Remote Build Server

Operational Concept Document

Beier Chen

CSE681 Project# 4

2017/12/8

Table of Contents

1 Executive Summary

This is an Operational Concept Document of a Remote Build Server, the course project of CSE681 Software Modeling and Analysis in fall 2017. It is implemented in C# using the facilities of the .Net framework class libraries and Visual Studio 2017.

The implementation is accomplished in three stages. The first, Project #2, implements a local Build Server that communicates with a mock Repository, mock Client, and mock Test Harness, all residing in the same process. Its purpose is to allow the developer to decide how to implement the core Builder functionality, without the distractions of a communication channel and process pool.

The second, Project #3, develops prototypes for a message-passing communication channel, a process pool to conduct multiple builds in parallel, which uses the channel to communicate between child and parent Builders, and a WPF client that supports creation of build request messages.

Finally, the third stage, Project #4, completes the build server, which communicates with mock Repository, mock Client, and mock Test Harness, to thoroughly demonstrate Build Server Operation. The Build Server will function as one of the principle components of a Software Development Environment Federation[1].

2 Introduction

During the development of large and complex software system, the code will be divided into relatively small parts, and many developers will contribute to interdependent packages.

When developers need to add new features, or Quality Assurance personal try to fix latent error, new parts of code will be inserted to the baseline. To make sure the baseline continues to function after changes integrating, all code should be tested thoroughly. Only when all the code passes the test, it will become part of the current baseline.

Because there are so many packages (perhaps contains thousands of packages and several million lines of code), it becomes necessary to automate the process of testing new contributions. The practice of merging all developer working copies to a shared mainline is called continuous integration.

In a Software Development Collaboration System, there are several components necessary to efficiently support continuous integration, as shown in the Figure 1.

[1] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/midterm/MTF17/MT1F17-InstrSol.pdf

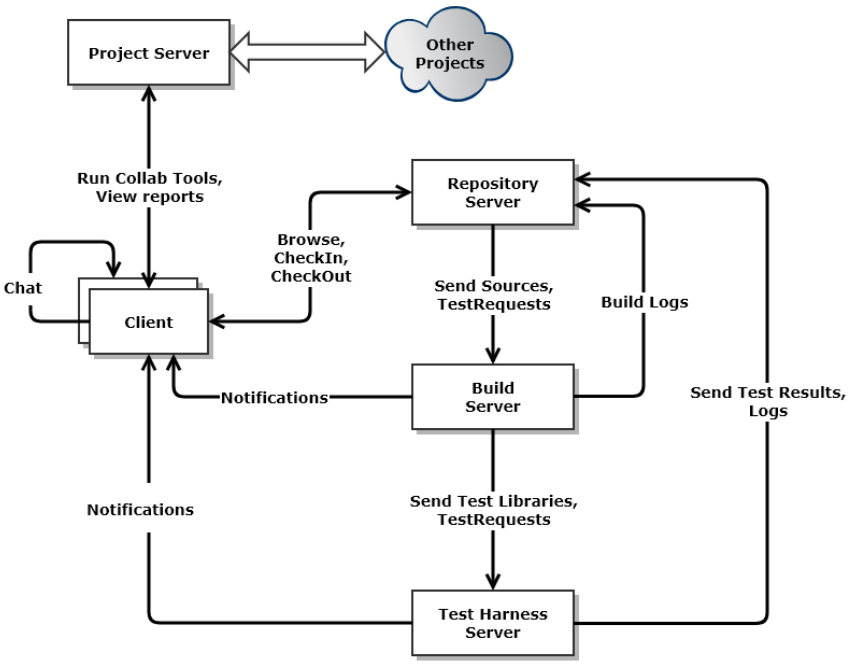

Figure 1 Software Collaboration Federation[2]

The projects for this course concentrate on one of these federated servers, the Build Server - an automated tool that builds test libraries. The Build Server is the heart monitor of Continuous Integration System. Its goal is to provide a stable and reliable environment for building distributed development projects, preventing integration conflicts happen.

This document first makes a user analysis and discusses the impart of different types of users on the design. Then discusses the role of four major components of a collaboration system, and their relationship. Next, describes the design and implementation of the Project# 4, including activity and partition. Finally, critical issues and implementation deficiencies are discussed.

3 Uses

This part of the OCD makes an analysis of four kinds of users and their uses of the project.

3.1 Developers

Developers are the primary user of the Remote Build Server. There are two main functions developers would need from the project. The first is requesting tests to make sure their own work are compatible with the baseline. The second one is getting logs of completed tests.

Impact on design

[2] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2017.htm.

The system should provide an easy way for developer to make test requests in XML files. Sometimes, developers might need tests results quickly to continue next step, so the build server should response fast enough. In other circumstances, lots of tests usually pile up at the end of the day, the developers would need the system to schedule running tests overnight and view the results the next day.

As the most critical information to developers is the results of the tests, the system should be able to allow developers pulling up all the completed test logs, and be able to view them easily. The logs include the tests developers made for their own work and request and results from other developer.

Thus, an accessible Graphical User Interface (GUI) would be useful for the developers to upload test requests, schedule tests and view the reports.

3.2 Quality assurance team

Quality Assurance team is responsible for the quality and sustainability of the software product before delivering to customers, preventing mistakes or defects. QA personnel extract large parts of a baseline to create builds for regression testing, and view the result of completed tests afterwards.

So, what QA needs from the Remote Build Server is very similar to the developers, but QA might run more demanding tests than developers do.

Impact on design
Almost same as the developer, but the XML test requests from QA might be much larger than developers'. The system should respond more quickly to deal with these requests.

QA may want to check test requests made by the developers to make regression testing, so the previous XML requests and results should remain available.

3.3 Project Managers

The project managers are responsible for delivering a large complex system that meets its customer requirements, satisfies code quality standards within the project schedule.

They have a less direct relationship with the Remote Build Server than developers and QA team. PM focus on the overall performance of the project, keep track of the progress of each team member and adjust task assignment accordingly. For this reason, the information provided to PM should be filtered and display in a more clear and general way.

Impact on design

To suit the special need of PM, the system should provide a different mode for them, which provides simpler GUI and has a programmatic extraction process to enable PM to get the most relevant information easily, including the numbers of tests submitted by individual developers, the percentage of tests that have passed and the summary of commit activities on a scheduled basis.

3.4 Customer

During the development process of software product, the development team need to supply progress information of the project to the customer regularly. This is likely to be a subset of the information that the Remote Build Server provided to the PM.

Impact on design
There should be no additional impact on design to satisfy the need for progress information for customer reviews. That is already provided for managerial use. However, it might be useful to provide customers a special interface to access progress information efficiently.

4 Federation Components

This part of OCD describes the role and relationship of the four major components of the Collaboration System, elaborating from the requirement of Project# 4[3].

The project will not integrate the Build Server to a full up Federation, instead, the project builds a mock Client, a mock Repository Server and a mock Test Harness, running on their own process. Their functions are relatively simple, just enough to demonstrate operations of the Remote Build Server. All the components of the system use Message-Passing Communication Service to communicate with each other.

The figure below shows the Block structure of the final project.

[3] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project4-F2017.htm

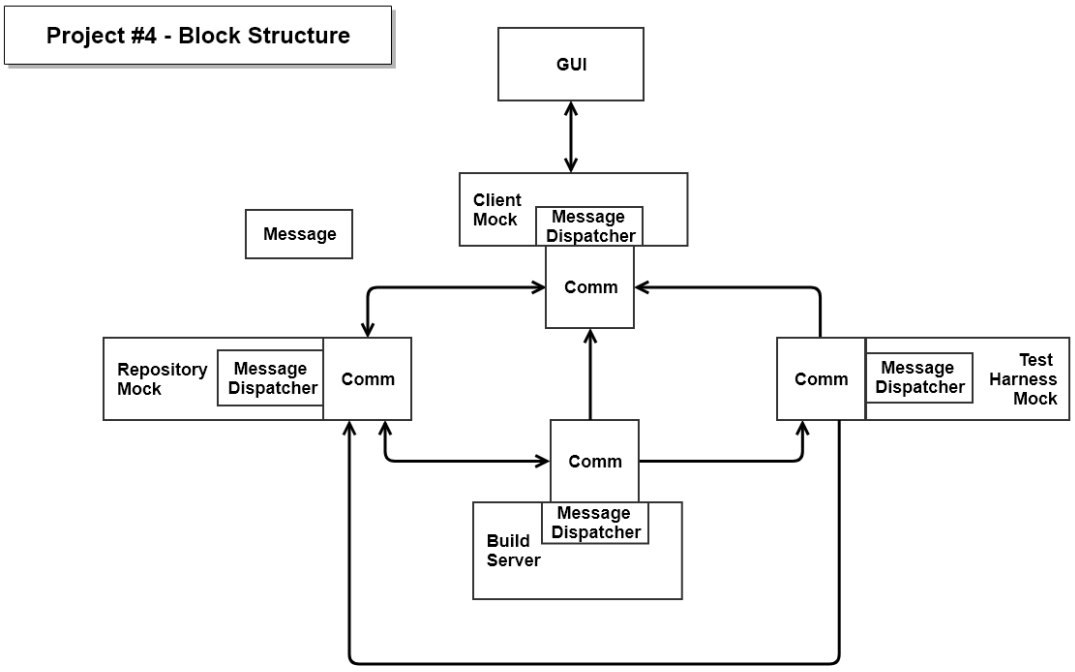

Figure 3 Block Structure of Project #4 [4]

4.1 Client

Client is the actor of system. It has a Graphical User Interface(GUI), which is a separate process, implemented with WPF. It uses Message-Passing Communication to communicate with other components of the system.

Client gets file lists from the Repository first. Then selects files for packaging into a test library, adding to a build request structure and send build request structures to the repository for storage and transmission to the Build Server. Client can repeat the process to add other test libraries to the build request structure.

Finally, Client can request the repository to send a specified build request in its storage to the Build Server for build processing. The diagram bellow shows the activities of Client.

[4] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2017.htm.

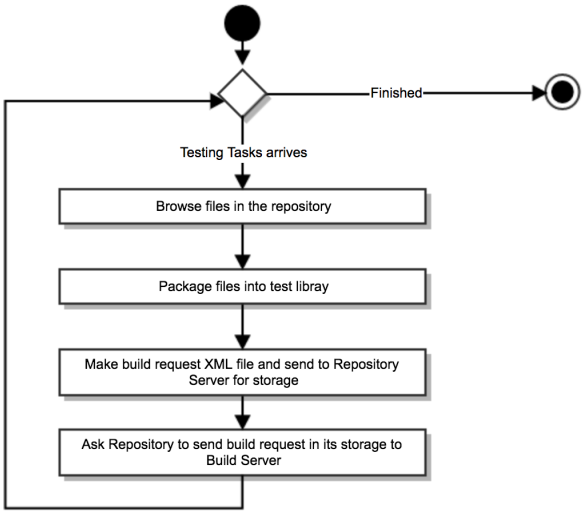

Figure 4 Activity Diagram of Client

Project# 4 implements Client function using ClientGUI package.

4.2 Repository Server

Repository Server waited for command from Client and Child Builders and responses accordingly. Repository Server also stores build request from Client, build log from Child Builder and test log from Test Hardness.

If received file request from Client Repository Server returns file list in the reply message to Client.

If received build request XML file from Client, Repository Server store the build request in its storage and return updated build request file list to Client.

If being asked to send build request to Build Server by Client, Repository Server sends the specified build request to Build Server.

If received file request from Child Builder, Repository Server sends specified files to Child Builder.

If received build log from Child Builder and test log from Test Hardness, Repository Server stores the log.

Project# 4 implements Repository Server function using Repo package.

4.3 Build Server

To make high throughput for building code, Build Server has a “Process Pool” component, which spawns a specified number of processes on command at startup. Pool Processes (Child Builders) use message-passing communication to access messages from the mother Builder process.

Each Pool Process (Child Builder) attempt to build each library, cited in a retrieved build request, logging warnings and errors in build log and send it to Repository Server.

If the build succeeds, Pool Process sends a test request and libraries to the Test Harness for execution. Child Builder then sends a ready message to Build Server and begin to another build process.

Project# 4 implements Build Server function using BuildServer and ChildBuilder packages.

4.4 Test Hardness

Test Harness attempt to load each test library it receives and execute it. After executing the test, Test Harness submit the results of testing to the Repository.

Project# 4 implements Test Harness function using TestHarness and DllLoader packages.

5 Task and Activities

The final project has four components of the collaboration system running at the same time. The diagram below shows the activities of the Federation executing a testing, which starts from Client selecting files and ends Test Hardness executing received test library.

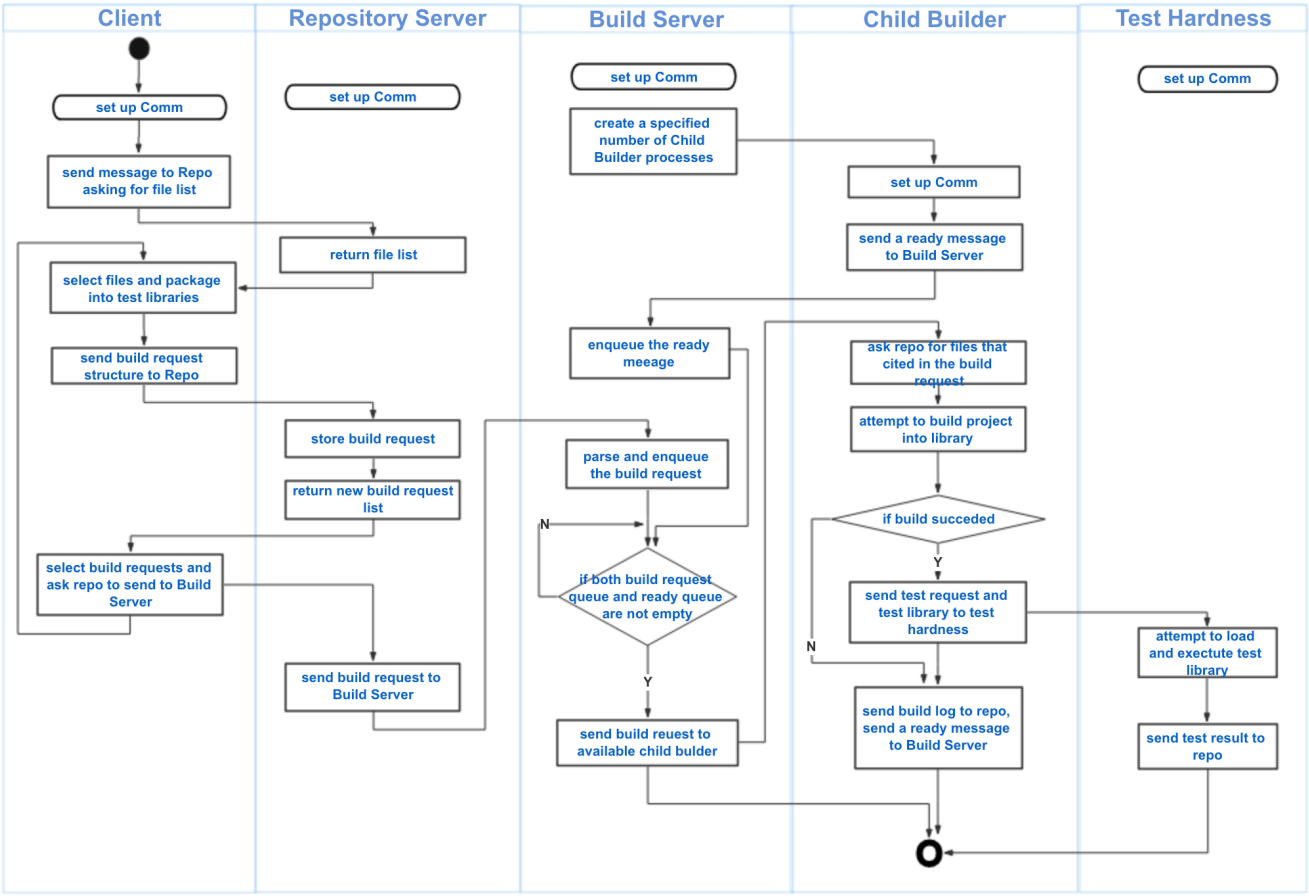

Figure 5 Activity Diagram of Federation

Project 4 starting with four processes: Client Console, Repository Server Console, Build Server Console and Test Hardness Console.

After starting, Client loads the WPF GUI Windows, and Build Server creates a specified number of Child Builders processes (in my submitting program, the number is two).

All the processes set up communication first, and all the communication in the Federation is through WCF Message-Passing Communication.

Client begins to browse files in the repository through the GUI Window. Then Client selects test driver and tested files to package into a test library, making a xml build request structure that cited the file name and sends to Repository Server to store. Repository Server stores the XML build request structure into a XML file.

Next, Client asks Repository Server to send a selected build request in the repository to Build Server.

Repository Server sends build request to Build Server.

Received request, Build Server parse the build request. As a build request can contain several test libraries, Build Server first separates them and makes a new build request with only one test. Build Server sends the build request to available Child Builder.

Child Builder then ask Repository Server for the files cited in the build request. After receiving files, Child Builder builds the project, logging build result and error in build log.

After build finished, Child Builder sends a build log to Repository Server and sends a ready message to Build Server. If build succeeded, Child Builder send a test request and the test library to Test Hardness.

Test Hardness received the test request from Child Builder, and begin to load and execute test library, logging test result in test log.

After test finished, Test Hardness sends the test log to Repository.

6 Project Partitions

This part of OCD describes the operations of each package in the final project.

Project# 4 has 12 packages: ClientGUI, Repo, BuildServer, ChildBuilder, TestHardness, Dll Loader, Environment, XMLHander, MPCommService, IMPCommService, FileManager and PrintTool. The diagram below shows all packages and relationship between them.

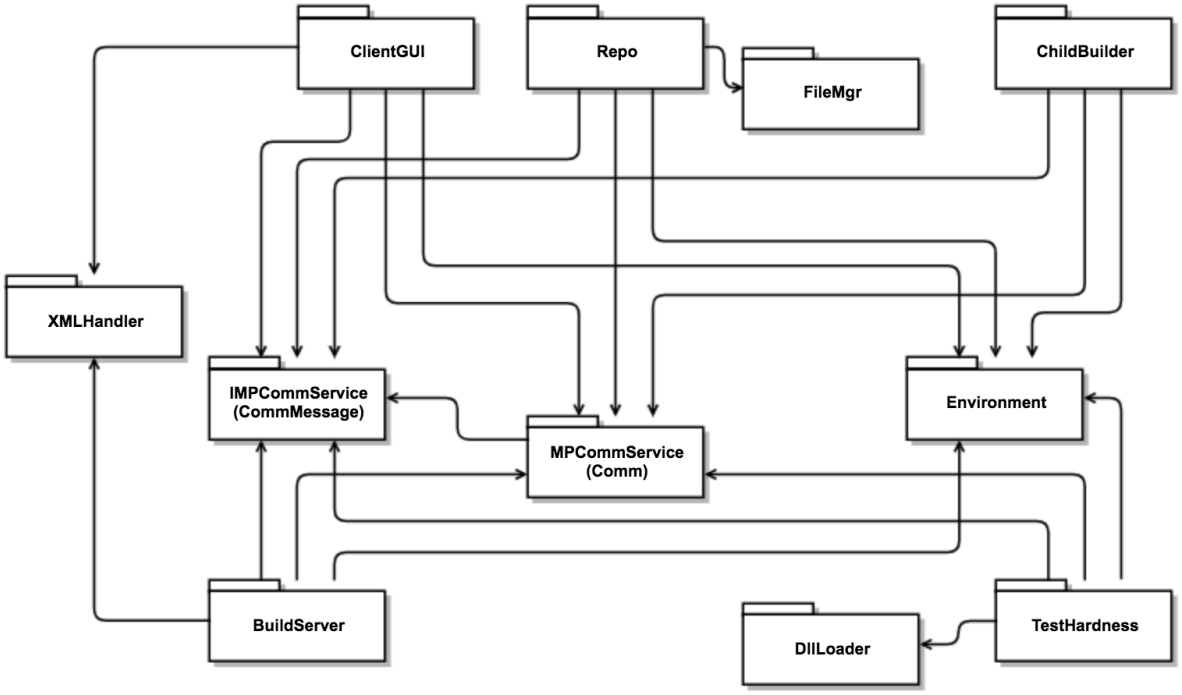

Figure 6 Package Diagram of Project# 4

5.1 Message-Passing Communication

The packages MPCommService, IMPCommService implement the Message-Passing Communication built with Windows Communication Foundation(WCF).

The communication uses a Peer-To-Peer message-passing structure, as the diagram below.

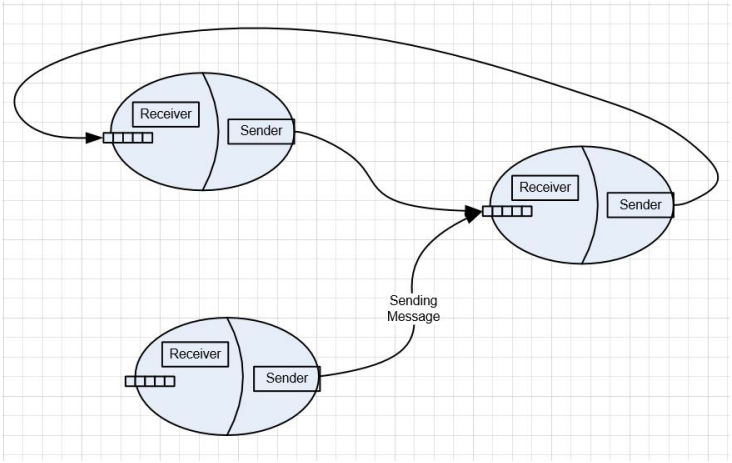

Figure 7 Peer-to-Peer Communication Structure[5]

[5] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/webpages/BlogMessagePassingComm.htm

The communication uses IMPCommService interface, which is the communication contract for services, data, and messages. Each Peer is a separate process, and contains a sender and receiver. The sender communicates with one receiver at a time. The receivers each handle concurrent senders by accepting messages in a thread-safe receive queue, which is implemented with BlockingQueue package in Project# 4.

The two diagrams below show the message flow of Project#4.

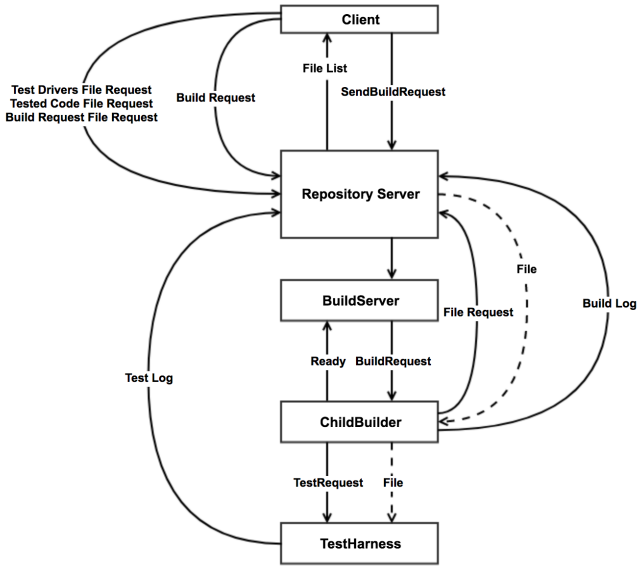

Figure 8 Message Flow Diagram of Project# 4

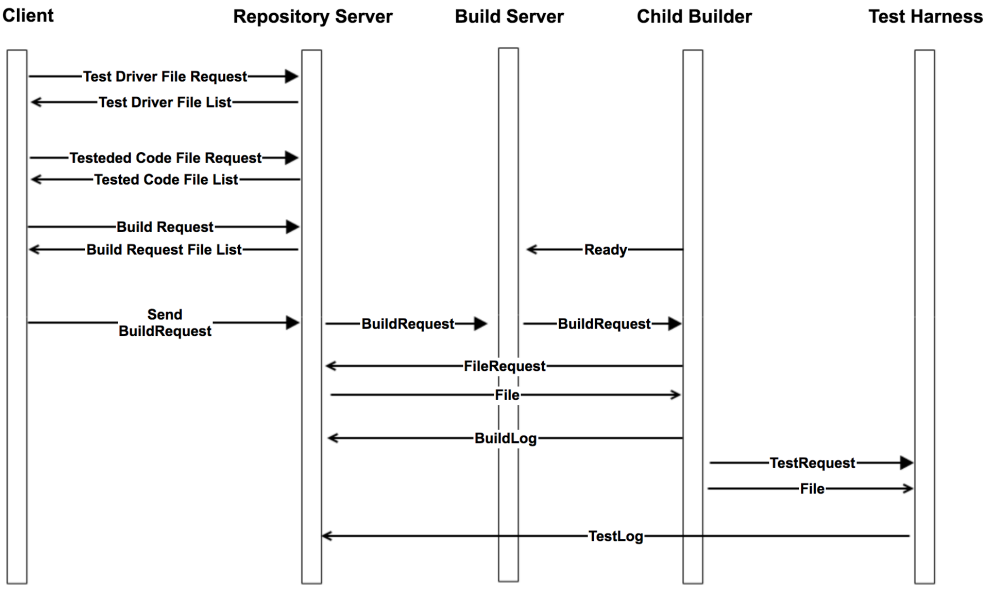

Figure 9 Alternative Message Flow Diagram of Project# 4

Every peer that establishes Message-Passing Communication has a Message Dispatcher to define different action towards different type of received message. In a WPF application, child threads cannot directly manipulate their parent's window. Instead, they are expected to Dispatch a delegate for the operation they want to execute to their parent's thread.

In Project# 4, Message Dispatcher is implemented with a Dictionary<CommMessage.Command, Action< CommMessage >>, CommMessage.Command defines the type of processing needed for a message, and the Action defines the processing used to handle that message[6].

The diagram below shows the Message Dispatcher activity for Repo in Project# 4.

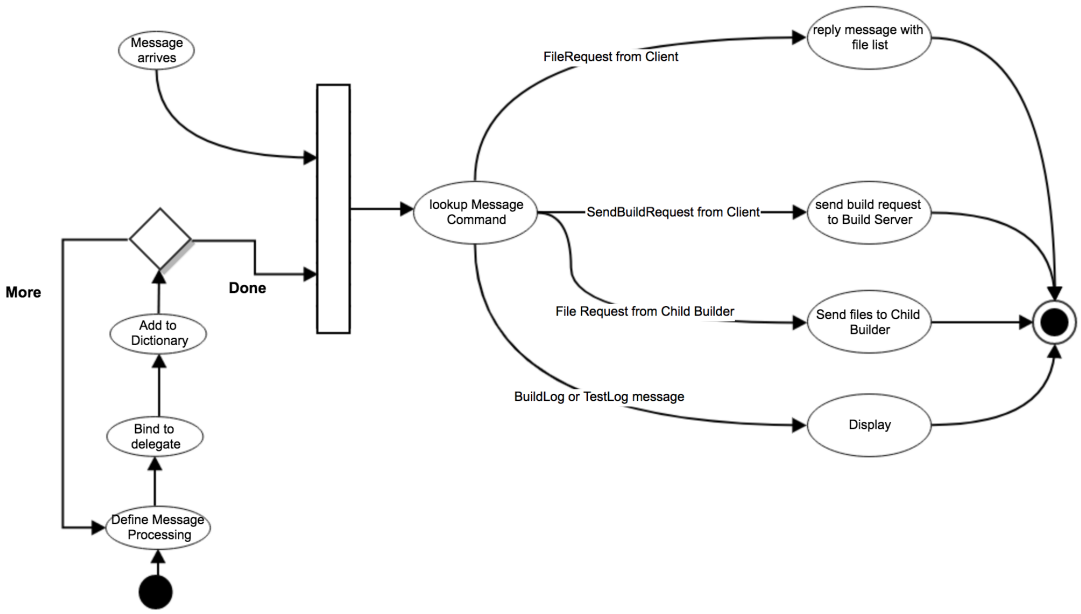

Figure 10 Message Dispatcher of Repo

5.2 ClientGUI

The ClientGUI package implement the Graphical User Interface for mock Client, as shown below.

[6] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/midterm/MTF17/MT1F17-InstrSol.pdf

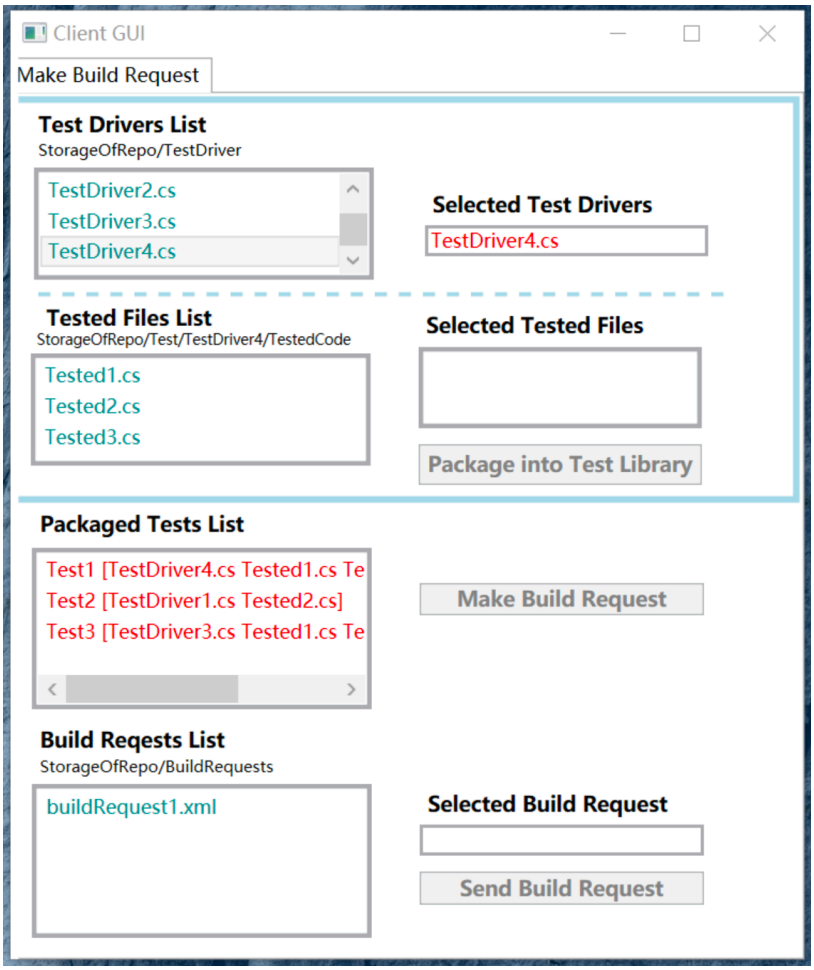

Figure 11 GUI of Project# 4

The Interaction logic for GUI:

0 Initialize:
- GUI sends a TestDriverList request message to Repo
- Repo returns the Test Driver List
- GUI updates “Test Driver List”

1 Select a file from “Test Driver List”
- GUI updates “Selected Test Driver” TexBox
- GUI sends a TestedList request message to Repo
- Repo returns the Tested File List
- GUI updates Tested File List and directory label

2 Select files from “Tested File List”
- GUI updates “Selected Tested List”
- GUI enables “Package into Test Library” Button

3 Push the “Package into Test Library” Button
- GUI updates “Packaged Test List”
- GUI stores selected test driver and tested files to a build request xml structure
- GUI enables “Make Build Request” Button

4 Push the “Make Build Request” Button
- GUI makes a build request XML string, save it to xml file and sends to Repo
- Repo returns the updated Build Request File List
- GUI updates “Build Request List”

5 Select file from “Build Request List”
- GUI updates “Selected Build Request” TexBox
- GUI enables “Send Build Request” Button

6 Push the “Send Build Request” Button
- GUI sends a SendBuildRequest message to Repo
- Repo sends a BuildRequest message to Build Server

The build request is a XML string, with elements of author, time, test name, test driver, tested code. A sample build request is as below:

```
<testRequest>
<author>Beier Chen</author>
<dateTime>2017/12/8 20:56:33</dateTime>
<test>
  <testProject>Test1</testProject>
  <testDriver>TestDriver4.cs</testDriver>
  <tested>Tested1.cs</tested>
  <tested>Tested2.cs</tested>
  <tested>Tested3.cs</tested>
</test>
<test>
  <testProject>Test2</testProject>
  <testDriver>TestDriver1.cs</testDriver>
  <tested>Tested2.cs</tested>
</test>
</testRequest>
```

Figure 12 Sample build request XML string

It contains Test1 with TestDriver4 and three tested code files, and Test2 with TestDriver1 and one tested code file.

5.3 Repo

The Repo package implement the Repository Server. The class diagram of Repo is as below:

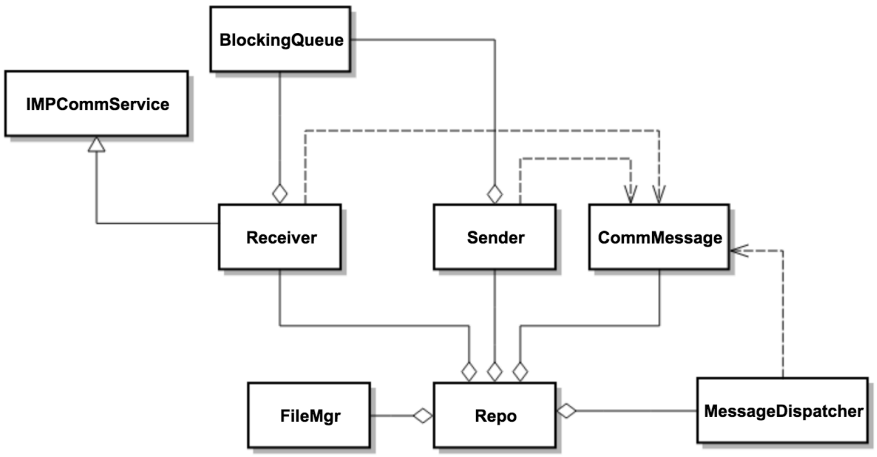

Figure 13 Class Diagram of Repo package in Project# 4

After starting the receive thread, Repo keeping checking message. If message arrives, Repo responds accordingly.

5.4 BuildServer

BuildServer package implement the Mother Builder (Main Builder) of Remote Build Server.

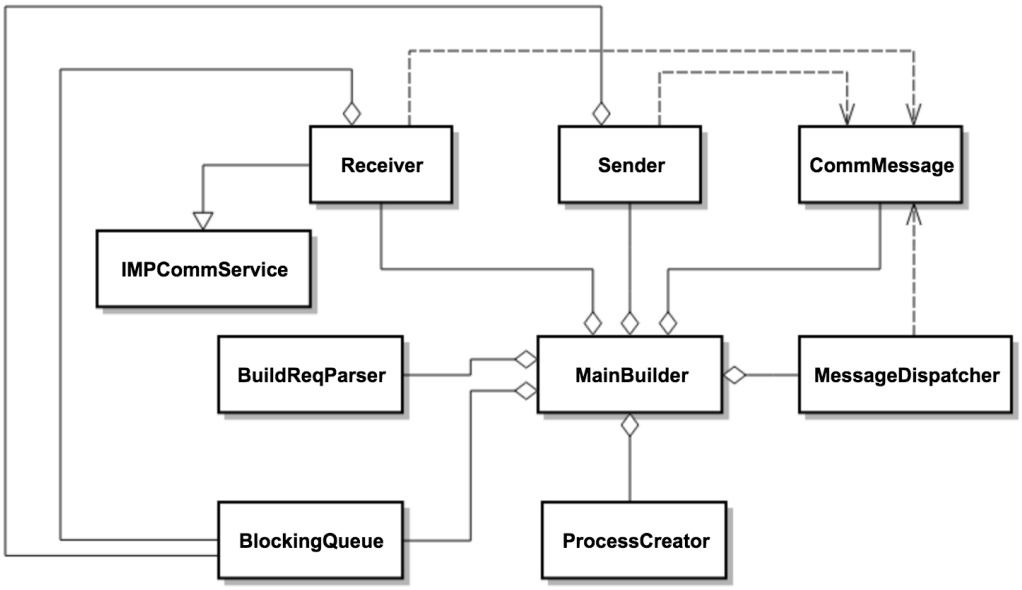

Figure 13 Class Diagram of MainBuilder package in Project# 4

It doesn't build projects itself, but distributes build request to Child Builders. At start up, the Main Builder spawns a limited number of Child Builders processes using a Process Pool. The Main Builder has a private BuildRequests queue and a private ReadyMsg queue.

Every time Main Builder received a Build Requests Message from Repository Server, it parses the XML request, and separate the different test of a build request. Then it queues the build request with only one test to the Build Request Queue.

Every time Main Builder received a Ready Message from a Child Builder, it queues the port number of the ready Child Builder to the Ready Message Queue.

If both Build Request Queue and Ready Message Queue are not empty, Main Builder dequeue a message from Build Request Queue and sends it to a ready Child Builder.

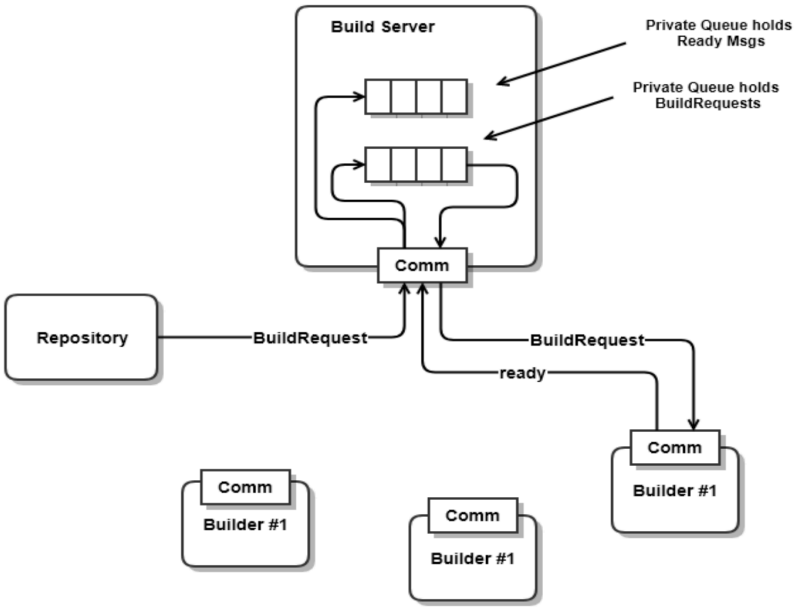

Figure 15 Build Server with Process Pool[7]

5.5 ChildBuilder

ChildBuilder package implement the Child Builder the perform the projects building function for the Remote Build Server.

[7] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project3HelpF2017/ProcessPool.pdf

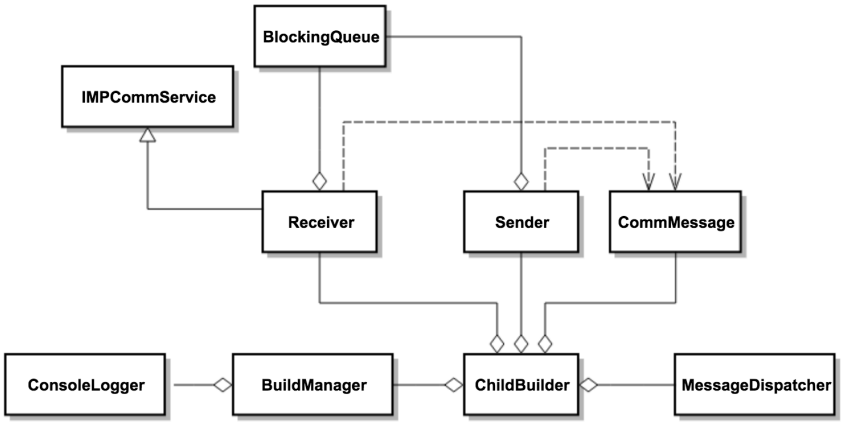

Figure 16 Class Diagram of ChildBuilder package in Project# 4

Child Builder process is started by Main Builder. Its port number can be specified in the command line. After being created, Child Builder process sends a Ready message to Main Builder and waits for command.

If received a Build Request, Child Builder sends File Request message to Repository Server, asking for files cited in the build request. After retrieving the file, Child Builder begins to build project using Microsoft.Build.Engine assembly. Record the result, warning and error in a build log and sends it to Repository Server. If build succeed, Child Builder sends the test library and a Test Request message to Test Hardness.

After finishing all the building, Child Builder sends a Ready message to Main Builder and waits for command.

5.6 TestHardness

TestHardness package implements the Test Hardness Server.

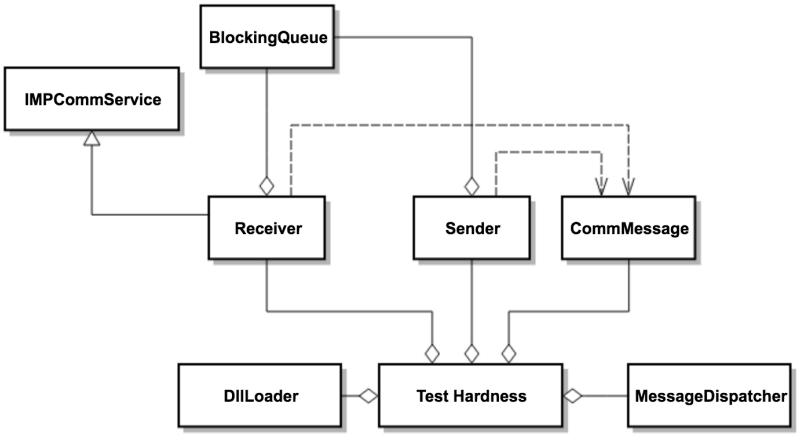

Figure 17 Class Diagram of TestHardness package in Project# 4

After starting the receive thread, Test Hardness waits command from Child Builder. If receives a Test Request message, Test Hardness begins to load and execute each test library it received, logging the information of testing in a Test Log, and sends the Test Log to Repository Server. Below is a sample Test Log

```
Test Log
Author: Beier Chen
Test Name: Test3
Test Begin Time: 2017/12/8 20:56:37
Test Library Path: C:\Users\BeierChen\source\Project4\Project4\StorageOfTestHardness
Test Library Name: TestDriver3.dll
Test Result: fail
```

Figure 18 A sample Test Log

5.7 Other Packages

Above are the major packages of Project#4, the remaining packages contains methods to support the major packages.

1 DllLoader

DllLoader package is used by TestHardness package to load test library. It provides the method to load libraries from a specified path using dynamic invocation.

2 FileMgr

FileMgr package is used by Repo package to get file name, directory and subdirectory.

3 XMLHandler

XMLHandler package is used by ClientGUI package and BuildServer package. ClientGUI uses XMLHandler to make a XML build request structure and save it to a XML document. BuildServer uses XMLHandler to parse the XML build request to retrieve the test name and cited files name.

4 Environment

Environment package is used by ClientGUI, Repo, BuildServer, ChildBuilder and TestHardnss. Environment define the port number, endpoint, storage directory for Client, Repo, Build Server, Child Builder and Test Hardness.

5 PrintTool

When running Project# 4, there is one WPF window application, more than five console applications running at the same time. To demonstrate Project# 4 more clear, PrintTool

package provides several printing methods that can change the color or add symbols to emphasize the string.

7 Critical Issues

7.1 Performance under heavy work load

In real use, the Remote Build Server would be under heavy work load, especially before customer demos and releases.

Proposed solution
To mitigate the pressure, the project uses Message-Passing Communication, which simplifies a lot of the designs, supporting functional changes and additions, as well as performance tuning.

Additionally, the build server uses a "Process Pool" to improve performance under heavy load. That is, a limited set of processes spawned at startup. The build server provides a queue of build requests, and each pooled process retrieves a request, processes it, sends the build log and, if successful, libraries to the test harness, then retrieves another request.

To scale out the build process for high volume of build requests, the Build Server should be cooperated Repository Server and Test Hardness Server with much more functions. Repository Server should have check-in and check-out and versions functions to support Build Server automates the builds process of high volume requests.

The Test Hardness Server should be equally able to finish high volume of testing, built by the Build Server. Test Hardness Server should have a Process Pool as Build Server, to spawn a set of tester at start up, and Test Hardness can queue the test request and distributes them to available testers.

7.2 Different source code language

Project# 4 is built with C#, but Build Server may have to build projects of different language, such C++ and Java.

Proposed solution
To support building different source code, Build Server should use different Configuration Set accordingly. And the Test Hardness should also have tester for different languge.

7.3 Single message structure

Using a single message structure for all message conversations between clients and servers can simplifies the message processing and provides a better

8 Comments and Deficiencies

8.1 Changes to the core concept

When wring the OCD in project# 1, many core technologies haven't been introduced. Before Message-Passing Communication is being used, the Project# 2 uses a simple file copy, and only all the component working on the same project. And as WPF haven't been introduced either, the mock Client cannot browse file and package test libraries.

In Project# 3, I tried to establish the communication channel between GUI and Repository but failed, because I didn't quite understand STA and Message Dispatcher.

8.2 Deficiencies of the final project

8.2.1 Client GUI function

In Project#4, the Client can't view build log and test result, which are only sent to the Repo. A more functional GUI would be as below:

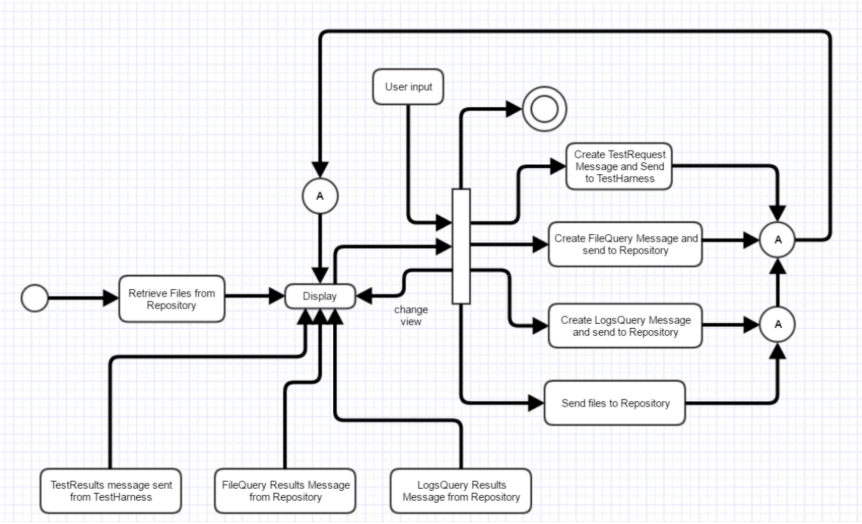

Figure 19 A more functional GUI[8]

8.2.2 Process Pool for Test Hardness

In Project# 4, the Test Hardness is a simple process. It don't have the process pool to spawn up several tester to perform testing concurrently, so its performance under heavy load might be poor.

[8] Sources: http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Midterm/MTf16/MT1f16-InstrSol.pdf

8.2.3 XML request

In Project# 4, only the build request made by Client is XML structure string, the build request sent from Main Builder to Child Builder, and test request are simply List<string>.

8.2.4 Different language support

Project# 4 don’t support building project in other language, only support C# code.