12/05/2017

INSTRUCTOR: Dr. Jim Fawcett

Vikrant Santosh Bhopatrao

cse 681 (project 4)

BUILD SERVER

DESIGN DOCUMENT

**TABLE OF CONTENTS**

1. EXECUTIVE SUMMARY …………………………………………………………………………………………………. 2
2. INTRODUCTION …….……………………………………………………………………………………………………… 3
3. USERS AND USES ……….……………………………………………………………………………………………….... 4
   1. Types of users …………………………………………………………………………………………………….... 4
4. DESIGN …………………………………………………………………………………………………………………………. 5
   1. Core Build Server ……………………..………………………………………………………………………… 5
   2. Message Passing Flow …...……….………………………………………………………………………….. 6
5. PACKAGE DIAGRAM ……………………………………………………………………………………………………… 7
   1. Mother Builder & Child Builder ……………………………………………………………………………… 7
   2. Federation …………………………………………………………………………………………………………….. 8
6. CLASS DIAGRAM …..………………………………………………………………………………………………………. 9
7. ACTIVITY DIAGRAM……………………………………………………………………………………………………… 10
8. TASKS ………………………………………………………………………..……………………………………………….. 11
9. USER INTERFACE ……………………………..…………………………………………………………………………. 12
10. ISSUES ………………………………………………………………………………………………………………………… 13
    1. Performance ……………………………………….…….……………………………………………… 13
    2. Ease of Use ……………………………………………..………………………………………………… 13
    3. Accuracy …………………………………..………………………………………………………………. 13
    4. Single Message Structure…………………………………………………………………………… 13
11. PROJECT DEFIENCIES AND FUTURE DEVELOPMENT ……………………………………………………. 14
12. REFERENCES ……………………………………………………………………………………………………………….. 15
13. **EXECUTIVE SUMMARY**

In this project, we will create a Remote Build Server, capable of building C# libraries, using a process pool to conduct multiple builds at the same time. The implementation is accomplished in the following stages.

* A local Core Build Server is implemented that communicates with a mock Repository, a mock Client, and a mock Test Harness. Initially the communication between these servers is made using simple multi-threaded blocking queue. The actual building of source files and generating log files is implemented in this build server.
* A message passing communication channel based of WCF(Windows Communication Foundation) is implemented and a Remotely accessible Build Server was developed.

* A pool process is implemented which spawns a number of child builders. These child builders obtain the test requests from the mother build server and can build multiple test requests in parallel.
* The final stage is to develop a GUI based on WPF( Windows Presentation Foundation) which will act as a mock client and support building of test requests.

Finally, we will have a federation of servers capable of remotely building source files provided by the user from the GUI, test them, and provide build logs as well as test logs to the Repository.

**Critical Issues:** Building source code using more than one language, scaling the build process for high volume of build requests, and using a single message structure for all message conversations between clients and servers. All of these issues have viable solutions.  
  
The Build Server will function as one of the principle components of a Software Development Environment Federation, the others being Repository, Test Harness, and Federation Client. Building these other Federation parts is beyond the scope of this development.

**Top Level Packages:**

* Test Request Creator and Parser
* DLL Builder
* DLL Loader
* Message Passing Comm
* Client Remote File Name dispatcher

**2. INTRODUCTION**

In big software systems consisting of million lines of code, containing thousands of packages, the complete software is divided into small partitions. Each of these partitions are then thoroughly tested and added to the baseline. Also after a certain period of time, when a newer version of an existing partition is to be uploaded to the baseline it is also tested by acquiring all its dependencies and then added to the baseline depending on the test result. This becomes a very complicated and a very slow process if it is a very large software system. For this reason, a continuous integration server is used. Whenever new code is created for a system, it is tested also with all its dependencies by this CI server and uploaded to the current baseline depending on the test result.

When working on a big software system, different developers are working on different partitions of the code with may be dependent on each other. So, when a developer submits his code to the repository, he modifies some of the elements which affect other parts of the repository. Thus, if the updates partition is not immediately tested and is not working satisfactorily i.e. contains some issues it will continue to cause problems to the baseline as it already been added to it. Therefore, a continuous integration server is needed which will automatically build and test the updated code with minimum delay and prevent any type mishandling occurring in the baseline.

To create a CI server, we will need a group of servers to carry out different tasks such as dependencies finder, tester, logger, test library creator, test request query handler, etc. For this reason, we will create a total of four servers each providing different functionality and a communication channel for their integration.

1. **Client Server**: To receive and process the test request from the client.
2. **Repository**: To store all the paths of the current source code and providing the required files for building and testing.
3. **Build Server**: To detect a test request and create a machine language program for testing and provide build results.
4. **Test Harness**: To fetch the compiled program from the build server, perform tests and provide the test result.

**3. USERS AND USES**

The primary use of the build server is to build, and test new packages or source code provided by the developer thus providing continuous integration to the software development process. The secondary use of the build server will be to check the status of the software system and also the stability of the system by the quality assurance engineers. Another use of the build server will be by the program manager to keep a track of the overall working of the system and other statistics.

* 1. Types of Users
* Developers

Developers of a large software system keep building improved versions of existing packages depending on the feedbacks from the previous versions of the software. They may also make some additional packages to improve the overall functioning of the software. All these new and updated packages are uploaded by the developer in the repository for testing. Once tested they are added to the current baseline if passed.

* Project Manager

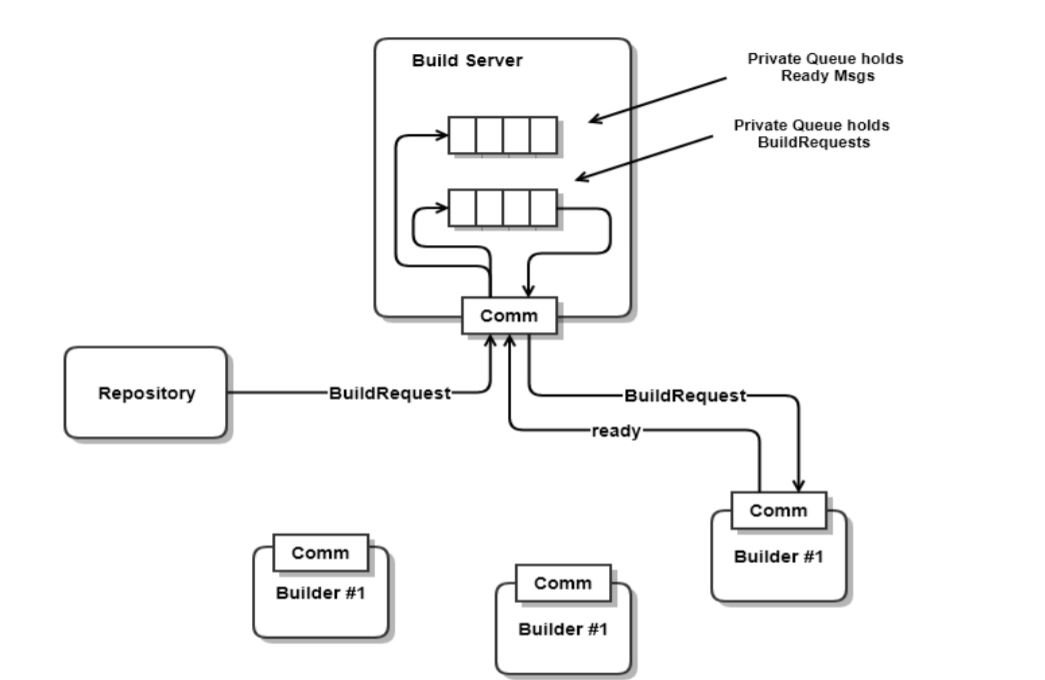
The Project Manager is a less frequent user of the build server than the developer. The Project manager will have a different user interface than the developer for accessing the build server. The manager will be able to see the overall progress of the software such as the number of successful builds carried out by each developer, current baseline of the software and information of the current packages in the software.

* Quality Assurance Engineers

Quality Assurance teams manage the baseline of a large software system. This means QA might run some very demanding builds on huge swaths of packages. For this, the build server needs to be very fast! The actual use will be very similar to that of the developer, they log in, queue up some test requests, and come back later for the results. They need to be able to input XML test requests, and view results afterwards. It is important to note that these XML requests might be much larger than those used by the developers. Previous XML test requests should remain available since QA reuses these quite often.

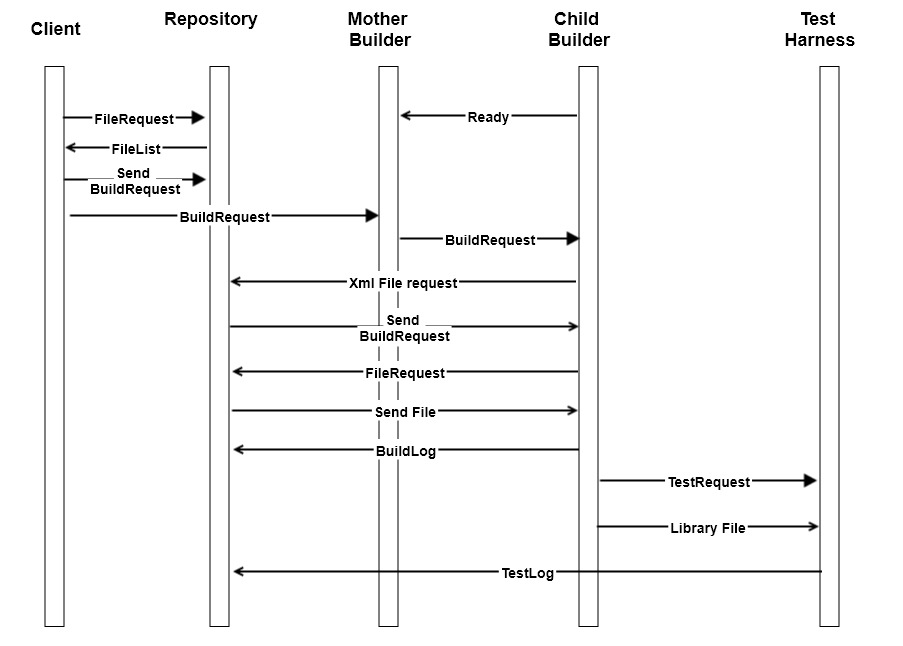
1. **DESIGN**

**4.1 CORE BUILD SERVER**



* The core build server consists of a mother builder and a number of child builders.
* Message passing communication channel is used to communicate between them.
* The mother builder consists of a ready queue which stores ready messages from the child builder and a test request queue to store the test requests send by the repository.
* The mother builder retrieves both the queues, creates a build request messages and sends it to the respective child builder.
* The process of building DLLs, generating log files is all done by the child builder and not the mother builder.
* The purpose of the mother builder is just to start the child builders and provide test requests to them.

**4.2 MESSAGE PASSING FLOW**



Windows Presentation Foundation is used for communication between different processes in this project. The message consists of a type attribute, a command attribute, an author attribute and also an argument attribute which can contain a list of strings.

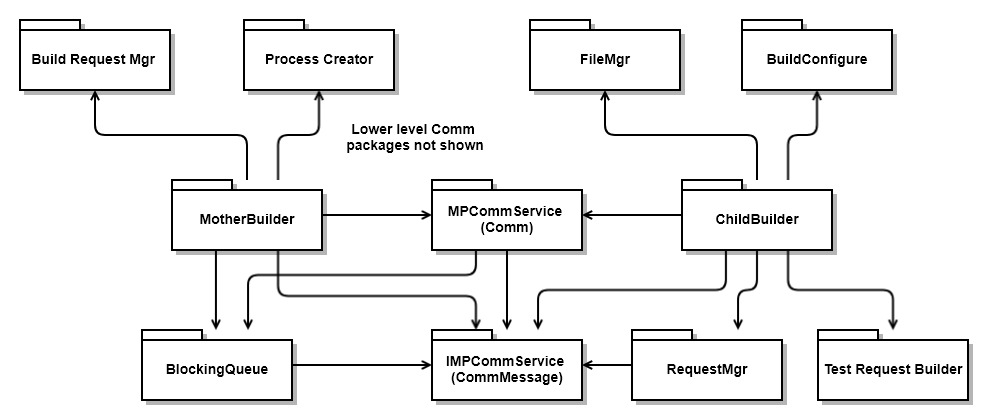
For sending a string to a process such as test request name or source file names, the argument attribute is used. The command attribute is used by the receiver to identify the type of process to be executed on receiving the respective message.

The type of the message can be “connect”, “request”, ”reply”, ”closeSender” and “closeReceiver”. The connect message is the first message that is sent in order to establish a connection. The reply and request messages are used for interaction between the processes. The closeSender message type is used to shut down the sender port of the current process and the closeReceiver message type is used to close the receiver port of the destination process.

File transfer is also implemented using message passing between these processes. A part of the file to be sent is stored in an array of bytes by the sender and then the array of bytes is sent as a message to the receiver. The receiver stores the received array in a local file and the process continues until the entire file is copied.

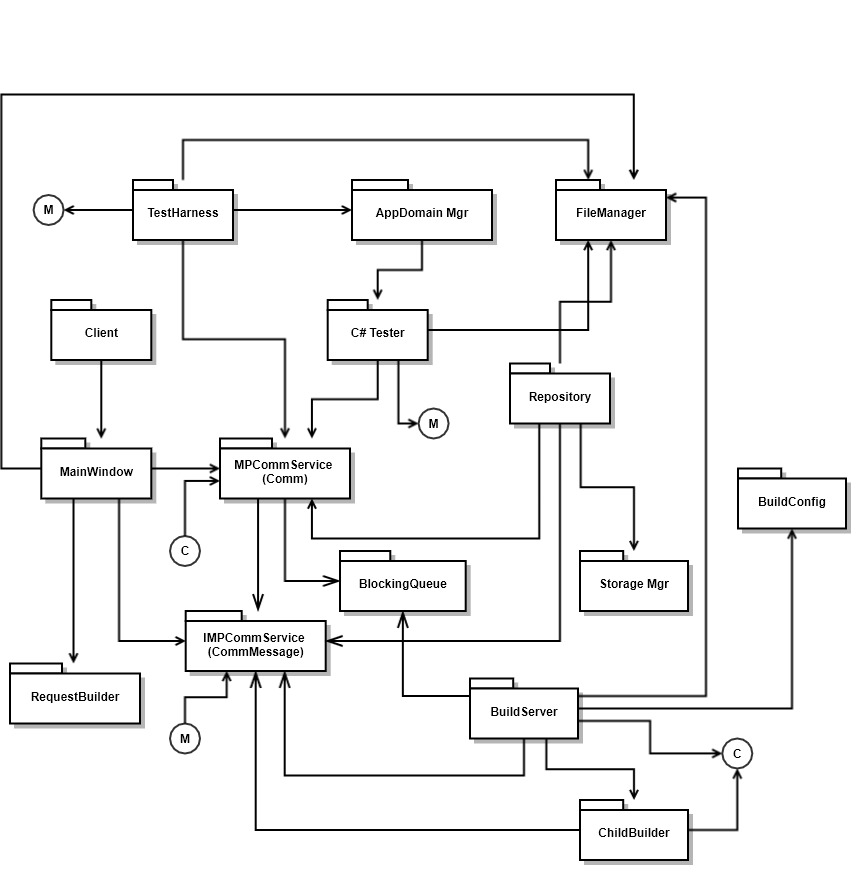
1. **PACKAGE DIAGRAM**

* **MOTHER BUILDER & CHILD BUILDER**



The above diagram represents the packages used by the core Build Server i.e. the mother builder and the child builders.

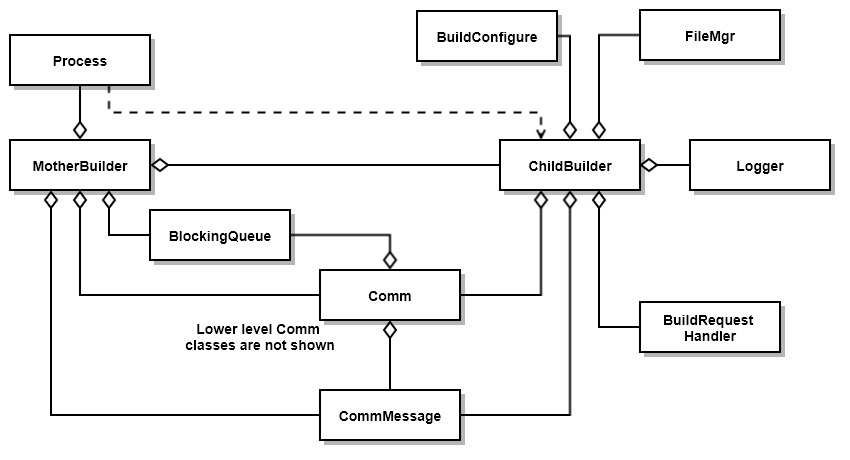
* The build request manager package is responsible for fetching test requests and ready messages from their there respective blocking queues a sending the build request to child builder.
* The process creator package is responsible for spawning the required no. of child builder processes.
* The blocking queue package is used to efficiently send and receive messages and also to store test request and ready queues.
* The file manager package of child builder is responsible for file storage and sending files to other processes.
* The Build Configure package is used to build the provided source files and generate logs.
* The RequestMgr package decodes the message received by the child builder and takes suitable action.
* The test request builder package is used to create a test request for the test harness if build succeeds.
* The MPCommService and IMPCommService package implement the message passing environment required to effectively send and receive messages.
* **FEDERATION**



The above diagram represents packages used by the entire federation in this project. The tasks performed by these packages are described in the “Tasks” section.

The MPCommService, IMPCommService and are common packages which are used by all the processes for communication.

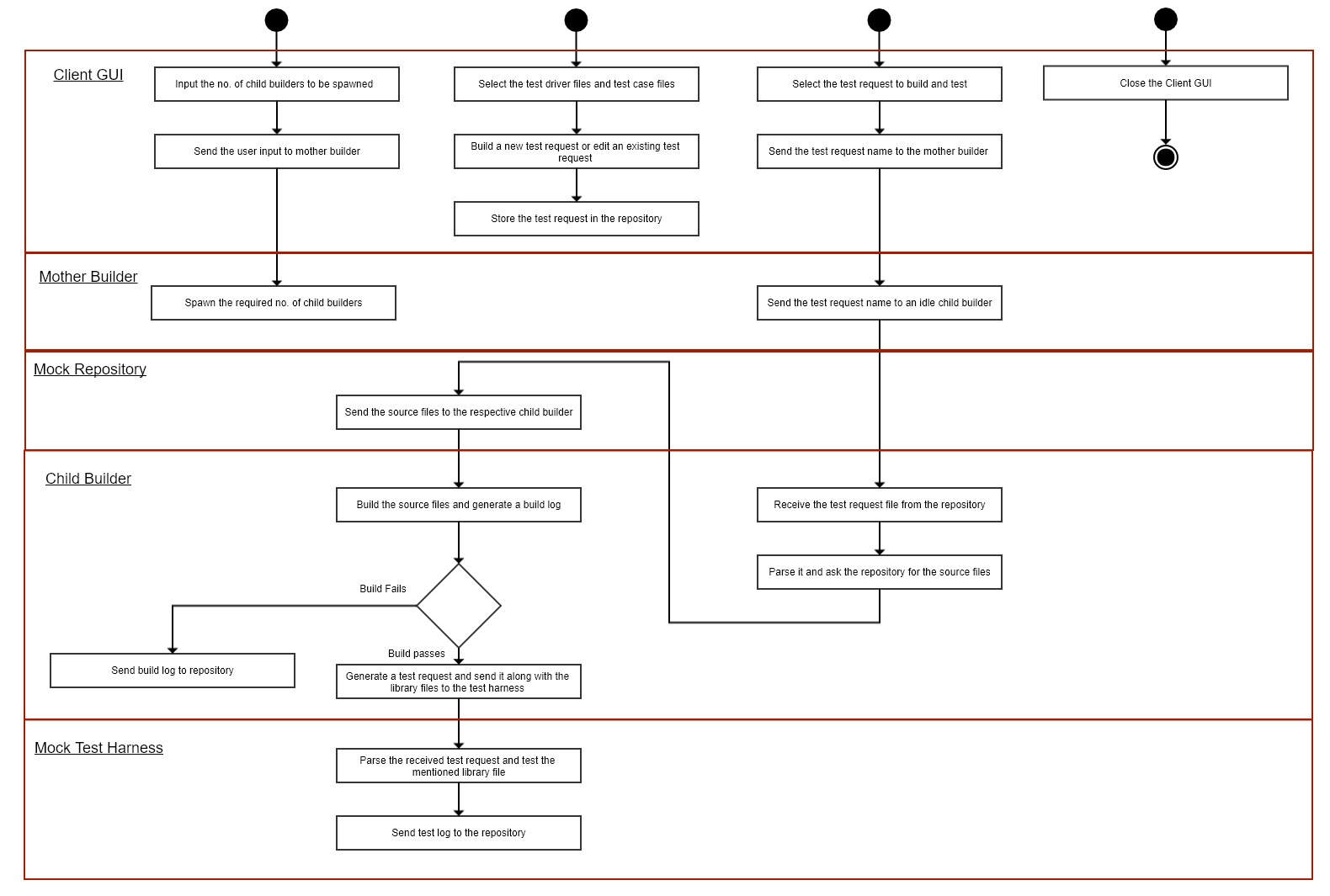
1. **CLASS DIAGRAM**



The above Diagram represents the classes used in the Core Build Server.

* The Process class uses the command prompt to start child builder processes.
* The blocking queue class is uses locking and “monitor.waiting” mechanisms to provide sequential queueing and dequeuing of messages.
* The BuildConfigure class uses “csc” command in to build source files from the command prompt.
* The file manager class “System.IO.Directory” class to create child builder’s storage, access the storage and also to delete files.
* Logger class uses stream writer to create a log file and write to it the logs provided by build configure class.
* Build Request handler uses a switch case to determine the received message’s type or command and call suitable method to take the respective action.
* CommMessage class uses “WCF” to create a sender port, a receiver port and a communication channel based on WSHttpBinding format.
* The IMCommMessage class defines Service Contracts and Data Contracts for communication.

1. **ACTIVITY DIAGRAM**

****

1. **TASKS**

The above activity diagram describes the flow of all the tasks performed in this project.

The tasks performed by the **Mother Builder** are:

1. **Enqueue Test Request**: The mother builder receives test request name from the Client and enqueues it in its request queue.
2. **Enqueue Ready message**: The mother builder receives ready message from the child builder and enqueues it in its ready queue.
3. **Spawn child builders**: It receives the no. of child builders to be spawned from the client and starts respective no. of child builders.
4. **Send Test Request**: If both the test request queue and the ready queue is not empty, then the mother builder send the test request name from test request queue to the child builder port in the ready queue.

The tasks performed by the **Child Builder** are:

1. **Parse Test Request:** Parse the received test request file from the repository and ask the repository to send the required source files.
2. **Build Source files:** Start a hidden process to build the given source files.
3. **Generate Logs:** Generate logs and send them to the Repository.
4. **Create Test Request:** If build succeeds create a test request and send the test request along with the libraries to the test harness.

The tasks performed by the **Client GUI** are:

1. Build or edit a test request.
2. Send the test request to repository.
3. Receive source file names from the repository.
4. Close the pool processes.
5. Send a build request to mother builder.

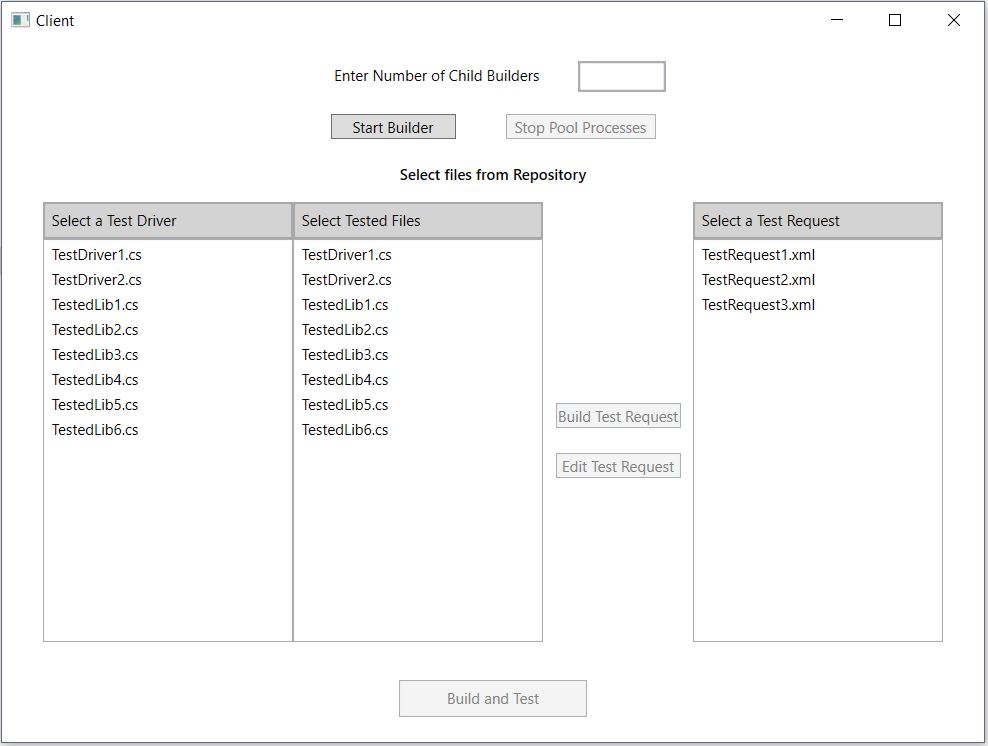
The tasks performed by the **Mock Repository** are:

1. Accept test requests from the client
2. Send test request and source files to other servers.
3. Accept build log from the child servers and test log from the test harness.

The tasks performed by the **Mock Test Harness** are:

1. Load all the libraries mentioned in the test request.
2. Execute all the required tests
3. Send test log to the repository

**9. USER INTERFACE**



For interfacing the build server, a GUI is provided to the user. The GUI will provide a simple and an easy access to the build server. The GUI consists of a textbox which where the user can enter the number of child processes the user needs to be spawned. On clicking “Start Builder” button, given number of child processes will be spawned. “Stop Pool Processes” button is provided in case the user needs to stop the building process and close all the child builders. Two list boxes are provided which displays all the source files available in the repository. The user can select a test driver and several test files to be tested and create a build request by clicking the “Build Test Request” button. The user can also edit an existing test request by selecting the test request to be edited from the list box and clicking the “Edit Test Request” button. Finally, to build and test a test request file, the user can select the filename from the list box and click the “Build and Test” button. On closing the GUI, a quit message is sent to all other processes and are automatically closed.

**10. CRITICAL ISSUES**

* 1. Performance

The build server should be able to build code libraries fast. For this the complexity of the system should be a small term. For big software systems, a tremendous amount of test requests will be added to the build server each day. Thus, the build server should efficiently parse the test requests and build the test libraries.

**Solution**: The build server code is kept simple with minimum lines of code. Also, proper message passing communication between the different servers is established for efficient communication.

* 1. Ease of Use

The build server should be easy to use i.e. it should be easily accessible to the user. The user should be able to send test requests and receive log files with ease by simply interacting with the GUI.

**Solution**: For this reason, we have built a clean and simple GUI for the user which can easily be interfaced by any software developer or IT professional.

* 1. Accuracy

The build server should accurately build test libraries to send them for testing to the test harness. If the build server fails to build the test code then there should really be a problem in the code that the developer can rectify and if it builds then the code should really be working properly.

**Solution**: An Xml file containing the test request is used to accurately provide information to the build server. The build server properly parses the test request and fetches the correct toolchain from the repository. All the dependencies are properly fetched so that the code can be completely built.

* 1. Single Message Structure

A single message structure that works for all messages is used in the Federation. All the servers in the federation need to send different messages to each other. These messages may include files names, files contents, specific commands, list of texts, and can be addressed to any of the servers in the federation.

**Solution:** A message that contains To and From addresses, Command string or enumeration, List of strings to hold file names, and a string body to hold logs will suffice for all needed operations.

**11. PROJECT DEFICIENCIES & FUTURE DEVELOPMENT**

* **Deficiencies:**
  + The current build server can only build and test c# source files and have no support for any other languages.
  + The GUI provided to user has very limited functionality and doesn’t have a login in screen which can provide only authorized personnel access to the build server.
  + The Client server, repository server and the test harness server are just mock servers and don’t provide facilities a complete server will provide of the respective type.
  + All the servers communicate using a local host and do not have a remote IP to run on different systems.
* **Future Development:**
* Build a robust build server which can build several types of source files such as C++, C, Java, etc.
* Build a process pool for the test harness similar to that of the build server so as to perform parallel testing on different source files at once.
* Provide more functionality to the user on the GUI.
* Build a fully functioning Repository capable of providing check-in, check-out facility to the user and providing versioning of the source files.

**12. REFERENCES**

1. Dr. Jim Fawcett’s website

<https://ecs.syr.edu/faculty/fawcett/handouts/webpages/CSE681.htm>

1. <http://deviq.com/build-server>
2. Wikipedia Website

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