash will always be in the forefront to create that link between the user and the service. LiveCycle demonstrates that it's not the technology that limits us, but the way we use it.

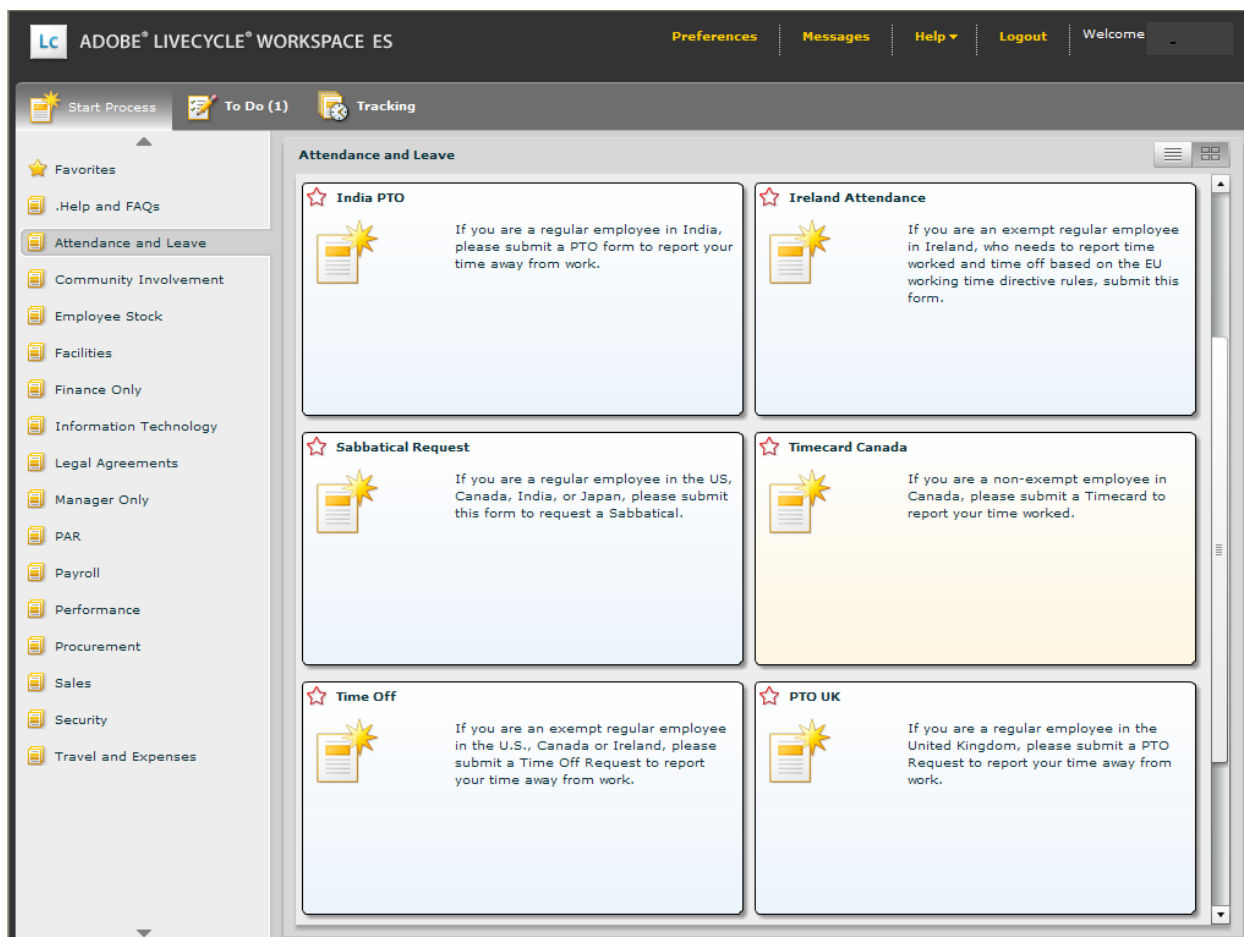
During my work term, I was able to apply concepts and knowledge learnt through my classes and personal experience. Through challenging projects and tasks, I am able to draw many lessons and will incorporate such information into future applications. I had the chance to develop a documentation catalogue and a checklist app that enhanced my knowledge of Flash and LiveCycle. I would recommend including a wider array of programming languages into the engineering program as it is invaluable for software engineers to be well rounded in many languages.

I would sincerely like to thank all employees of the Adobe Ottawa LR team for allowing me the chance to be a part of their group and to learn from their expertise. I gained valuable hands-on experience about LiveCycle and the entire Flash framework. Most importantly, I gained interpersonal and team skills by working with a supporting team which dedicated their time to teach me. I valued my time at Adobe and hope to be able to carry on all my new skills into my future projects.

BIBLIOGRAPHY

- [1] Adobe LiveCycle Workspace ES, <http://lcforms.corp.adobe.com/workspace/>, visited December 7, 2009
- [2] Execute an external program – Real's Java How-to, <http://www.rgagnon.com/javadetails/java-0014.html>, visited December 9, 2009
- [3] Inside Adobe, <http://inside.corp.adobe.com/>, visited December 10, 2009
- [4] Installer commandline – LiveCycle – Adobe Zerowing, <https://zerowing.corp.adobe.com/display/lc/Installer+commandline>, visited on December 9, 2009
- [5] Installing and Deploying LiveCycle® ES2 for JBoss®, *install_jboss.pdf*
- [6] LiveCycle ES Overview October 2008, printout manual
- [7] LiveCycle ES2 * Programming with LiveCycle ES2, http://help.adobe.com/en_US/lifecycle/9.0/programLC/help/index.html, visited December 7, 2009
- [8] Self-Service – LDAP at Adobe, <http://isweb.corp.adobe.com/isweb/iskb.nsf/docs/IS007736>, visited December 10, 2009
- [9] Tutorial List at Papervision 3D Tutorials, <http://papervision2.com/tutorial-list/>, visited December 10, 2009

APPENDIX A: LiveCycle Workspace ES



The Workspace offers the easiest way to use LiveCycle processes. In the upper-left hand menu, users can start a process, see a “To Do” list of processes or even track the progress of past processes.

In this screenshot, you can see that I am logged into the Workspace (as indicated by my name at the top right). From here I can select the “Attendance and Leave” category and fill out a “TimeCard Canada”. This will launch a PDF form that I can fill out and submit.

When a process is started, it can be viewed from the “Tracking” tab. In the case of a TimeCard, the last step of the process is for the submitter to review his paycheck stub. When this happens, the process is moved back into my queued list (To Do), where I can go complete the process.

APPENDIX B: LiveCycle Workbench ES

The screenshot displays the Adobe LiveCycle Workbench ES interface. The central canvas shows a process flow with three main components: an 'emailWithDocument' task, an 'encryptPDF' task, and a 'writeDocument' task. A red box highlights the 'encryptPDF' task, with a red arrow pointing to its configuration panel on the right. The configuration panel is divided into several sections: General, Input, Encryption Options, and Output. The 'Input' section shows 'Input PDF - (variable)' set to 'inPDF'. The 'Encryption Options' section includes 'Compatibility' set to 'Acrobat 5.0 and later', 'All document contents' selected, and 'Document Open Password' set to '****'. The 'Output' section shows 'Output PDF - (variable)' set to 'outPDF'. A red box also highlights the 'Conditional test' section on the left, which contains the following logic:

```

if (myTest==0) {email}
else {writeDocument}

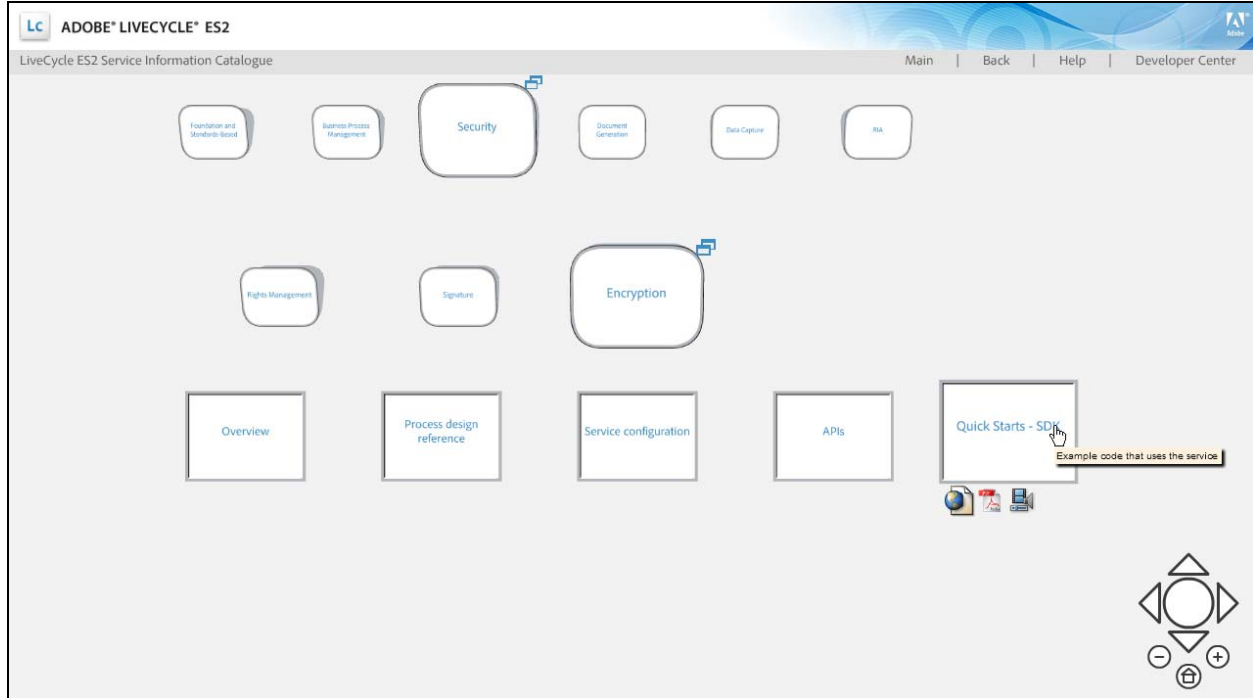
```

Below the canvas, the 'Main Branch' table lists the variables used in the process:

Name	Type	Input	Output	Required
inPDF	document	✓		✓
myTest	int	✓		
outPDF	document		✓	

The 'Variables' section on the right lists the variables: inPDF, myTest, and outPDF. The 'Settings' section on the right lists the settings: myTest, myTest, and myTest.

APPENDIX C: Docmap



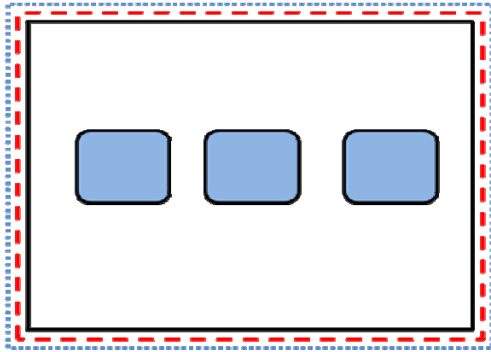
In this screenshot of the docmap, the user went down the tree and selected the “Quick Starts - SDK” topic. This topic has 3 attachments: an HTML link, a PDF file and a video. By clicking on the appropriate icon, a new window will open with the information.

We can also see the navigation panel at the lower right which allows the user to move the camera around using the arrows. The plus/minus icons can zoom in and out and a home button is provided so if ever the user as lost themselves and can no longer find the folders, the home button will bring them back to the original view of the 6 top level folders.

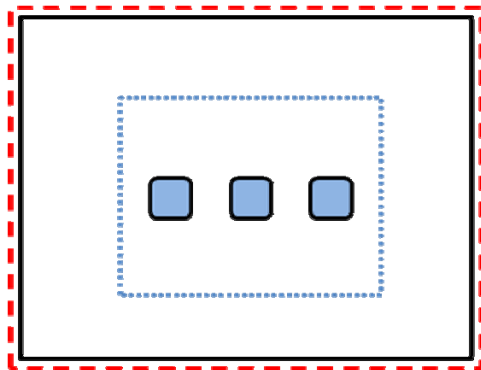
This user interface is a second iteration and provides a very clean looking window. It follows the general LiveCycle theme and keeps a simple business look. Most colors in the docmap are editable from the XML file. For example, some users have mentioned that the blue text is difficult to read and would prefer black. Having a `<font_color>` tag in the xml made for a very simple change.

APPENDIX D: Viewport manipulation

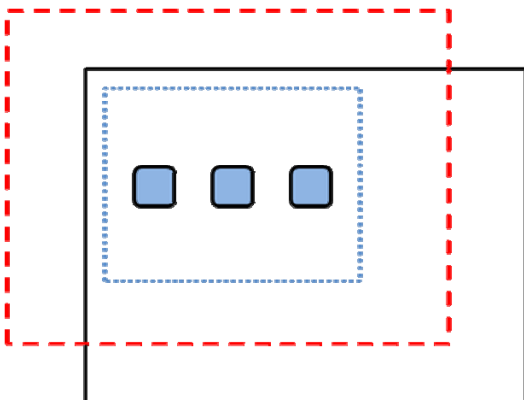
- Browser window
- - - - Papervision3D viewport
- Screen



The browser window and viewport are both at full screen, camera is at regular position.

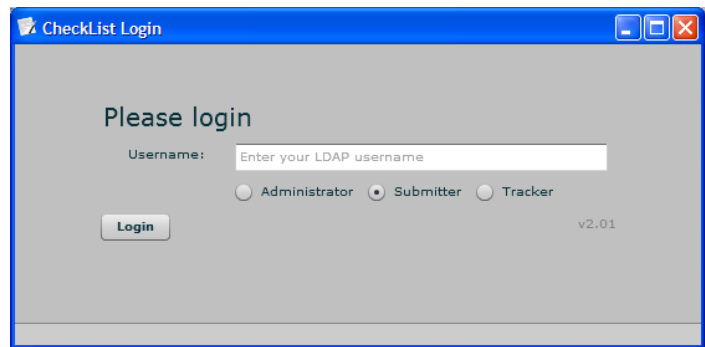


The browser window is smaller but the viewport always remains the size of the screen. The camera is zoomed out to make the folders smaller.

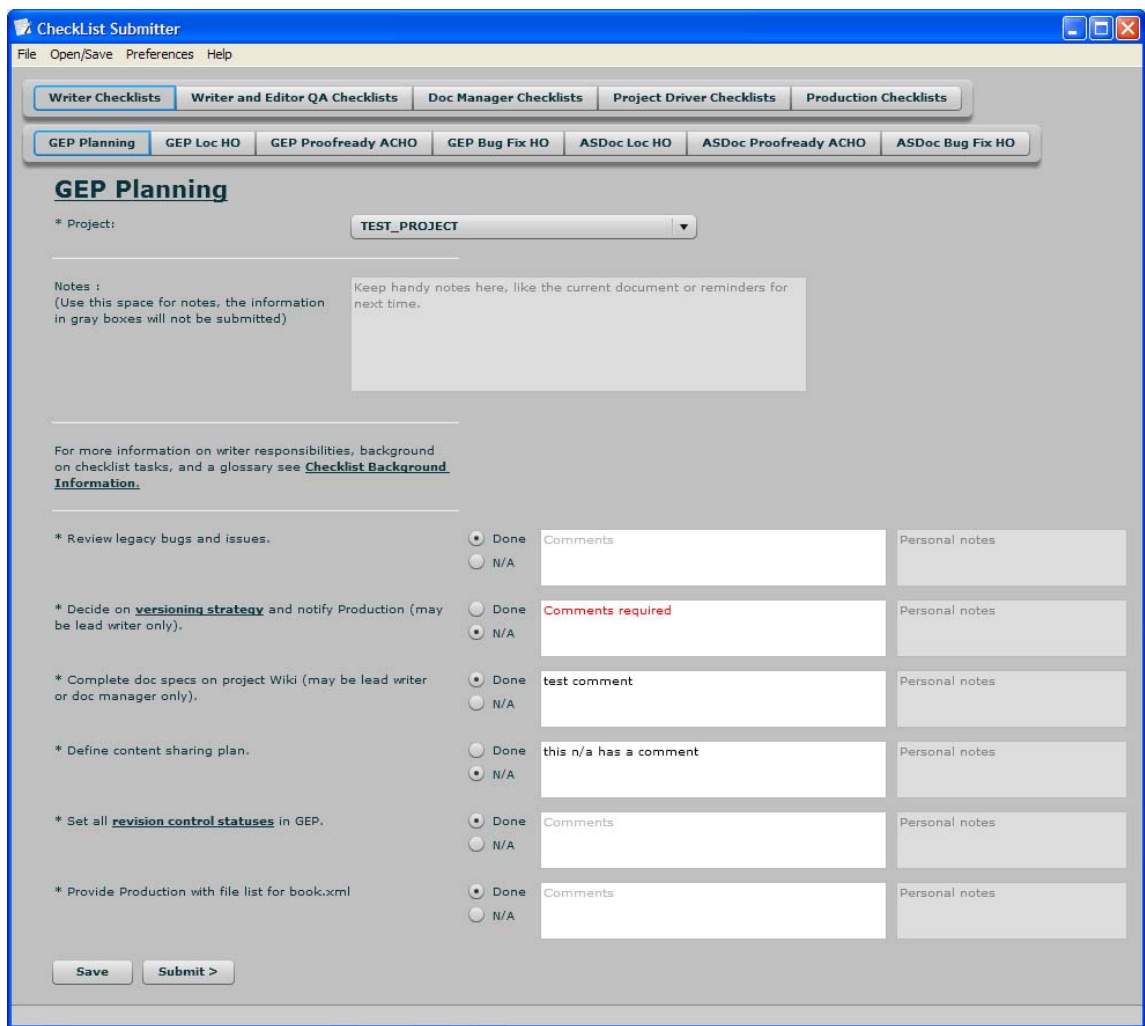


The browser is smaller and off center, so the viewport is shifted to keep the folders centered on the browser. The camera is zoomed out to make the folders smaller.

APPENDIX E: Checklist app



The Checklist Login window has a blue title bar with the text "CheckList Login" and standard window controls. The main area is gray and contains the text "Please login". Below this is a "Username:" label followed by a text input field with the placeholder "Enter your LDAP username". Underneath the input field are three radio buttons labeled "Administrator", "Submitter" (which is selected), and "Tracker". A "Login" button is positioned to the left of the "v2.01" version number in the bottom right corner.



The Checklist Submitter window features a blue title bar with "CheckList Submitter" and standard window controls. It includes a menu bar with "File", "Open/Save", "Preferences", and "Help". Below the menu bar is a series of tabs: "Writer Checklists", "Writer and Editor QA Checklists", "Doc Manager Checklists", "Project Driver Checklists", and "Production Checklists". Under these are more specific tabs for "GEP Planning", "GEP Loc HO", "GEP Proofready ACHO", "GEP Bug Fix HO", "ASDoc Loc HO", "ASDoc Proofready ACHO", and "ASDoc Bug Fix HO". The "GEP Planning" tab is active, showing a "Project:" dropdown menu set to "TEST_PROJECT". Below this is a "Notes:" section with a text area and a warning that gray boxes are not submitted. A link for "Checklist Background Information" is provided. The main task list includes items like "Review legacy bugs and issues", "Decide on versioning strategy", "Complete doc specs on project Wiki", "Define content sharing plan", "Set all revision control statuses", and "Provide Production with file list for book.xml". Each item has radio buttons for "Done" and "N/A", a "Comments" text field, and a "Personal notes" text field. At the bottom are "Save" and "Submit >" buttons.

Please note: I am only showing the submitter here, but there is also a working Administrator app (formerly the Manager) and a new Tracking app (not discussed in this report). Both are selectable from the login screen.

APPENDIX F: Invoking a LiveCycle process - Sample

```
//Create the remoting object "GetProjectService" to talk to LiveCycle
<mx:RemoteObject id="GetProjectsService" destination="GetProjects"
                  result="resultHandler(event);">
    <mx:method name="invoke" result="handleGetProjects(event)"/>
</mx:RemoteObject>

.....

//=====//
// GET PROJECTS //
//=====//
private function getProjects():void
{
    //Set the channel properties for GetProjectService
    var cs:ChannelSet= new ChannelSet();
    cs.addChannel(new AMFChannel("remoting-amf", "http://" + AdminApp.serverPort +
                                "/remoting/messagebroker/amf"));
    GetProjectsService.setCredentials("administrator", "password");
    GetProjectsService.channelSet = cs;

    //Set the MySQL database parameter
    var params:Object = new Object();
    params["dbUrl"] = AdminApp.dbUrl;

    //Invoke the LiveCycle ES Process
    var token:AsyncToken;
    token = GetProjectsService.invoke(params);
}

private function handleGetProjects(event:Event):void
{
    //get the XML file that was outputted by the process
    var projectsXml:XMLList = event["result"]["projects"].project;
    var projectsData:ArrayCollection = new ArrayCollection();

    //Populate the dataProvider for the project list from the XML.
    for(var i:int = 0; i < projectsXml.length(); i++)
    {
        var obj:Object = new Object();

        obj.label = projectsXml[i];
        obj.docManager = projectsXml[i].@docManager;
        obj.projectDriver = projectsXml[i].@projectDriver;
        obj.productionLead = projectsXml[i].@productionLead;
        obj.others = projectsXml[i].@others;

        projectsData.addItem(obj);
    }

    projList.dataProvider = projectsData;
}
```

This is an example of calling LiveCycle from Flex. We declare a RemoteObject that is mapped to our process, and set some properties to create the connection. We create an array of parameters to pass to the invocation. When the invocation completes, the handleGetProjects function is called (as declared in the RemoteObject). Using the event parameter, we can fetch the output variables and treat them however we like. In this case we add the list of projects to an array collection which is used as the data provider of a drop down list (projList).

SUPERVISOR'S APPROVAL

As supervisor of CO-OP student XXXXXXXX, I, _____, certify that, to the best of my knowledge, this report is entirely the student's work and is free of confidential information to the extent that it can be read by university faculty members.

Signature _____

Date _____