Build Server

Operational Concept Document

CSE681-PROJECT #1

By Hao Zhang (SUID: 504247184)

Instructor: Jim Fawcett

Date: 13th September, 2017

Contexts

[1. Executive Summary 4](#_Toc493109113)

[2. Introduction 5](#_Toc493109114)

[1. Application Obligations 5](#_Toc493109115)

[2. Organizing Principles 6](#_Toc493109116)

[3. Key Structural Ideas 6](#_Toc493109117)

[3. Uses 7](#_Toc493109118)

[1. Myself 7](#_Toc493109119)

[2. Instructor and TAs 7](#_Toc493109120)

[3. Developers 7](#_Toc493109121)

[4. Quality Assurance Engineers 7](#_Toc493109122)

[5. Project managers 8](#_Toc493109123)

[4. Application Activities 8](#_Toc493109124)

[4.1 Read and enqueue the XML files 9](#_Toc493109125)

[4.2 Start Build Server manager and create loader directory 9](#_Toc493109126)

[4.3 Decode XML files and fetch specific tool chain to the loader 9](#_Toc493109127)

[4.4 Start build process 9](#_Toc493109128)

[4.5 Get the build results and then write the log 10](#_Toc493109129)

[4.6 Request test harness execute test or Error management 10](#_Toc493109130)

[4.7 Display the result, end build execution 10](#_Toc493109131)

[5. Partition 12](#_Toc493109132)

[5.1 Build Server Executive 12](#_Toc493109133)

[5.2 Lib Fetcher 13](#_Toc493109134)

[5.3 XML File Decoder 13](#_Toc493109135)

[5.4 Queues 13](#_Toc493109136)

[5.5 Builddomain manager 13](#_Toc493109137)

[5.5 Build Server Loader 14](#_Toc493109138)

[5.7 Logger 14](#_Toc493109139)

[5.8 IBuild 14](#_Toc493109140)

[5.9 Messaging 14](#_Toc493109141)

[5.10 Client Interface 15](#_Toc493109142)

[5.11 Test Harness Interface 15](#_Toc493109143)

[6. Critical Issues 15](#_Toc493109144)

[1. Build Server Requests handling 15](#_Toc493109145)

[2. Performance 15](#_Toc493109146)

[3.Unreadable Input 16](#_Toc493109147)

[4.User Interface 17](#_Toc493109148)

[7. Conclusion 17](#_Toc493109149)

[8. Appendixes 18](#_Toc493109150)

[9. Reference 19](#_Toc493109151)

Figures

[Figure 1High level activity diagram of Build Server 8](#_Toc493101060)

[Figure 2 Detailed activity diagram for Build Server 11](#_Toc493101061)

[Figure 3 Package Diagram of Build Server 12](#_Toc493101062)

[Figure 4 File Manager output 18](#_Toc493101063)

# Executive Summary

This document illustrates the operational concept of the Build Server system.

**Introduction:**

In today’s software industry, big system has become very popular and common. Big systems with large scale can hardly be developed by one single developer. Developers, quality assurance engineers and project managers always form up as a team to develop systems. How can we fit the modules and packages developed by different people with different technics into the baseline of bug system? Build Server is designed to help us solve this issue.

**Uses:**

Build Server is designed to meet requirements from 5 types of users. Every user can choose the user mode in the command line when start the Build Server.

**Application Activities:**

The first step shall be accepted build requests in the form of XML files. The XML files will be changed into string format and enqueue the queue. Then Build Server creates child Builddomain for the XML file, and the child Builddomain requests the queue to dequeue new XML string. XML string shall be decoded into application readable format in child Builddomain. Lib Fetcher package shall search the directory tree and find source and tool chain files in the repository for further processing.

Then source files tool chains will be loaded in the child builddomain. The build() method in IBuild can be called by Test Executive, taking no arguments and returning the build pass status, e.g., a Boolean true or false value. The getLog() function in IBuild can be used to retrieve logs from the child Builddomain. Child Builddomain will merge the results and logs according to users’ mode, then it will query the Queue. If the XML queue is not empty, child Builddomain will be deleted and new child Builddomain will be created for next XML string. If the XML queue is empty, Build Server will delete the child Builddomain and end the build process.

**Partition:**

The system will be divide into 11 modules. Main modules are as below:

* Build Server Executive- holds the main method, and is responsible for the main control flow of the system.
* Builddomain Manager- is used to create child Builddomain and call the Build Server Executive to do the test.
* IBuild- contains build() method which can be call by Build Server Executive, taking no arguments and returning the build pass status, e.g., a Boolean true or false value

**Critical Issues**

In implementing Build Server, several critical issues shall be carefully considered. Their possible solutions have been given, and some of the issues are as blow:

* Requests handling- in which Build Server get build requests and change into application readable format.
* Performance- how to deal with the extreme situation, like hundreds of build requests flood in or single build request contains hundreds of source files and tool chains.

The Build Server thus ensures that it can handle the build requests, do the build successfully and demonstrate results clearly.

# Introduction

When we develop large projects, we should always struggle with the issue that, how we can successfully insert the packages, classes and modules developed by different developers into the baseline of the final project. Are there any easier and safer ways to solve that issue? Continuous Integration is great idea for this large-scale develop. And Build Server is one of the most important parts of a continuous integration system. Build Server is designed to make sure the source files from the potential clients can be correctly built in fast pace.

## Application Obligations

The primary requirement of Build Server is building the code. So that the different code written by different program tool chains can fit into the baseline without any error. The main obligations of the application should be:

* To accept one or more Build Requests, each in the form of an XML file.
* To start the Build Server manager and create loader directory for current build process, delete it when build ends.
* To decode the XML files and fetch specific tool chain(C#, Java, C++, etc. ) into the loader
* To execute the build() method and returns build results as DLL files, store them in the mock repository.
* To write the build status in logs and store them at the child Builddomain
* To request Test Harness Mock execute the DLL files, return test results and write test logs.
* To retrieve build results and build logs for every build execution along with the code developer’s identity and current time for display.

## Organizing Principles

The organizing principles are to perform build requests in queue order, decode the XML files for address of build source files, fetch the specific tool chain, execute the build request in build loader, request test harness to run the test, store the build results, logs etc. and retrieve the logs and build results to the client..

## Key Structural Ideas

* The building request defer to the queue order, as first come, first build. It protects the whole system from request flood.
* To create the Build Server Loader directory for each specific build.
* Builddomain manager decodes the XML files and get the build source file directory and the specific tool chain to be used.
* Build loader contains the tool chain files fetched by Lib fetcher. Builddomain manager use the corresponding build method(like MSBuild for C#) chose from the XML information to build the source files. Ibuild returns the build status and build results, and stores them in logs and repository.
* The Builddomain manager realizes build process within different tool chains, which can meats various kind of developers.

# Uses

## Myself

I am one of the primary users, when I finish it, I will build source files on it to make sure it can successfully build. Furthermore, different classes have different language requirements. I can use the same Build Server while actually I use different languages and frames.

Another use for me is that in the project 4, remote Build Server is going to be designed. In that case, I can code wherever I want only if I have a laptop with connection to the Internet.

## Instructor and TAs

The Build Server will help the instructor and teaching assistants when they are grading the codes from students. They will read the documents of it, build source files on it and examine whether all the requirements have been successfully accomplished. Meanwhile, as one instructor may use different languages in different classes, the Build Server will help instructor to deal with different build request within different language at the same time.

## Developers

Developers especially those who develop large systems, will use Build Server to do continuous integration. Build Server will help them build source files within styles of packages, modules etc. created by themselves. They can get helpful debug information from the build logs and results which can help developers make sure that the individual’s works can fit into the baseline of large project.

What I think the most important thing is that, developers with different coding language customs can work together now as the Build Server can build codes in different languages.

Developers’ interface will concentrate on detailed build logs, build files and possible error information.

## Quality Assurance Engineers

The quality assurance engineers will use Build Server for integration build to guarantee that every part of the project builds well in the baseline. They have to make sure the program integration builds successful.

The quality assurance engineers concentrate on the build results, the success rate and the error control. Their interface will demonstrate the results of build more than the logs.

## Project managers

Project managers will use Build Server to capture general information of their projects. They can read the build logs and results and understand the current status of the projects conveniently. Meanwhile, they can check each developers’ work condition and efficiency, to make sure every developer are productive and every part of the project is within due.

# Application Activities

Activity diagrams are drawn to show the detailed steps of the application execution thus apparently illustrate the function of the application.

The Build Server processing shall be divided into 7 activities. The high-level activity diagram is expressed in below diagram. The detailed activity diagram will be showed in another diagram.



*Figure 1High level activity diagram of Build Server*

## 4.1 Read and enqueue the XML files

First activity of the system shall be getting input from users. The whole build process starts at Build Server Executive. It decodes the XML files into string format for saving space. Then Build Executive calls the Queue package to enqueue the XML files. The XML files shall not dequeue until Queue receives request from Builddomain manager.

## 4.2 Start Build Server manager and create loader directory

Builddomain manager is the direct manager of each build process. As a matter of fact, when each build request comes, a child Builddomain is built and create a new directory, which is going to store source files, related tool chains and logs. The execution of source file build will run by the child Builddomain, isolated from the processing of Build Server Executive. In that case, the specific build process will not influence the whole Build Server. Before XML file dequeues, the Builddomain manager shall create the new child Builddomain, and send request to Queue for new XML file. When Queue receives the request, it will dequeue the XML file which is in string format, then send it to the child Builddomain.

## 4.3 Decode XML files and fetch specific tool chain to the loader

XML string shall be decoded by the XML File Decoder firstly. XML File Decoder will decode the XML string to application readable format. When successfully decoded, the Builddomain manager will ask Lib Fetcher to search the information in XML file and display the information including client name, request time, tool chain, and source file directory. After that, the Lib Fetcher is going to fetch the tool chain and source file from repository mock for further processing.

## 4.4 Start build process

The tool chain and source file will be loaded at the Build Server Loader after they are fetched by the Lib Fetcher. The Lib Fetcher shall make sure all the tool chain libraries are prepared before building. The build() method in Ibuild can be directly called by Builddomain manager, execute build order and return build results with the build pass status, e.g., a Boolean true or false value. This status value will then be fetched by the getLog() function in Ibuild and be stored at the logger later.

## 4.5 Get the build results and then write the log

After the Ibuild finishes the building process, a certain result will come out. If the build succeeds, then the child Builddomain will return the success build result files with a true Boolean pass status. The success build result will be sent to repository mock by Isend(). If the build fails, the child Build will return the false Boolean pass status and leaves the Build Server Loader decided by the Builddomain manager. The Builddomain manager will use IHello() and IReply() as interface to contact with Build Server Executive about the build status. Furthermore, the build status will be written in the logs by the logger.

## 4.6 Request test harness execute test or Error management

Once the build process succeeds, the Build Server Executive will send a success message to the Client mock and a test request to the Test Harness mock by the Test(). If the build fails, the Build Server Executive will also send a fail message to the Client mock and trying to rebuild the source files.

## 4.7 Display the result, end build execution

Build results shall be sent to Message manager package, and the Message manager will send the build results then to the Client mock and then display them. When results are successfully displayed, child Builddomain will query whether the queue is empty. If the queue is not empty, child Builddomain shall use IHello() interface to send a message to request Build Server Executive to delete the child Builddomain with its Build Server Loader and create a new one for the next XML file in the queue. If the queue is empty, child Builddomain shall use IHello() interface to send a message to request Build Server Executive to end the process.



*Figure 2 Detailed activity diagram for Build Server*

# Partition

Build Server is divided into several packages, each of them conduct its own responsibility. The performance of Build Server depends on the interactions among these packages.

Figure 3 is Package Diagram of Build Server.



*Figure 3 Package Diagram of Build Server*

## 5.1 Build Server Executive

Build Server Executive is the primary package of the system, which controls various interactions with other packages. The Build Server Executive holds the main method, and is responsible for the main control flow of the system, beginning with reading input as XML files and ending in delete the child Builddomain. The packages Build Server Executive calls while conducting builds are as follows:

* XML File decoder package to decode the XML files and build queue for loading.
* Builddomain manager to create the child Builddomain manager for the test execution, and delete it when Build finishes.
* Logger package to get test logs and store logs by child Builddomain.
* IBuild package to call the build() function, executive the build and return the results.
* Message package to pass the results and logs to other packages.
* Client Interface package to connect with Mock Client.
* Test Harness Interface package to connect with Mock Test Harness.

## 5.2 Lib Fetcher

Lib Fetcher can be used to search the directory tree in XML files and display the names of all source files and the tool chains, libraries they need. Lib Fetcher will be called by the Build Server Executive to find the source files in the repository, each with their own build libraries. source files will be further processed in the child Builddomain manager.

## 5.3 XML File Decoder

When new XML file dequeues and be sent to the child Builddomain manager, this package will be called by the Build Server Executive to decode the input XML files. The XML files will be decoded into the format which can be read by Build Server Executive. And then the results will be passed to the Executive for later search and process.

## Queues

Queues package is used to build a queue for the XML files and manage the dequeue process. Queue builds the queue when the first build request XML file come in, and then Build Server Executive enqueues the XML files. After that, new build request XML files will enqueue as soon as it comes. When a the a request is going to reply and the build process is going to be done, the Build Server Executive will send the dequeue require the queues will dequeue the XML file.

## 5.5 Builddomain manager

Builddomain manager is used to create child Builddomain for a build request. Child Builddomain contains the source files to be built and the tool chains need in the build process. It requires the Queue to dequeue new XML file to find the necessary information, and isolates the build processing from the Build Server Executive processing.

After one build process finished, Builddomain Manager will query the Queue whether queue is empty. If the queue is not empty, Appdomain Manager will call Test Executive to delete the child Builddomain and create a new one for next test request. If the queue is empty, Builddomain Manager will call Build Server Executive to delete the child Builddomain and end the test execution.

## Build Server Loader

Build Server Loader is the ‘Repository’ of each child Builddomain. When source files are retrieved from the repository, Build Server Executive calls the Build Server Loader to load the source files for further processing. When the build request XML files are decoded, the Build Server Loader will also be called by the Build Server Executive to load the tool chains.

## 5.7 Logger

Logs are very important test information in this system. Logger package declares methods to get logs of the build execution and store the logs by the child Builddomain. Build logs will then be passed to Mock Repository for Client and Test Harness by Messaging package.

## 5.8 IBuild

IBuild is the package which contains the main functions to execute build on build code. The build() method in IBuild can be called by Build Server Executive, taking no arguments and returning the build pass status, e.g., a Boolean true or false value, together with the build result files as DLL. In this document, the build() method in IBuild is MsBuild, in later project 3, the concept maybe enlarge to more kind of programming languages and the build() method will also support them. The getLog() function in IBuild can be used to retrieve logs from the child Builddomain.

## 5.9 Messaging

Messaging package is used to pass the commands, results and logs to other packages. For example, when the child Builddomain queries the XML files queue, the queue package uses the messaging package to send the result of the query. When the Logger get the logs of the build, the Messaging package pass the logs to the Mock Repository waiting for the client to fetch.

## Client Interface

The Client Interface package is used to pass messages between Mock Client and Build Server, including source files, tool chains, build logs, build results.

## 5.11 Test Harness Interface

The Test Harness Interface package is used to pass messages between Mock Test Harness and Build Server, including source files, tool chains, build logs, build results.

# Critical Issues

## Build Server Requests handling

What kinds of Build request can be accepted by the Build Server Executive? How to deal with the Build request? Build Server Executive should take the input Build request, and provide the order of Build Server execution, decode the request into application readable format.

This issue concerns the entry of the system, and is of vital importance.

**Solution:**

* The Build Server Executive only accept Build request in the form of a XML file, to make the application can understand the build request.
* XML Decoder package will decode the XML files into application readable format for further processing.
* Lib Fetcher will search the information in decoded XML files and find the build source files in the repository, each with its own tool chain(libraries). Build files and their tool chains will be further processed in the child Builddomain.

## Performance

Performance is a great important issue in large scale systems’ construction. In this system performance issue concerns about how many Build Requests it can deal with, how many source files and their tool chains a Build Request can contain, how long it takes to conduct a build execution on a build request with several source files.

In the typical situation, estimated that only a few XMLs in the queue and each of them contains just a few source files and tool chains, the performance of the systems will not a problem. But when the situation comes bad, for example at the deadline of a task or the deadline of a new version of a software is going to update, hundreds of build request will be sent to the queue. Furthermore, each build request may contain different tool chains, or large project may contain numerous source files. At that time, performance of the Build Server system is an issue of vital importance.

**Solution:**

* The package Queues is designed to conduct enqueue and dequeue operations with input XML files. In case the amount of XML files is really large, Queue will allocate large space to create the queue. And the XML file will not be dequeued until Queue receives requirement from child Builddomain.
* Build Server Executive will allocate enough space for Child Build domain to deal with a build Request with hundreds of source files and tool chains. Build Server Loader package is designed to load the source files and tool chains.
* When XML files are in queue, they will be stored in string format. Because the string format need less space than the XML files.
* Test Executive will delete the child Builddomain when a build request finishes the test execution. This operation will free the space for new Builddomain., and prevent Build Server from being crashed by the accumulating load.
* In some extreme situations as there are too many tool chains to be loaded, then the Build Server manage may initiate an extra directory specially load these tool chains to ease the pressure of child Builddomain manager.

## 3.Unreadable Input

If unreadable input is accepted, errors will occur when input is read by the application. Build Server must handle the unreadable input.

**Solution:**

Build Server should declare that the input build requests should be in the form of XML files. The get input function will not accept build requests in other format except XML files.

## 4.User Interface

User Interface should be clear and convenient. And different user should choose different mode to execute the build.

**Solution:**

User can choose the user role for three modes: student/instructor, developer, quality assurance/project manager. Command line shall be given as below

*/u for choosing user mode*

*/p for input XML files*

*/r for displaying the results*

*/q for ending the test execution*

# Conclusion

In conclusion, Build Server can receive build requests, execute the build process and demonstrate results clearly.

The OCD illustrates the system aimed users and what they can do with Build Server. Different modes that meet different requirements of users are discussed. Build Server is divided into 11 packages to perform the function.

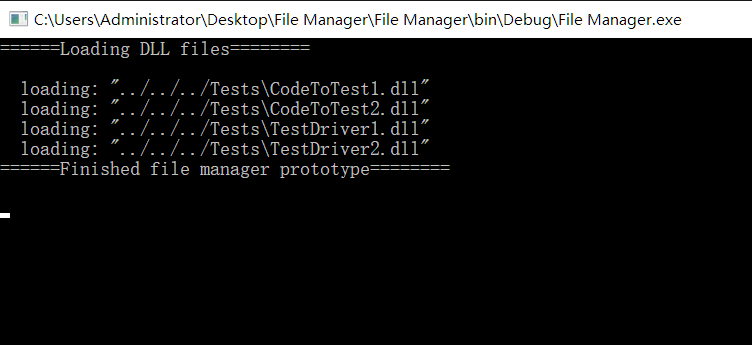
The OCD also present detailed activities in the Build Server processing. OCD illustrates how the build execute in the system and help us understand the system.

# Appendixes

**File manager prototype:**

File manager prototype is File Manager package that searches a directory tree in a specified path, and displays the names of all files encountered. The source code is attached to the document as files.

The output of file manager prototype is as below:



*Figure 4 File Manager output*

This prototype is referred from:

<http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/BestProject1s/TianYou.zip>

# Reference

<https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm#top>

<https://docs.microsoft.com/en-us/visualstudio/msbuild/walkthrough-creating-an-msbuild-project-file-from-scratch>

<http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/BestProject1s/TianYou.zip>

<https://en.wikipedia.org/wiki/Continuous_integration>