CrunchTime Data Pump: Design and System Development

Apr 24, 2013

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Bachelor of Science degree

Chenchen Zhang

Luyang Zhang

Advisor
Professor Elke Rundensteiner
Department of Computer Science
# Table of Contents

Abstract ........................................................................................................................................... 4
Acknowledgements .............................................................................................................................. 5
Authorship Page .................................................................................................................................. 6
Table of Figures ..................................................................................................................................... 7
List of Tables ......................................................................................................................................... 8
Executive Summary ............................................................................................................................... 9

Chapter One: Introduction .................................................................................................................. 11
  1.1 CrunchTime Overview .................................................................................................................. 11
  1.2 Introduction to CrunchTime Data Pump ....................................................................................... 11
  1.3 CrunchTime Data Pump Business Needs ....................................................................................... 12

Chapter Two: Background ................................................................................................................... 13
  2.1 Running Java Application as Windows Service .............................................................................. 13
      2.1.1 Apache Commons Daemon .................................................................................................. 13
      2.1.2 Java Service Wrapper ......................................................................................................... 14
  2.2 Apache Maven .................................................................................................................................. 15
      2.2.1 Project Administration ....................................................................................................... 15
      2.2.2 Comparison with Other Tools ........................................................................................... 15
      2.2.3 Maven’s Use in CDP .......................................................................................................... 17
  2.3 Apache Camel .................................................................................................................................. 18

Chapter Three: Methodology ................................................................................................................ 19
  3.1 Requirement Analysis ................................................................................................................. 20
  3.2 Design Document & Prototyping .................................................................................................. 20
  3.3 Iterations, Demonstration & Feedback ........................................................................................ 22
  3.4 Identify Defects & Resolve Bugs ................................................................................................ 22
  3.5 Production & Technical Support ..................................................................................................... 23
  3.6 Project Timeline ............................................................................................................................ 23

Chapter Four: Design & Implementation ............................................................................................. 26
  4.1 Phase R&D: FTP/SFTP vs. HTTP/HTTPS .................................................................................... 26
  4.2 Phase 1: CDP Service Program .................................................................................................. 27
  4.3 Phase 2: CDP Automatic Update Program .................................................................................. 29
      4.3.1 CDP Updater as Windows Service ....................................................................................... 29
      4.3.2 CDP Updater Optional ....................................................................................................... 29
4.3.3 Windows Application Log

4.4 Phase 3: Applying Business Logic to CDP Service Program

4.4.1 Apache Camel & File Routing

4.5 Test Driven Development

4.5.1 Test Objects

4.5.2 Test Deliverables

4.5.3 Test Procedure

4.5.4 Test Environment

Chapter Five: Analysis & Evaluation

5.1 The Chidamber & Kemerer Metrics

5.1.1 Weighted Methods per Class (WMC)

5.1.2 Coupling Between Objects

5.1.3 Responses for Classes

5.2 Potential Improvements

5.2.1 Possible Improvements for CrunchTime Data Pump

5.2.2 Improvements for CDP Updater

Chapter Six: Conclusion

References

Appendix

Appendix A: Time Budget for Project Specification

Appendix B: Quality Assurance Test for CDP & CDP Updater
Abstract

In this project we designed the new-generation CrunchTime Data Pump (CDP). CDP is a software program that pushes and pulls flat files to and from the CrunchTime Net-Chef clients to the Net-Chef server application for the purposes of loading and extracting sale statistics data from local clients. Such data is used by CrunchTime to provide business solutions to its clients. The new CDP program, which will replace the old-generation Delphi version, is a Java application implemented to run as a Windows Service. It enjoys the advantages of better maintainability and user control, logging and error handling. As additional feature, it has also been enhanced with the capabilities of auto-update and installation of newer software releases of the CDP system without requiring client input.
Acknowledgements

Our group would like to acknowledge many people for the opportunity to take part in the real-world project and the experience learned from it.

First, we are deeply grateful for our professor Elke A. Rundensteiner and PhD. Student Chuan Lei from Worcester Polytechnic Institute for all of their input and support.

Last but not least, we want to acknowledge our sponsor at Crunchtime! Information System, Jim Krawczynski for introducing us to this project, and providing us with ample guidance and support during this project.
Authorship Page

Chenchen Zhang and Luyang Zhang contributed equally to the writing of this report, to researching and executing the methodology, and to designing, developing and testing the final software deliverable.
**Table of Figures**

Figure 1: POM.xml for CDP ................................................................. 17

Figure 2: CDP Development Cycle ...................................................... 19

Figure 3: CDP Overview ..................................................................... 20

Figure 4: CDP Flow Diagram ............................................................... 21

Figure 5: CDP First Iteration ............................................................... 24

Figure 6: CDP Second Iteration ........................................................... 25

Figure 7: Windows Service Installation .................................................. 47

Figure 8: Service Panel View of CDP ..................................................... 48

Figure 9: Service Panel View of CDP Updater ....................................... 48

Figure 10: CDP Version# in Regedit ...................................................... 49

Figure 11: Windows Service Uninstaller ............................................... 49

Figure 12: CDP Server Directory .......................................................... 51

Figure 13: CDP Local Directory ............................................................ 52

Figure 14: CDP Updater Newer Version ............................................... 54
List of Tables

Table 1: HTTP Analysis

Table 2: FTP Analysis

Table 3: CDP Installation Batch

Table 4: Weighted Methods Per Class for CDP

Table 5: Coupling Between Objects for CDP

Table 6: Response for a Class for CDP
Executive Summary

As a global leader in strategic enterprise solutions for the restaurant, cruise, foodservice and hospitality industries, Crunchtime! Information Systems strives to provide modernized application software to cater the business needs of its clients and customers.

The main goal of the WPI project team is to redesign the next-generation Crunchtime Data Pump, which is a proprietary software application of Crunchtime! Information Systems. The old version of CDP has been written in Delphi (a Pascal-based language) about 10 years ago, so our objective was to undertake a new redesign using Object-Oriented principles and state-of-art technologies. In addition we were also charged with creating a corresponding updater application “CDP Updater” to automate the process to install new releases of the CDP software without human intervention. Both applications were designed, set up and launched as windows service on the clients’ side with minimum requirement for user interaction and maintenance.

The first objective is to migrate the code base from Delphi to Java, so that thereafter the project could be managed and deployed in a more systematic and consistent manner. Object-oriented design concepts and the agile development cycle were to be utilized as they satisfy the industry standard in modern software engineering. Various open source tools and packages have been utilized to achieve this expectation, including Apache Common Daemon, Apache Maven, Spring Framework, and Apache Camel.
The second part of the project focused on providing an automated windows service to update the “Crunchtime Data Pump” application hosted on the clients’ side, whenever a new release of CDP becomes available on the server of Crunchtime! Information Systems. As the original Delphi based system did not have this capability available, our team had to design and determine the details of the auto-update software architecture, research possible designs and tool choices, and in that process this exploration was essential as not all details were covered in the specification by our customer Crunchtime.

During the final stage of the project, the WPI project team documented the quality assurance procedure for the two CDP applications by introducing the appropriate test environment setup, and demonstrating application performance. The project code base was also evaluated based on one of the industry standard mechanisms, namely, the C&K metrics. Potential improvements for this project include, but are not limited to, better mechanisms for failure reporting, and robust web services for client usage reporting and recording.
Chapter One: Introduction

1.1 CrunchTime Overview

CrunchTime! Information Systems is a company founded by Bill Bellissimo in 1995. Currently it has about 50 to 60 employees in its Boston headquarter. Its main goal is to provide strategic enterprise solutions to its clients, especially those in the restaurant, cruise and hospitality industry. (Crunchtime Overview)

Specifically, CrunchTime provides high-quality technologies and solutions to help its clients to reduce food and beverage costs, drive labor efficiencies, and better manage the quality and consistency of their food service operations. Among other tasks, CrunchTime tracks the real-time inventory of items from ordering to avoid their depletion for restaurants. Based on the collected data, CrunchTime will build a business model, which can help to predict which recipe will be popular and more profitable. Currently a software client needs to be manually installed in each restaurant to collect and send relevant information to the central database. This requires Crunchtime staff significant time either on site or via telephone support. (CrunchTime! Solutions Overview)

1.2 Introduction to CrunchTime Data Pump

The CDP is a software program that is used to pull files from local CDP windows services hosted at clients’ side and then push these files to the CrunchTime Net-Chef application hosted at Crunchtime! Information Systems for the purpose of data processing by interacting with database.
The CDP is currently implemented using Delphi, and available either as an interactive client program or as a windows service installation.

This CDP program is deployed to individual restaurant locations. It then monitors files in certain designated folders on the local operating system. When a specific file is detected by CDP, the program will upload the file to the Net-Chef server, which, upon successful parsing and validating the file, will transform the data and load it into the server database.

1.3 CrunchTime Data Pump Business Needs

The WPI team needs to rewrite the CDP client from Delphi into Java or .NET. CrunchTime no longer is supporting Delphi as a primary development language. In addition, the new software needs to be provided with automatic upgrade capability so it can detect new releases of the actual CDP software published by CrunchTime, download the new version and install it without end user interaction. The new program should also run solely as a windows service.

The CrunchTime client base is growing and so is in turn the number of restaurant locations each client has. The clients are only running the CDP software as a service and not via its web interface because a service can be set to start automatically on reboot. This significantly reduces the IT administrative costs in reacting to unplanned outages since each restaurant location is typically maintained by either external or their own IT department. This same administrative burden is placed on the corporate IT department each time CrunchTime releases a new version of the CDP software. The requirement thus is to solve this by automating the whole process.
Chapter Two: Background

During the implementation of the CDP client service and CDP automatic upgrade service, the team researched, learned about and then deployed several open-source tools and libraries, including Apache Common Daemon, Apache Maven and Apache Camel to help manage and build the project.

2.1 Running Java Application as Windows Service

2.1.1 Apache Commons Daemon

Apache Commons Daemon is a software application that can be used to run executables or java applications as windows service or as a UNIX daemon. It is made of 2 parts. One written in C that interfaces to the operating system and the other in Java that provides the Daemon API. By using Apache Commons Daemon in our project, the team can turn the CDP client service from a Java application into a windows service. On the tomcat apache homepage we download “Procrun”, which is a set of binaries and applications that contain two executable files: “prunsrv.exe” and “prunmgr.exe”. The “prunsrv” is a native Windows binary that runs as a windows service and starts up an embedded JVM to run your Java program. The “prunmgr” is a GUI application to start/stop and configure the service. By default, “Procrun” is configured to call the main method in your Java project. Hence, it is appropriate that the main method could decide to start the service or stop the service based on the arguments provided by users. The standard folder hierarchy for deployment is the following:

C:\MyService
Notice that “prunsrv.exe” has been renamed to “myService.exe” by the WPI project team. When we check the running windows service through the control panel, it will be listed as “myService.exe”.

2.1.2 Java Service Wrapper

Java Service Wrapper makes it possible to run a Java application as windows service by packaging service wrapper binaries and libraries into the java application directory. The Java application then can be launched as a windows server either by using command line or a batch file that contains the command argument for the service, such as start, stop, install, uninstall, etc. Our project team has decided to use this approach because it is more client-weighted. That is we do not need to bother about implementing any specific service functionalities such as start and install. Therefore it requires minimum disruption of the Java program.
2.2 Apache Maven

2.2.1 Project Administration

Apache Maven is a software project management and development tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from configuration. A project object model covers the full configuration of a single project, such as the project’s name, owner, and dependencies on other projects. For large projects that consist of several sub-projects, one can also have independent POM for each sub-project and then compile the whole project using a root POM. (Srirangan, 2011)

Another important feature that is core to the administration of project is the dependencies. Maven's dependency-handling mechanism is organized around a coordinate system identifying individual artifacts such as software libraries or modules.

2.2.2 Comparison with Other Tools

Apache Ant is another tool with a similar functionality to that of Maven. Both of them are widely used in industry to manage the projects.

However, there is one fundamental difference between Maven and Ant. Namely, the design of Apache Maven requires that all projects should have a fixed structure with certain group of components such as getting resources from source control, compiling the project, and unit testing. While most software projects in concept support these operations and indeed do have a well-defined structure, this practice may still pose a
challenge if some developers don’t follow these prescribed guidelines. Maven requires that this structure and the operation implementation details be defined in the POM file. Thus, Maven relies on a convention on how to define projects and on the list of workflows that are generally supported in all projects.

This design constraint resembles the way that an IDE handles a project. It provides many benefits, such as a succinct project definition, and the possibility of automatic integration of a Maven project with other development tools such as IDEs, builds servers, etc.

But one drawback to this approach is that Maven requires a user to first understand what a project is from the Maven point of view, and how Maven works with projects. The reason is that process flow of Maven is not immediately obvious just from examining the Maven project file. In many cases, this required structure is also a significant hurdle in migrating a mature project to Maven, because it is usually hard to adapt from other approaches.

In Ant, projects do not really exist from the tool's technical perspective. Ant works with XML build scripts defined in one or more files. It processes targets from these files and each target executes tasks. Each task performs an operation such as running a compiler or copying files. Targets are executed primarily in the order given by their defined dependency on other targets. (Holzner, 2005) Thus, Ant is a tool that chains together targets and executes them based on inter-dependencies and other conditions.
The benefits provided by Ant are numerous. It has an XML language optimized for clearer definition of what each task does and on what it depends. Also, all the information about what will be executed by an Ant target can be found in the Ant script.

2.2.3 Maven’s Use in CDP

In the implementation of CDP client service and CDP automatic update service, the team used Apache Maven based on Eclipse to effectively manage the CDP project, its library dependencies and to integrate all of its components.

Here is a picture of how the POM file looks like in CDP.

Figure 1: POM.xml for CDP
The team finally decided to adopt Maven for project administration because of the significant advantages associated with Apache Maven. For the team as developers, it makes the building process much easier by integrating every step such as compiling and building path of the libraries, and providing a uniform build system. For any users who want to build on the existing project, it provides quality project information and allows transparent migration to new features.

2.3 Apache Camel

In this project the team was required to use Apache Camel to manage the dataflow between server and client. The flow of data should be represented as standard Camel routes. Instead of implementing from scratch, it is also highly recommended that the team implement the routing functionality by using the enterprise integration pattern supported by Apache Camel. The latter is a small library with minimal dependencies for easy embedding into any Java application. Another advantage is that Apache Camel can support type-safe smart completion of routing rules in an integrated development environment using regular Java code without requiring XML configuration files, though XML configuration inside Spring is also supported.
Chapter Three: Methodology

The team strictly has applied the agile development cycle as methodology. Within this overall methodology, we divided our project into three main phases. That is, we developed phase of the core project goals in specification in a test driven development approach by writing the unit tests first then doing the development of the feature.

The whole process was started by our sponsor Crunchtime Information System, providing us with some initial specification. The WPI project team was responsible for analyzing the requirement and scratching an initial UML design, and then implemented the program thereafter. Once the WPI project finished documenting and testing the deliverable for the performance phase, then their feedbacks are used as input for the next iteration. Both requirement of the test delivered result as well as the kick-off of the next core features to be tackled by the next phase in the project.

Software Development Cycle

![Software Development Cycle Diagram]

Figure 2: CDP Development Cycle
3.1 Requirement Analysis

The team initiates the project by first identifying the customer needs as stated in the document “Feature Specification – CDP – Client and Service Migration and Enhancements”. A time budget for each feature is estimated and then assigned to each task, after discussion between the team members. Since the total time WPI has for the CDP project is approximately 21 weeks, the team split the budget according to the level of complexity for each feature. For example, features such as setting up the upload/download polling interval will be assigned less time compare to features such as realizing the actual upload/download functionality.

3.2 Design Document & Prototyping

A high-level infrastructure of the CDP and CDP updater is depicted in Figure 3.
The WPI project team developed a UML design to address the business requirements summarized from the infrastructure above. Figure 3 is a flow diagram that illustrates how the CDP windows service is instantiated and how it interacts with the CDP updater windows service.

In Figure 4, once a windows service is started, it will call the main CDP functionality by passing a configuration file to it, and simultaneously start a socket server that is used to interact with CDP updater. The main CDP service, after correctly parsing the configuration file, we will launch the transfer and archival functions accordingly.

Figure 4: CDP Flow Diagram
3.3 Iterations, Demonstration & Feedback

The implementation process partly followed the test-driven development principle, as each iteration is initiated by creating a set of automated test cases, usually as JUNIT tests, that define a desired improvement or function. The team then places the minimum amount of code to pass the test, and refactors the code based on the widely accepted industry standards. (Kemerer, 1994) However, due to the special nature of the CDP windows service, whose main functionality is file transmission between the local OS folder and remote server, certain functionalities are infeasible to be examined using JUNIT test.

The WPI project team had several follow on meetings with Crunchtime Inc. representatives to discuss about deviations of our implementation from the expectation. Constructive feedbacks are collected after each meeting for further development.

3.4 Identify Defects & Resolve Bugs

As a windows service that handles file routing between the local host and the remote server, the chances that defects and bugs will occur inevitably become larger. Behaviors of the CDP windows service when the server stops working, a firewall blocks the file transmission, a local folder does not have the permission to write/delete are recorded and carefully evaluated. Potential defects and bugs such as unsynchronized access to shared variables, and failure to report disconnection are identified and foreseen by the team in an early stage during development.
3.5 Production & Technical Support

Because of the complicated nature of setting up a windows service, and of applying all the configuration settings for the CDP windows service, the WPI project team has composed a document “CDP - Developer's Guide” to help developers at Crunchtime Inc. with setting up the test environment, performing the JUNIT test and lastly conducting the smoking test before releasing the application into production. (Kaner, 2002)

After the CDP service has gone through the smoking test conducted by the developers, a second document was produced to serve as a user guide. This aims to ease the usage of CDP windows for customers and accommodates any necessary technical support information.

3.6 Project Timeline

During the first iteration, the WPI project team meets with our sponsor and strived to gain a basic understanding of the business importance of CrunchTime Data Pump after viewing a live demo of the old CDP windows service and inputting with an explanation of this service by our sponsor. After that, the sponsor handed us a detailed specification. The team then conducted background research and undertook a literature review to identify the proper tools necessary for the project. A skeleton windows service that could start and terminate gracefully and an UML design are presented to our sponsor by the end of first iteration.
The second iteration started by thorough analysis of the detailed feedbacks provided by the sponsor on the first deliverable. The WPI project team then distributed and reassigned the remaining time budget based on the priority as well as newly understood complexity of the unfinished tasks. An agenda with a more detailed 7-week timeline for the second iteration was then proposed. In addition at this stage, we summarized our key question using a written document. The sponsor was always punctual in offering step by step responses.
Project Timeline

- **Phase 2 & Phase 3**

  - Get feedback of last deliverable from Jim
  - Send in questions through email
  - Get clarification from CrunchTime
  - Hand in the second deliverable

  According to the received feedback, make agenda for tasks to be completed in C term
  Background research on which tools to use in our project

*Figure 6: CDP Second Iteration*
Chapter Four: Design & Implementation

4.1 Phase R&D: FTP/SFTP vs. HTTP/HTTPS

This phase focused on the team performing an evaluation on which transfer protocol is best suited for the project. The team explains the pros/cons of each option and then produces this analysis for a final decision from CrunchTime. This document was created using HTTP/S because that is what the current application does. This doesn’t necessarily mean it is the best way to accomplish the goals.

HTTP/HTTPS (Shiflett, 2003)

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real time data interaction</td>
<td>Implementation could be time consuming</td>
</tr>
<tr>
<td>Elimination of redundant data</td>
<td></td>
</tr>
<tr>
<td>Many ways to secure the transport, like https, security certificates</td>
<td></td>
</tr>
<tr>
<td>Allows soap messages to contain additional metadata</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: HTTP Analysis

FTP (Mary Ann Pike, 1995)
### Pros vs Cons

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simpler and well-established legacy protocol</td>
<td>Need to take extra steps to ensure the file is transported securely</td>
</tr>
<tr>
<td>Good to interoperate with some external third party facilities, such as document exchange</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: FTP Analysis

### 4.2 Phase 1: CDP Service Program

This phase focused on creating the CDP service software architecture that incorporates the use the CDP.INI settings, validation of the CDP.INI settings, error logging to log files and Windows Application Log and executing as a Windows Service to simply write a message to the log file.

The WPI project team used the open source toolkit, Apache Common daemon rather than Java service Wrapper, which is a charged third party software, to establish the java application as a windows service.

Below we display an installation batch file utilized to set up the CDP windows service:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisplayName</td>
<td>CrunchTime Data Pump (CDP)</td>
</tr>
<tr>
<td>Description</td>
<td>Used for uploading and downloading data to CrunchTime Net-Chef and Enterprise Manager. Contact CrunchTime Customer Service for additional information.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>StartClass</strong></td>
<td><strong>Startup</strong></td>
</tr>
<tr>
<td><strong>StartMethod</strong></td>
<td><strong>main</strong></td>
</tr>
<tr>
<td><strong>StartParams</strong></td>
<td>C:\CDP_HOME\CDP.ini</td>
</tr>
<tr>
<td><strong>StopClass</strong></td>
<td><strong>Startup</strong></td>
</tr>
<tr>
<td><strong>StopMethod</strong></td>
<td><strong>Stop</strong></td>
</tr>
<tr>
<td><strong>Classpath</strong></td>
<td>C:\CDP_HOME\CrunchtimeDataPump.jar ; C:\CDP_HOME\lib*</td>
</tr>
<tr>
<td><strong>LibraryPath</strong></td>
<td>C:\CDP_HOME\lib\</td>
</tr>
</tbody>
</table>

Table 3: CDP Installation Batch

Some of parameter values are defined as: (Apache Commons Daemon, 2013)

**StartClass**: Class that contains the startup method. Applies to the jvm and Java modes.

**StartMethod**: Name of method to be called when service is started. It must be static void and have argument (String args[]). Only applies to jvm mode - in Java mode, the main method is always used.

**Start Params**: List of parameters that will be passed to either StartImage or StartClass. Parameters are separated using either the “#” or “;” character.

**StopClass**: Class that will be used on Stop service signal. Applies to the jvm and Java modes.

**StopMethod**: Name of method to be called when service is stopped. It must be static void and have argument (String args[]). Only applies to jvm mode. In Java mode, the main
method is always used.

ClassPath: Set the Java classpath.

LibraryPath: Directory added to the search path used to locate the DLLs for the JVM. This directory is added both in front of the PATH environment variable and as a parameter to the SetDLLDirectory function.

4.3 Phase 2: CDP Automatic Update Program

In second phase our team focused on creating the CDP automatic update program and corresponding hosted update application.

4.3.1 CDP Updater as Windows Service

The automatic update program is designed to run as a windows service. The program displays in the Windows Services panel as “CrunchTime CDP Automatic Updates”. The above business requirement is addressed by modifying the “DisplayName” and “Description” part in CDP Updater installation batch file. Developers could customize these options in the install batch to reflect any changes or modifications in the business requirements.

4.3.2 CDP Updater Optional

The automatic update program may not be used by all clients. Some client may not have access to the Internet, while other clients may want to administrate the software upgrades themselves. The creation of the update process should thus take into account that the CDP Service program may be installed independently from the automatic update program.
Because of this business need, our team has created separated installation batch files for both the CDP and the CDP Updater windows service, so that the both services could be installed independently from each other.

4.3.3 Windows Application Log

The CDP automatic updater will keep a processing log to record error messages and certain aspects of its processing. In addition to this log file, the Windows Application Log will be used for all error messages.

The log file will be created in the CDP install location and the log file name will be CDP_Automatic_Update.log.

Using Log4j, the CDP updater addresses this business requirement in the specification. However, the default logging mechanism of Apache Common Daemon conflicts with the usage of Log4j. We found that the log file could be successfully generated within the IDE, but not in the install location while CDP updater is running as a windows service.

The requirement for the windows application log is accomplished by using the log4j toolkit. However, due to the conflict with Apache Common Daemon Default logging, windows application log cannot be properly generated for CDP Updater windows service. One of the potential improvements will be to solve this conflict.

4.4 Phase 3: Applying Business Logic to CDP Service Program

This third phase focused on adding functionality to the CDP Service Program that will perform the upload and download functionality within a Net-Chef application. Instead of modifying the Net-Chef application directly, the team created their own web service to
accept and store the uploaded file and a second web service to download the same file back to the machine the CDP service program is running from. The uploaded file and downloaded file should have no difference when compared by using a visual diff tool or when comparing their checksums.

4.4.1 Apache Camel & File Routing

In order to implement the file routing functionality, the WPI project team decided to take advantage of the FTP component and FILE component within Apache Camel. (Claus Ibsen, 2011) These two components are basically used to convert the local directory and remote server url specified in the CDP configuration file into the corresponding endpoints.

The detailed requirements for file transmission, such as encoding, polling interval, server authentication, can be specified by modifying the available options for each components.

4.4.2 Camel File Component

The Camel File component provides access to file systems; allowing files to be processed by other Camel Components or messages from other components can be saved to disk. (Apache Camel: File) Since local directory paths are specified in the CDP configuration as “upload” and “download”, we used camel file components to construct the directory paths as endpoints for routing.

The general URI format to construct an endpoint for file routing is the following:

file://fileOrDirectoryName[?options]
In order to address specific business needs in the project specification, we have harnessed the following options for the Camel File component:

- **delete**: If true, the file will be deleted when it is processed (the default is to move it, see below)

- **delay**: milliseconds before the next poll of the file/directory

- **charset**: this option is used to specify the encoding of the file, Camel will set the Exchange property with Exchange.CHARSET_NAME with the value of this option. Exchange property here is one of the options that is available when user establishing a Camel route.

### 4.4.3 Camel FTP Component

The Camel FTP component provides access to remote file systems over the FTP and SFTP protocols. (Apache Camel: FTP) Since the url of Net-Chef server is specified in the CDP configuration file, with corresponding “userid” and “password” for authentication, we utilized the camel ftp component to establish the Url as endpoint for routing.

In order to use Camel FTP component in a Maven project, users have to add the following dependencies to the POM.XML:

```xml
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
</dependency>
```
The general URI format to construct the endpoint for a file routing task is the following:

- ftp://[username@]hostname[:port]/directoryname[?options]
- sftp://[username@]hostname[:port]/directoryname[?options]
- ftps://[username@]hostname[:port]/directoryname[?options]

username: Specifies the username to use to log in to the remote file system.

port: If no port number is provided, Camel will provide default values according to the protocol (ftp = 21, sftp = 22, ftps = 2222).

Directoryname: Represents the underlying directory and it can contain nested folders.

password: Specifies the password to use to log in to the remote file system.

4.5 Test Driven Development

The WPI team develops each phase in a test driven development approach by writing the unit tests first then doing the development of the feature. To the extent the unit tests can be automated they should be.

4.5.1 Test Objects

All the classes in the CDP project should be tested. Every public method in each class should be tested. In the project deliverable, the WPI project team has included a complete set of JUNIT tests for both the CDP and CDP updater. The test coverage reaches around 75% of the total code base.
4.5.2 Test Deliverables

Before the test begins, the source code should compile gracefully without error or warning. After the test, a copy of unit test report and problems reported would also become deliverable if necessary. In Appendix B, a comprehensive document for quality assurance is attached. The developers could use it as a guidance to run down the whole process briefly and identify any problems that may exist in the deliverable.

4.5.3 Test Procedure

Test cases must be written down, including both input and output values before the test. For classes that are not stand alone, a test program might need to be developed to invoke the method to be tested and stubs from other classes that are called by the class to be tested.

After the test, a test report is composed. The test specification as well as expected output and actual output should be documented in the report.

4.5.4 Test Environment

It would be ideal to run the test in the same native environment where CDP will be eventually deployed. The WPI project has thus set up a testing environment that simulate the actually deployment environment for the test purpose. The directory structure has been created based on the specification about both the clients’ side and server’s side setups. The clients’ side in the test environment is a local machine, while the server’s side is an FTP server with basic authentication functionality created by the project team.
Chapter Five: Analysis & Evaluation

5.1 The Chidamber & Kemerer Metrics

First introduced by Shyam R. Chidamber and Chris F. Kemerer in their paper “A Metrics for Object Oriented Design” (Kemerer, 1994), the C & K metrics have gained extreme popularity through the years. They have now become one of the most canonical ways for quantitatively analyzing industry standard software produced by object-oriented design. Six metrics are mentioned in the paper as: Weighted methods per class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Object Classes (CBO), Response for a Class (ROC), and Lack of Cohesion of Methods (LCOM).

(Chidamber & Kemerer object-oriented metrics suite) Because of the relatively small scale of both the CDP core and the CDP updater applications, it is unnecessary to apply all six metrics when evaluating the code base. We thus selectively used the following metrics in our context: Weighted methods per class (WMC), Coupling between Object Classes (CBO), Response for a Class (ROC), and Lack of Cohesion of Methods (LCOM).

5.1.1 Weighted Methods per Class (WMC)

Consider Class C1 with methods M1, M2, ..., Mn.

Let c1..cn be the static complexity of the methods.

WMC can indicate:

- the time and effort to develop and maintain
-impact on subclasses more (complex) methods will have a bigger impact

-reusability

More methods generally make the application more specific and thusly reduce the overall reusability. It is also obvious that more time needs to be devoted for maintain classes with more methods, and subclasses will experience significant impact upon changes of the parent classes.

For the sake of simplicity and without loss of generalization, we assume that complexity for each method is 1. After applying the weighted methods per class metrics to CDP, we have the following:

<table>
<thead>
<tr>
<th>Class</th>
<th>Weighted Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDP</td>
<td>1</td>
</tr>
<tr>
<td>StartUp</td>
<td>2</td>
</tr>
<tr>
<td>Configuration</td>
<td>19</td>
</tr>
<tr>
<td>Helper</td>
<td>5</td>
</tr>
<tr>
<td>Transfer</td>
<td>15</td>
</tr>
<tr>
<td>UploadAchivePrunning</td>
<td>4</td>
</tr>
<tr>
<td>Verbosity</td>
<td>2</td>
</tr>
</tbody>
</table>
As we observed from Table 2, although certain classes have methods far above the average, the average number of weighted methods per class is 6.25. This represents a fairly low density of methods. Thus there is a less chance of bugs and lower cost of maintenance.

5.1.2 Coupling Between Objects

Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. The uses relationship can go either way: both uses and used-by relationships are taken into account, but only once.

Multiple accesses to the same class are counted as one access. Only method calls and variable references are counted. Other types of reference, such as use of constants, calls to API, handling of events, use of user-defined types, and object instantiations are ignored. If a method call is polymorphic (either because of Overrides or Overloads), all the classes to which the call can go are included in the coupled count.

Table 3 represents the coupling between classes for our CDP program. As we can see, only several main classes are actually coupled to the some subclass that contains the concrete implementation for its functionality. Our implementation indeed has a low degree of coupling. Thus we can conclude that the quality of modular design is preserved, which is known to enhance the reusability.
Class | Coupled Class
--- | ---
CDP | Transfer, UploadAchivePrunning
StartUp | CDP
Configuration | Helper
Helper | None
Transfer | None
UploadAchivePrunning | None
Verbosity | None
Server | None

Table 5: Coupling Between Objects for CDP

5.1.3 Responses for Classes

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. RFC is simply the number of methods in the set. The RFC for CDP, based on Table 4 below, is approximately 8.6. This would be considered a relatively low degree of method complexity.

Class | Set of Methods
--- | ---
CDP | 3
Table 6: Response for a Class for CDP

<table>
<thead>
<tr>
<th>StartUp</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>20</td>
</tr>
<tr>
<td>Helper</td>
<td>5</td>
</tr>
<tr>
<td>Transfer</td>
<td>16</td>
</tr>
<tr>
<td>UploadAchivePrunning</td>
<td>16</td>
</tr>
<tr>
<td>Verbosity</td>
<td>3</td>
</tr>
<tr>
<td>Server</td>
<td>2</td>
</tr>
</tbody>
</table>

### 5.2 Potential Improvements

Although the project deliverable produced by the WPI team meets the basic criteria and satisfies the majority of business requirements stated in the CDP specification, there are still several places for future improvement.

#### 5.2.1 Possible Improvements for CrunchTime Data Pump

The current release of CDP can successfully route files between the local host and the remote server. However, the error handling and failure reporting for CDP could be further refined. According to the specification, errors and failures that occurred during the execution of the CDP windows service should be handled and logged into the windows application log.
The current release does not support the windows application log functionality outside of the IDE yet, due to the conflict caused by the Apache Common Daemon default log and log4j toolkit used in the project. One potential improvement would thus be to solve this conflict, so that error messages are correctly logged and visualized in the windows application log itself.

Another potential improvement of CDP would be to leverage the “locationid” setting in CDP configuration file. The purpose of this setting is to identify and track the usage of CDP updater, so that Crunchtime Information System could monitor and explore potential market needs.

5.2.2 Improvements for CDP Updater

By the end of our project, we have been able to finish the design and implementation of a functional CDP automatic updater. Our deliverable has the advantages of running as a windows service, which includes the ability to run without the user actually logged on. Besides, the feature of simplicity and stability is what is much desired by CrunchTime and its clients, many of who are not IT specialists.

However, much improve can still follow up on the server side of the updater, which is supposed to be managed by CrunchTime. It is out of the business need of CrunchTime that data analytics functionality can be implemented as web service on the CDP updater server. For example, all attempts to authenticate will be recorded in an audit table in the database supporting the automatic update hosted application. The audit table will store...
the information such as the location id of the client, which provides the identification as well as the timestamp to provide the identification.

**Chapter Six: Conclusion**

In this project we presented a new object-oriented design of the CrunchTime Data Pump (CDP) software system. CDP is a software program that pushes and pulls flat files to and from the CrunchTime Net-Chef clients to the Net-Chef server application for the purposes of loading and extracting sale statistics data from local client. Such data is used by CrunchTime to provide business solutions to its clients. Following standard software engineering practices, we implemented CDP and CDP updater as Java applications and utilized the tool Apache Common Daemon to run them as Windows Services. From the result of testing and performance analysis, we observe that the new CDP could bring several advantages over the original system. These benefits include:

1. Java is a modern programming language which is more popular as well as better supported than Delphi. There is an ongoing trend by CrunchTime to migrate their products to make them Java-based.
2. The new object-oriented design is considered to enjoy a higher maintainability than the previous design.
3. There is an added feature of automatic update of the new CDP. It helps clients of CrunchTime to better manage the CDP client software in their local stores, especially for those clients with no IT specialists on their staff.
There are some future directions where following work can be done to improve the performance of CDP client and its updater. First of all, more functions can be implemented on the update server running as web service so that CrunchTime! can collect the information about who has downloaded the latest version of CDP and when. Second, as is required by CrunchTime! company, the CDP and CDP updater don’t run with a GUI interface. On one hand, this feature makes the service simple and clear. On the other hand, it also means restaurant staff has to manually dig through the cumbersome log file to make everything is working correctly. One potential solution to it will be automatically sending the error report to a server managed by CrunchTime! after getting such permissions from the users.
References

_Apache Camel: File_. (n.d.). Retrieved Apr 2013, from Apache Camel:
http://camel.apache.org/file.html

_Apache Camel: FTP_. (n.d.). Retrieved Apr 10, 2013, from Apache Camel:
http://camel.apache.org/ftp.html

_Apache Commons Daemon_. (2013, Mar 28). Retrieved from Daemon: Procrun:
http://commons.apache.org/proper/commons-daemon/procrun.html


_Crunftime Overview_. (n.d.). Retrieved Apr 15, 2013, from Crunchtime:
http://www.crunchtime.com/about-us-overview.htm

http://crunchtime.com/solutions-overview.htm


Appendix

Appendix A: Time Budget for Project Specification

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement</th>
<th>Spec. Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Rewrite the CDP program in either Java.</td>
<td>4.1</td>
</tr>
<tr>
<td>2.</td>
<td>CDP program will be written to only run as a windows services.</td>
<td>4.2</td>
</tr>
<tr>
<td>3.</td>
<td>CDP program will be written to automatically apply updates to itself by checking a CrunchTime website for newer version and taking appropriate action.</td>
<td>4.3</td>
</tr>
<tr>
<td>4.</td>
<td>CDP program will maintain all user configurable preferences via CDP.INI file settings as it does in its current implementation.</td>
<td>4.4</td>
</tr>
<tr>
<td>5.</td>
<td>CDP program will support upload and download data transfers in a multi-byte Unicode character set (UTF8).</td>
<td>4.1</td>
</tr>
<tr>
<td>6.</td>
<td>CDP program will allow for the archiving of data files that were successfully uploaded in a fashion that will allow files of the same name not to overwrite each other.</td>
<td>4.3</td>
</tr>
<tr>
<td>7.</td>
<td>CDP program will provide a clean-up process to remove successfully archived files and a configured retention period.</td>
<td>4.7.14, 4.7.15</td>
</tr>
<tr>
<td>8.</td>
<td>CDP program will allow for the preservation of data files that fail</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>CDP program will have a logging mechanism that will record all upload and download attempts, success and failure messages, and a configurable level of debugging verbosity.</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>CDP program will have an authentication mechanism that will ensure the CDP program will interact only with the intended Net-Chef application.</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>CDP program will upload and download data through industry standard internet protocols.</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>CDP program will allow for both encrypted and un-encrypted upload and download of data during transmission via a configurable setting.</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>CDP program will allow the polling interval to check for files awaiting upload to be set via a configurable setting in seconds.</td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>CDP program will allow the polling interval to check for files awaiting download to be set via configurable setting in minutes.</td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>CDP program will allow the OS folder or UNC to be polled for files awaiting upload to be set via a configurable setting.</td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>CDP program will allow the OS folder or UNC that will be the destination for the downloaded files to be set via configurable setting.</td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>CDP program will have a multi level debugging verbosity to allow the execution of the program flow to be recorded for debugging purposes.</td>
<td></td>
</tr>
<tr>
<td>18.</td>
<td>CDP program will write errors in the application execution to the Windows Application Log.</td>
<td></td>
</tr>
<tr>
<td>19.</td>
<td>CDP program will have a configurable setting to define which client location the CDP program is executing from.</td>
<td></td>
</tr>
<tr>
<td>20.</td>
<td>CDP program will ensure that form/query data being sent over to the server are properly encoded to handle special characters through industry standard means.</td>
<td></td>
</tr>
<tr>
<td>21.</td>
<td>CDP program will allow the target Net-Chef URL to be set via a configurable setting.</td>
<td></td>
</tr>
<tr>
<td>22.</td>
<td>CDP program will ensure that more than one CDP program is not able to process files simultaneously in the event more than one CDP service program is running using the same the upload OS folder or UNC.</td>
<td></td>
</tr>
<tr>
<td>23.</td>
<td>CDP program will allow the OS folder or UNC that will be the destination for the downloaded EXPORT files to be set via configurable setting. If this value is set it will override the normal download destination.</td>
<td></td>
</tr>
<tr>
<td>24.</td>
<td>CDP program will allow the Labor Exports to be disabled by a single configurable setting.</td>
<td></td>
</tr>
<tr>
<td>25.</td>
<td>CDP program will allow the OS folder or UNC to archive successfully uploaded files to be set via a configurable setting.</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B: Quality Assurance Test for CDP & CDP Updater

1.1 Project Setup inside IDE

The current project is implemented solely in Java using Eclipse(Juno) J2EE and packaged as CDP.zip. Before import the project, please install the maven plugin for Eclipse IDE. After import the file into IDE, developer needs to run the maven install option so all the required maven dependencies will be downloaded and a jar file should be generated in “target” folder upon successful maven build.

1.2 Guide for Project Setup as Windows Service

After checking the project into the IDE, under the “service” folder, there are three files: CDP.exe, CDPw.exe, installService.bat. Please copy the jar file generated in “target” folder into the “service” folder, and rename the jar file as: “CrunchtimeDataPump.jar”. Configure the installService.bat file:

- **DisplayName**: The name of CDP service displayed in the windows service panel.
- **Description**: The description of CDP service displayed in the windows service panel.
- **StartClass**: Startup
- **StartMethod**: main
- **StartParams**: CDP.ini (Path pointed to the CDP.ini file)
- **StopClass**: Startup
- **StopMethod**: stop
- **Classpath**: CrunchtimeDataPump.jar (Path points to the jar); lib\* (Path points to the lib folder in the project directory)
LibraryPath: lib (Path points to the lib folder in the project directory)

Comments: Please make sure the install batch is working properly by trying to use absolute path first.

Service installation: Then run the windows command as administrator, and launch the installService.bat

![Figure 7: Windows Service Installation](image)

See Section 2.1 to set up the correct CDP configuration. Then start the service in windows service panel. In Figure 8 & 9, we can check the status of both services after they have been installed.
The registration information and version number regarding the installed CDP service can be viewed in regedit as illustrated in Figure 10:
Service uninstallation: Stop the CDP service in Windows Service panel by manually clicking stop option. Then execute “sc delete CDP” in the windows command (Figure 11).
2.1 Setting up CDP Configuration File

Under the project directory, locate the CDP.ini file and open it.

Here are all the configurable settings in the file:

“upload”: the local OS directory that stores the files uploaded to CDP server.

“download”: the local OS directory that stores the files downloaded from CDP server.

“uploadtimerseconds”: the polling interval for checking local upload OS path.

“downloadtimerseconds”: the polling interval for checking CDP server.

“url”: the url of CDP ftp server

“ssl”: configure protocol as either “ftps” or “ftp”

“verbosity”: the level of error message that will be logged into log4j file under “$project_home$/log/log4j/CDP.log”

“location”: currently not using this configuration

“userid”: the credential used for authentication with CDP ftp server

“password”: the credential used for authentication with CDP ftp server

“laborexport”: configure whether export files will be downloaded or not

“downloadexport”: the local OS directory that stores the export files downloaded from CDP server.

“uploadarchive”: the local OS directory that stores the files successfully uploaded to the CDP server.

“uploadarchiveprunningenabled”: configure whether pruning feature should be enable or not
“uploadarchiveprunningdays”: files in upload archive directory older than current time by this configuration will be deleted.

“uploadarchiveexecutiontime”: daily time when pruning feature will be executed.

“uploadarchiveexecutionfrequency”: frequency for executing pruning feature.

2.2 Set up CDP Client and Server

Set up FTP server as: “ftp://url/”. The “url” should be identical to the setting in the CDP.ini file.

Under the ftp server setup, create two folders called “download” and “upload”. Note that folder name is case sensitive.

The upload folder will receive all files uploaded from the local upload OS path. Files stored in the download folder will be downloaded to the local download OS path. Since authentication with FTP server will use “userid” and “password” settings in CDP.ini file. The server should be set up consistently.
The local OS folder is created in the following way:

Download: Corresponds to “download” setting in CDP configuration file.

Upload: Corresponds to “upload” setting in CDP configuration file.

Uploadarchive: Corresponds to “uploadarchive” setting in CDP configuration file.

Update: Directory where the latest release of CDP is stored after download from server.

### 2.3 Launching CDP Service

Then you can create couple test files in the local upload OS path to be uploaded to the server and several test files in the download folder under CDP FTP server. Thereafter, please build the project in the IDE, then right click project and run as java application.

While the service is running, users could expect to observe files in the local upload OS path to be uploaded to the upload folder on CDP server and Files in the download folder on CDP server to be downloaded to the local download OS path.
If the application does not behave as described, please check if the local OS folder has the permission to read and write. Second please also check the firewall setting, as file transfer might be blocked by the firewall.

3.1 CDP Automatic Update Service Architecture

Currently the CDP Automatic Updater Service is composed of three files, Client.java, Transfer.java and Startup.java. Each of these files is responsible for a different part of the functionality, as we will explain below.

Client.java

Client.java manages the Inter-process communication (IPC) with the CDP service. The main functionality of it is to make sure no download/upload job is running when software upgrade takes place.

Transfer.java

Transfer.java contains the class that manages the download of the latest CDP jar file from the ftp server.

Startup.java

Startup.java corresponds to the main functionality of the CDP Automatic Update Service. It serves as a bridge to connect the other two components of the service. Besides, it contains the functionality of checking for the existence of a new version and of applying the new update.
The version checking mechanism implemented in the CDP Automatic Update Service is to compare the version information saved in the local registry with the version of the downloaded file. (This is a feature that could be improved in the future, namely, that CDP Update service will receive the latest version information on the server without actually downloading the file.)

After the CDP Service is properly installed in the client, the current version of it will be saved in the location HKLM/SYSTEM/ControlSet001/services/CDP under the key “version”. If the string is different from the one we get from the update server, the program will initiate the process of applying the new CDP service.

![Registry Editor](image)

Figure 14: CDP Updater Newer Version

After a newer version is downloaded from the server (see Figure 14), the first step is to create a socket client communicating with the CDP service. The application of the new version will not start until all currently running download/upload jobs have been completed. The second step is to determine the current status of CDP service (e.g.
Running, Stopped, and Starting). If the CDP service is in a pending status like starting or stopping, then the program will wait until it enters a stable state.

The step to upgrade a running service is to first stop it, delete the original jar file, copy the new class files and finally restart the service. If the service is stopped, then we can simply copy the class files. We leave the requirement to revert to an earlier version as future work.

### 3.2 FTP Server Directory Structure

Our current design is to have an update folder located at the root directory of the ftp server. All CDP Service jar files will be put inside this folder. There is a naming convention for the jar files so that the Automatic Update service can get the version information by parsing the file name. For the time being we follow the format of CDP_Service_<Version #>.jar.

In the future, we expect that a more sophisticated structure could be implemented on the server that works with the http web service.
3.3 Future Improvements

Future work will focus on two aspects. The first is to move the functionality of Transfer.java to a hosted web service on a server. The second is to design a local file directory structure for CDP Automatic Update Service. Currently many directories referenced in the batch file are hardcoded. After such a structure has been determined, we can substitute all these absolute paths with their relative paths.