REMOTE BUILD SERVER

Operational Concept Document

Instructor: Prof. Jim Fawcett

Course Name: CSE 681 Software Modeling & Analysis

Sonal Patil [SUID 997435672]

Email: spatil06@syr.edu

Date: 12-06-2017

TABLE OF CONTENTS

Table of figures 1

1. Executive summary 3

2. Introduction..........5

    2.1. Background 4

    2.2. Application Obligations 4

    2.3. Organizing Principles 4

    2.4. Key Architectural Idea 4

3. Users & Uses..........5

    3.1. Software Developers 5

    3.2. Managers 5

    3.3. Quality Assurance Team 5

    3.4. Release Team 6

    3.5. Customers 6

4. Modular Strucutre..........9

    4.1. Overall Structure 9

5. Application Activities & Architecture 12

    5.1. Client [GUI] 12

    5.2. Repository 14

    5.3. Process Pool 15

    5.4. Mother Builder 16

    5.5. Child Builder 18

    5.6. Test Harness 20

    5.7. Comm 22

    5.8. Comm Messages 23

    5.8. Designed User Interface 25

6. Evolution 3029

7. Critical Issues..........31

8. Conclusion 32

9. References 33

TABLE OF FIGURES

Figure 3.1 Activity Diagram for QA Team 7

Figure 3.2 Customer-Build Server Connection Diagram 8

Figure 5.1 [a] GUI Activity Diagram 12

Figure 5.1 [b] GUI Class Diagram 13

Figure 5.2 [a] Repository Activity Diagram 14

Figure 5.2 [b] Repository Class Diagram 15

Figure 5.3 Build Server with Process Pool 16

Figure 5.4 [a] Mother Builder Activity Diagram 17

Figure 5.4 [b] Mother Builder Class Diagram 18

Figure 5.5 [a] Child Builder Activity Diagram 19

Figure 5.5 [b] Child Builder Class Diagram 20

Figure 5.6 [a] Test Harness Activity Diagram 21

Figure 5.6 [b] Test Harness Class Diagram 22

Figure 5.7 Comm Class Diagram 23

Figure 5.8 List of Comm Messages 24

Figure 5.9 [a] Browse Button GUI [Tab 1 – Main] 25

Figure 5.9 [b] Clean GUI [Tab 1 – Main] 26

Figure 5.9 [c] GUI [Tab 1 – Main] 26

Figure 5.9 [d] GUI [Tab 2 – Logs] 27

Figure 5.9 [e] CodePopUp GUI 28

1. EXECUTIVE SUMMARY

This Operational Concept Document details the design concept of remote build server which is nothing but a centralized, stable and reliable environment for building distributed development projects. Few of the advantages of build server include:

- Acceleration of compile and link processing
- Elimination of redundant tasks
- Elimination of dependencies on key personnel
- Improvement of product quality
- Minimization of "bad builds"
- Having history of builds and releases in order to investigate issues
- Saving time and money - because of the reasons listed above

Following sections describe our remote build server which automates the tasks of building C# code using process pool concept to conduct multiple builds in parallel, creating C# test libraries, testing them in Test Harness and logging results of every such operation. Each time a new code is to be added or existing code is to be changed, build server is triggered by the user of the server. In response, the server will pull the new code from its repository storage build it and send built libraries to Test harness to run all the unit tests if the build is successful. The end result is the build of the most current code.

This increases predictability by enforcing source control, making it possible to flag issues, notify all users (developers, managers, QA team etc.) of server quickly if there are conflicts or missing dependencies.

The advantages of the build server extend beyond developers. With a build sever in place; QA teams always know which version they need to test. This allows them to streamline their own workflow by introducing automated testing tools. It helps ensure that the same dynamic link library is used for all builds and that out-of-sync check-ins don’t lead to failure during quality assurance (QA) testing.

Managers can use such a server for their code sign off operations while release teams benefit from it while rolling in or out the production versions of the software.

Few critical issues can be identified for this build server:

1. Building Source code using more than one language
2. Scaling the build process for high volume of build requests.
3. Use of single message structure for all message conversations between client and servers.
4. Increased storage space to maintain long build logs and all the build versions

All of these issues have viable solutions.

Our Software Development Environment Federation is divided into following main modules which perform specific tasks:

▪ GUI	: Medium for users’ interaction with build server
▪ Repository	: Storage area which holds build input-output related files
▪ Mother Builder	: Holds the multiple build requests & implements process pool concept
▪ Child Builder/s	: Building of C# files is done here
▪ Test Harness	: Performs testing operation on generated test libraries

The Build Server will function as one of the principle components with combination of Mother Builder and Child Builder. The others being Repository, Test Harness, and Federation Client [GUI]. Building these other Federation parts is beyond the scope of this development.

Section 2 of this document explains about the build server obligations and its architecture. Section 3 illustrates different users of the build server and their use cases while section 4 and 5 go into details of build server activities and design. Section 6 discusses the evolution in my design from OCD 1 to Project 4 OCD. Section 7 states some of the critical issues related to my design and provides their viable solutions.

This server and its prototypes are both developed in C# using .Net framework class libraries and Visual Studio 2017.

2. INTRODUCTION

2.1. Background

Research spanning over the entire life of software industry has shown us harmful impacts of disconnects between important activities e.g., planning, development and implementation. The bigger the project more is the need to streamline and automate the important activities to mitigate risks in software development. Having highly efficient Build servers is one of the many such risk management strategies. Build Server is like heart beat monitor. It tells you how healthy your system is and it also notifies you in advance if your system is going to break down.

2.2. Application Obligations

The main responsibilities of this system are to build the build requests, create the build logs, trigger the Test Harness and notify users about the build & test results. Such a system needs to be scalable, efficient and secure with following obligations:

1. It should be machine independent
2. It should store the files sent from users to the repository
3. It should pull the required dependency files from the Repository on every build request
4. It should use process pool concept to handle multiple build requests in parallel
5. It should parse build request (XML File) & attempt to build request, create build logs, test libraries
6. It should also trigger Test Harness on a successful build and send test libraries to it
7. It should save build & test logs to the repository and notify users of the build & test results
8. It should use WCF for all the communication between federation packages

2.3. Organizing Principles

The organizing principle of the system is to perform the functionality of the Build Server, capable of building C# libraries using a process pool and testing them. It will rely on the mock client (GUI), mock repository and mock test harness to perform its functions.

2.4. Key Architectural idea

The key architectural idea is, building a GUI that'll interact with the user and other federation packages to transfer the messages sent by users. On command from user all other federation packages will start performing their own operations in order to support the main component of federation i.e. Build Server. Together all federation packages will perform the operation of accepting build requests, build and test them and give the feedback to user in terms of build logs and test logs. For all communication between packages, we must use WCF.

3. USERS & USES

Following types of users can interact with the Build Server:

3.1. Software Developers –

Developers' work is to develop a piece of software as per the customer requirement. When a developer makes changes in his or her code or introduces a new code, it shouldn't affect any other team member's work or shouldn't cause whole system to break down.

Uses – The build server can help developers to build & execute the test cases on their code and notify the result of build request to all team members and other teams working on same project. Using the build server thus helps maintain the source control and to pinpoint the cause of build or test.

3.2. Managers –

Manager is the person who works for both development and deploying (staging) stages. He has access to the development code which he'll then deploy for staging branches of the repositories for QAs.

Uses – Manager will get notified from the build server about the developer teams' builds and test results. This will help him keep track of the project's progress. Manager can run his own build requests on developers' code and get the build and test result from the build server. He can then decide whether to move the current clean code to the staging branch and do code sign-off for QAs.

3.3 Quality Assurance team –

Quality Assurance (QA) team works on project level testing. They need to check the developer team's code before sending it out for final sign off.

Uses – Build server can be useful for QA team to run their own high-level test cases on the staging branches' (signed off by Managers) code. If the bug is found, ticket filed against the developer will keep track the upcoming code fixes. QA can then take such fixed code and send requests to build server to see the validity of the fix. They will notify the developer team if any build fails or test fails. Build server also helps QAs keep track of all those builds and their results with the help of version ID provided in the log.

Once QA team gets the clean build and test results from build server and Test Harness, they can proceed with signing off the code for release team. Following activity diagram explains how the QA team is going to use the build server:

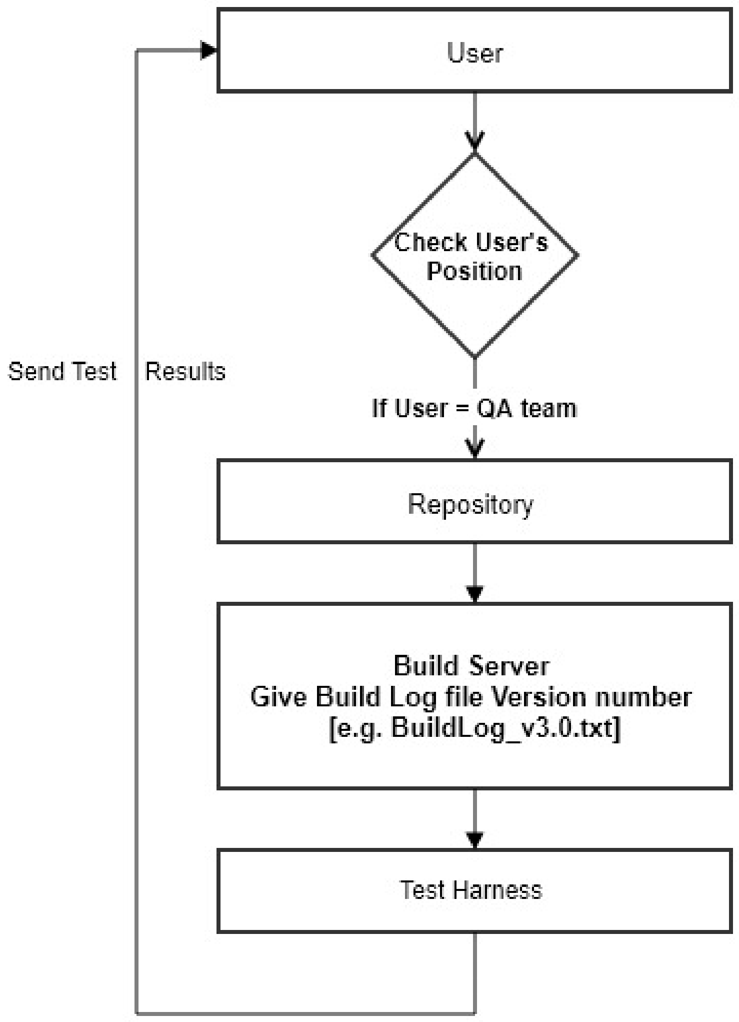


Figure 3.1 Activity Diagram for QA Team

3.4. Release Team –

Release teams decide to roll-out or roll-back the code form production branches after all the above users sign-off the software.

Uses – They need to be able to build and test key tests in case customers report some problem with the final software.

3.5. Customers –

Customers are the users who give the developers the job of software development. Their only Interaction with the build server is via cloud [future development not implemented in this project]. Release teams push built software libraries to cloud for users to download. Build version info generated by build servers which is kept in the cloud to help users track the version of software they are using.

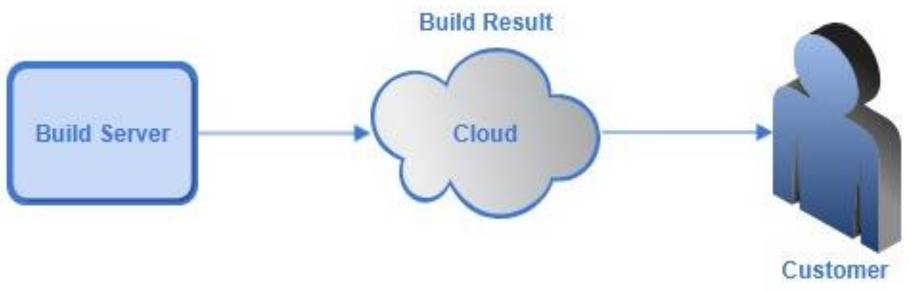


Figure 3.2 Customer-Build Server Connection Diagram

4. MODULAR STRUCTURE

4.1. Overall Structure -

Partition processing of top level packages describes each package included in system, how it works and what job is expected from that package along with communication between all packages.

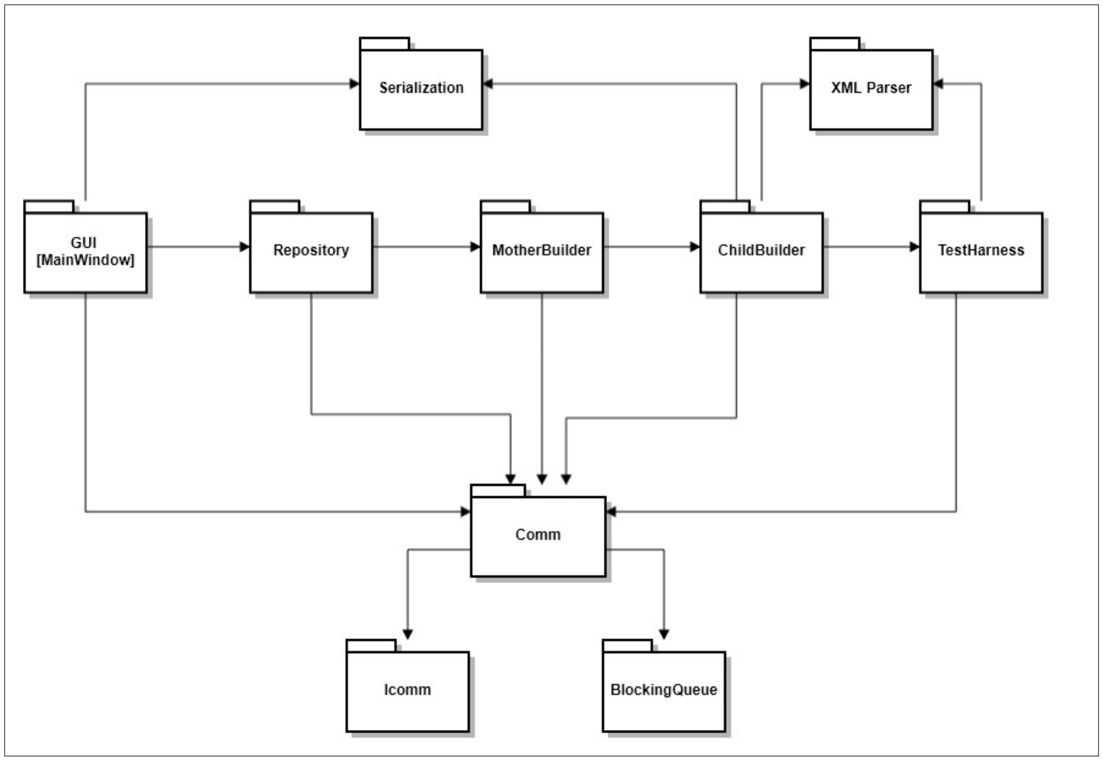


Figure 4.1 Top Level Package Diagram

1. GUI –

   Graphical User Interface package is used as a medium of communication takes place between user and build server. It allows user to give commands or send files on which he/she wants to perform build & test operation. In return GUI gives back responses it received from build server after build/test operation.

2. Repository –

   Repository package is nothing but storage space for build server. User sends files he/she wants to build to repository from its' own local storage. Whenever build server needs some dependency files while performing build/test operation, it goes to repository and asks for the files.

Our Builder Server performs build functionality using two packages – Mother Builder & Child Builder.

3. Mother Builder –
   Mother Builder package accepts Build Request files from Repository and implements process pool concept by spawning that many number of child builder processes to handle multiple build requests concurrently.

4. Child Builder –
   Child Builder receives a build request when he sent free message to Mother Builder. It starts building the build request by parsing it first and then using process class it builds that buildrequest. And at the end it produces the build log results for the user.

5. Serializer –
   Serializer package generates XML file in particular format using given file names. This package is used to generate Build Request xml file and also to generate Test Request xml file.

6. XML Parser –
   This package communicates with Child Builder & Test Harness. It will parse both XML files, Build Request xml file & Test Request xml file.
   When XML parser package receives a XML document, it will call the XmlDocument class to parse the given file. It will then retrieve the test drivers and other dependency files and sends the output to proper package.

7. Test Harness –
   Test harness package will accept the Test Request file sent from Child Builder and will extract the library files (.dll) by parsing that test request. It will then load the test library to find the test driver and performs test operation. It then writes the output result into test log file.

8. Comm –
   Comm package defines the way communication will be done between all packages. It demonstrates Message Passing Communication service created using Windows Communication Foundation. This Communicator service will get used by all clients & servers to communicate with each other except for Serializer and XML Parser.

9. Icomm –
   Icomm package implements Icomm interface which Defines a service contract that describes the operations, or methods, that are available on the service endpoint, and exposed to the outside

world. It also has a CommMessage class which defines Data Contract, message type and the message structure that gets used in the communication.

10. Blocking Queue –
    This package implements a generic blocking queue and demonstrates communication between two threads using an instance of the queue. If the queue is empty, when a reader attempts to dequeue a comm message then the reader will block until the writing thread enqueues an item. Thus, waiting is efficient. Here, blocking queue is implemented using a Monitor and lock, which is equivalent to using a condition variable with a lock.
    This package gets used by Comm for creating blocking queues for sender & receiver. Blocking queue holds the comm messages. It also defines the functionality of enqueuing and dequeuing of messages. The messages will get inserted & retrieved in string format from blocking queue.

5. APPLICATION ACTIVITIES & ARCHITECTURE

The purpose of presenting activity diagrams is to give clear idea about how activities are going to get performed in the software/application. If the system breaks down, activity diagram helps user to identify the root place of the break down.

5.1. Client GUI –

Concept –

The client is the Graphical User Interface that interacts with the users, conveys commands to the server and displays information about the result for the user. It is built using Windows Presentation Foundation [WPF].

Activities –

Following are the client activities –

1. It browse files from the users' local storage and allows user to send selected files to Repositorys' storage.
2. If user is performing operations multiple times then at some point he/she may want to see the currently present files in the repository. So client asks the repository to send file names of currenlty present files in its' storage and displays them for user.
3. After getting files in the repository, GUI builds the build request for user selected files and send it to the repository for storage.
4. User sends command for sending build request files to Mother Builder through GUI.
5. It allows user to give count of number of processes to spawn by Mother builder & it starts the Mother builder.
6. It also allows user to shutdown the process pool i.e. processes spawned by Mother Builder.
7. The result of build and test performed on particular build request gets displayed in the GUI for user.

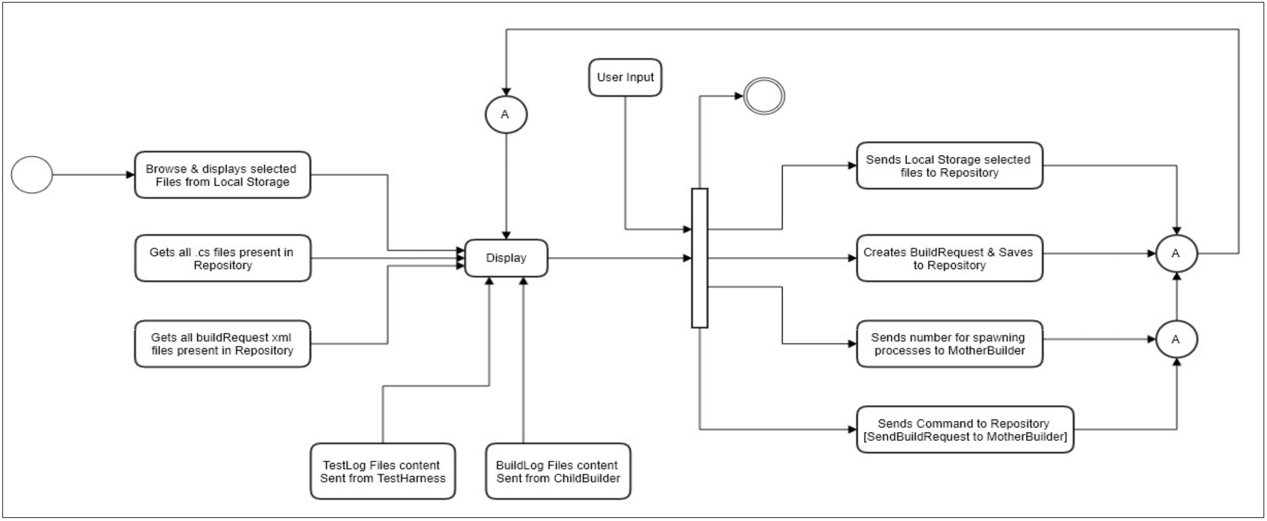

A
User Input
Browse & displays selected Files from Local Storage
Gets all .cs files present in Repository
Gets all buildRequest xml files present in Repository
Display
Sends Local Storage selected files to Repository
Creates BuildRequest & Saves to Repository
Sends number for spawning processes to MotherBuilder
Sends Command to Repository [SendBuildRequest to MotherBuilder]
A
A
TestLog Files content Sent from TestHarness
BuildLog Files content Sent from ChildBuilder

Figure 5.1. [a] GUI Activity Diagram

Structure –

The structure here defines the classes used by Client [GUI] and relationship between them.

1. Window – It is a base class from which MainWindow & CodePopUp class is derived. The class relationship between Window class & MainWindow, CodePopUp class is Inheritance.
2. MainWindow – It is class defined in the GUI which is inherited from Window class. It means MainWindow class inherits all the members of Window class except its constructor.
   All the activities mentioned above are performed in the MainWindow class only.
3. CodePopUp - This class is also inherited from Window class. It is used to display the content of build request. After generating build request and displaying it in the 3rd listbox of MainWindow, when user double clicks on the build request, content of that build request gets popped up in this CodePopUp Window. The class relationship here is Aggregation.
4. Serializer – MainWindow class owns some part of this class so relationship is Aggregation. This class is used to create build request xml file in specific format.
5. Comm – MainWindow class owns some part of this class so relationship is Aggregation. This class is used to create sender and receiver channel for communication. GUI sends messages to Repository, Mother Builder and receives from Repository through this message passing communication class only.

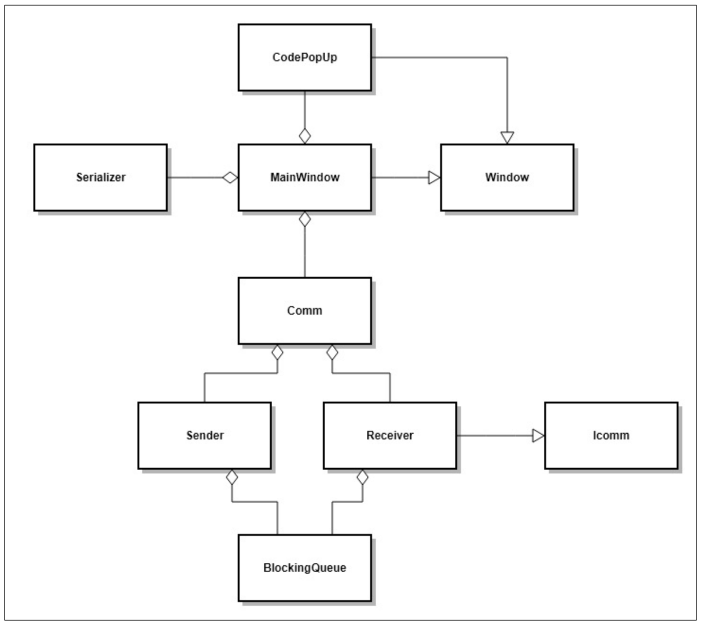


Figure 5.1. [b] GUI Class Diagram

5.2. Repository –

Concept –

The Repository is basically a storage space where all the files that are needed for building and testing build request get stored with the final result logs.

It allows GUI, Child Builder, Test Harness to perform check-in and check-outs of files.

These operations are gets performed through the WCF comm messages.

Activities –

1. It gets the file names that user wants to transfer from his/her local storage to Repositorys' storage as a comm message.
2. It uses upload model to copy those files from given path to its own storage directory. The writing of file is done block by block.
3. It gets built request generated by GUI as a string.
4. On command from client [GUI], it sends the .cs file names that are currenlty present in its storage directory using comm message.
5. On command from client [GUI], it sends all build request xml file names that are currenlty present in its storage directory using comm message.
6. On command from client [GUI], it sends buildrequest to Mother Builder using comm message.
7. It gets the result log file names. Build log file name from Child Builder & Test log file name from Test Harness.
8. It then again uses upload model to copy those files from given path to its own storage directory. The writing of file is done block by block.

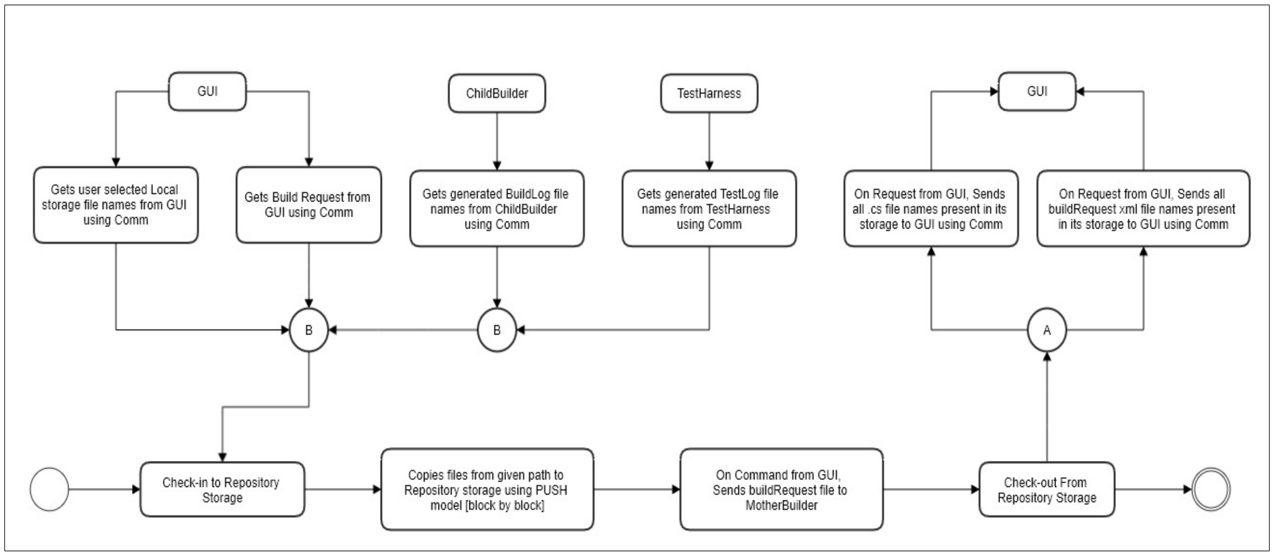

GUI
ChildBuilder
TestHarness
GUI
Gets user selected Local storage file names from GUI using Comm
Gets Build Request from GUI using Comm
Gets generated BuildLog file names from ChildBuilder using Comm
Gets generated TestLog file names from TestHarness using Comm
On Request from GUI, Sends all .cs file names present in its storage to GUI using Comm
On Request from GUI, Sends all buildRequest xml file names present in its storage to GUI using Comm
B
B
A
Check-in to Repository Storage
Copies files from given path to Repository storage using PUSH model [block by block]
On Command from GUI, Sends buildRequest file to MotherBuilder
Check-out From Repository Storage

Figure 5.2. [a] Repository Activity Diagram

Structure –

The structure here defines the classes used by Repository and relationship between them.

1. Repository – Repository class is the main class which handles all the repository activities defined above. It owns some part of comm class for message passing communication.
2. Comm – Repository class owns some part of this class so relationship is Aggregation. This class is used to create sender and receiver channel for communication. Repository sends messages to GUI, Mother Builder and receives from GUI, Child Builder, Test Harness through this message passing communication class only.

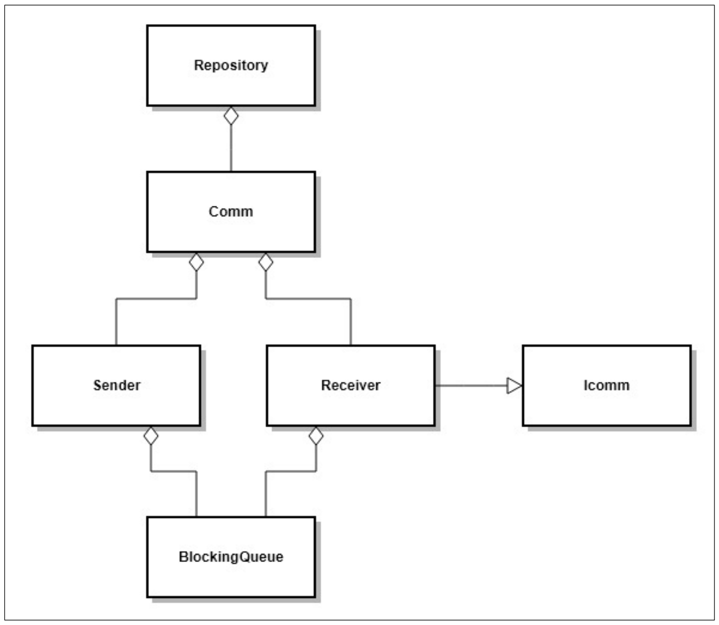


Figure 5.2. [b] Repository Class Diagram

5.3. Process Pool –

The process pool is the concept of spawning multiple processes at the same time. In this, multiple processes run parallelly to perform multiple operation simultaneously.

The reason of using process pool instead of thread pool is, in thread pool if one error gets occurred in one thread whole thread pool and so as whole process gets shut down as all threads run in the same process.

On the other hand, in process pool concept, each process has its own thread and all processes are independent so if any error occurs on any one of the thread of any process, all other processes don't get affected, only that particular process gets shut down.

Build Server we have designed uses the process pool concept.

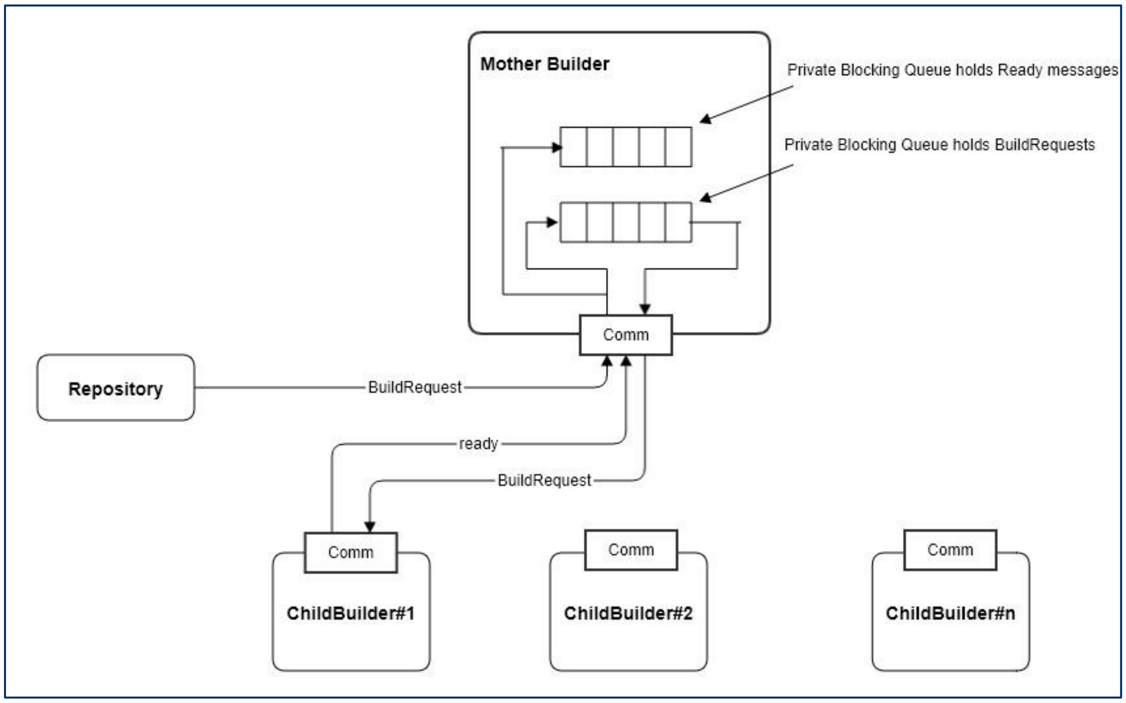


Figure 5.3. Build Server with Process Pool

Activities of Process Pool –

1. Mother Builder accepts multiple build requests from Repository and enqueues them one by one in its blocking queue.
2. Mother Builder process then spawns multiple [number given by user] Child Builder processes. Each Child Builder performs same functionality of building build request and producing test library file for that build request. But difference is each child builder builds one build request at a time & this helps achieving multiple builds for multiple build requests sent by user.
3. For this, Mother Builder has 2 separate blocking queues one for receiving ready messages from Child Builder and one for receiving build requests from Repository.
4. When Child Builder doesn't have any build request to build it sends ready message to Mother Builder's ready blocking queue.
5. Mother builder then checks the ready blocking queue and sends the build request to that particular child builder from whom it received ready message.

5.4. Mother Builder –

Concept –

The main build server functionality gets performed by two main classes Mother Builder & Child Builder. Mother Builder performs main two functionalities –

1. It accepts build request/s sent from GUI through Repository using WCF.
2. It creates the process pool for building multiple build requests.

Activities –

1. Mother Builder accepts the build requests sent from GUI through Repository usign WCF.
2. It then spawns multiple [number given by user as user input to GUI and received at Mother Builder through comm message] Child Builder processes at different ports.
3. The receiving thread of Mother Builder keeps dequeuing messages from ready blocking queue to get ready message from Child Builder/s.
4. It checks the message body of comm message received from Child Builder through ready blocking queue.
5. If the comm message body is 'ready' it sends the build request from it's another blocking queue to that particular Child Builder.
6. If no ready string in message body then it keeps checking the ready blocking queue till te gets ready message.

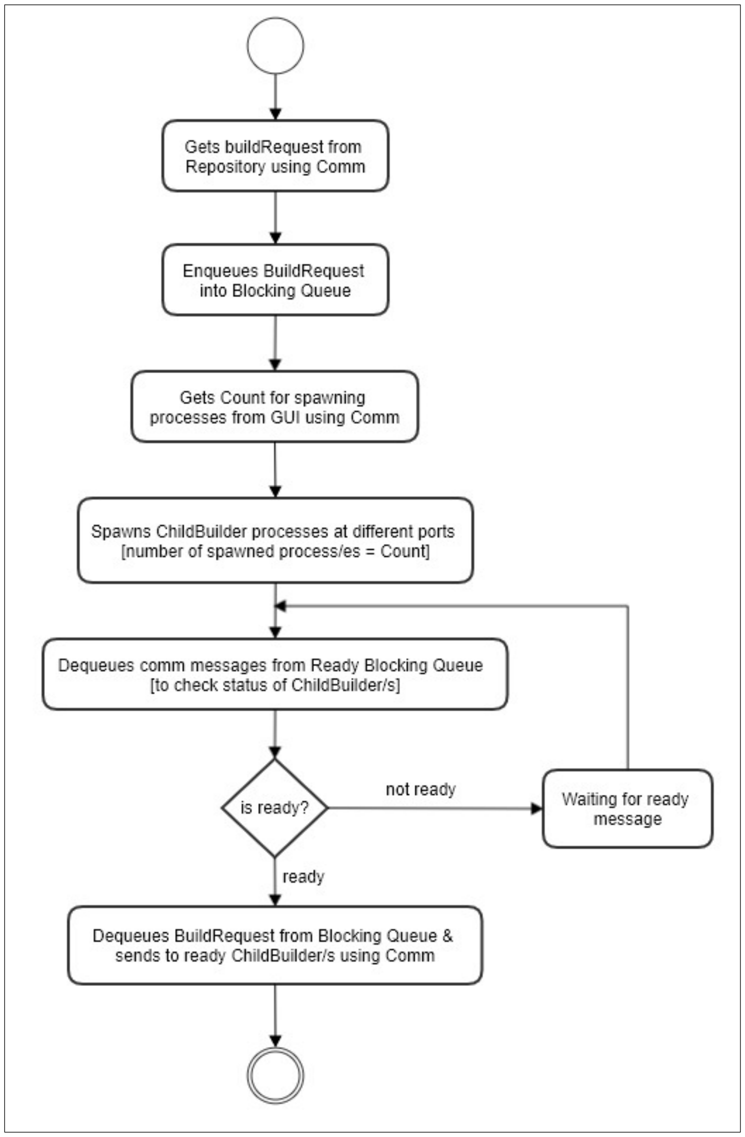

Gets buildRequest from Repository using Comm
Enqueues BuildRequest into Blocking Queue
Gets Count for spawning processes from GUI using Comm
Spawns ChildBuilder processes at different ports [number of spawned process/es = Count]
Dequeues comm messages from Ready Blocking Queue [to check status of ChildBuilder/s]
is ready?
not ready
Waiting for ready message
ready
Dequeues BuildRequest from Blocking Queue & sends to ready ChildBuilder/s using Comm

Figure 5.4. [a] Mother Builder Activity Diagram

Structure –

The structure here defines the classes used by Mother Builder and relationship between them.

1. Mother Builder – This is the main class which handles all the Mother Builder activities defined above. It owns some part of comm class for message passing communication.
2. Comm – Mother Builder class owns some part of this class, so relationship is Aggregation. This class is used to create sender and receiver channel for communication. It is also used to create two blocking queues for performing process pool operation neatly. Mother Builder sends messages to Child Builder and receives from GUI, Repository through this message passing communication class only.

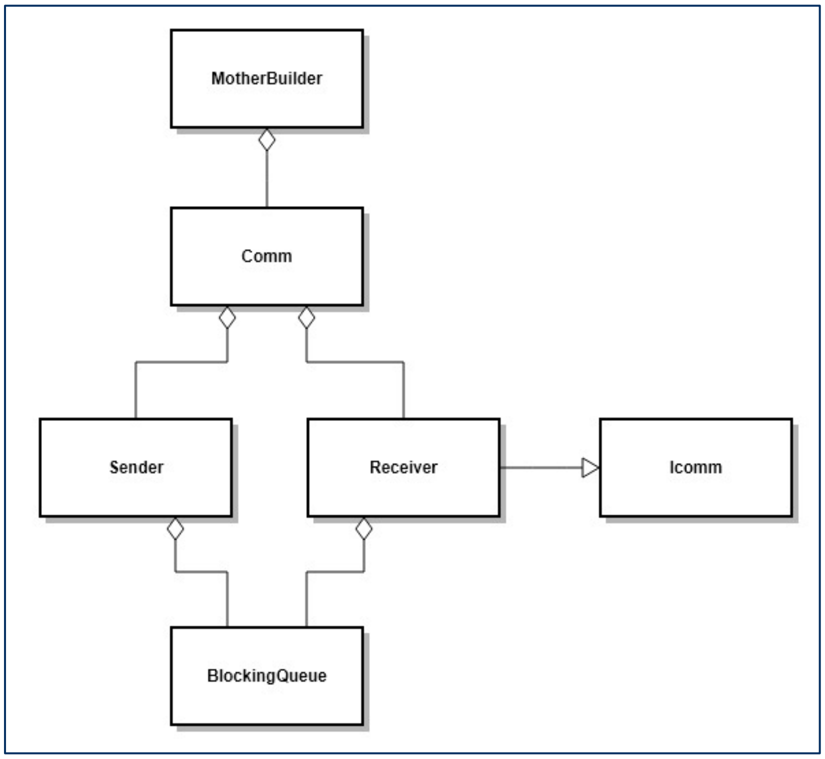


Figure 5.4. [b] Mother Builder Class Diagram

5.5. Child Builder –

Concept –

This is another part of the Build Server. This is the package where build request gets build using .Net Process class and library file [.dll] gets created for the current build request. Child Builder also builds the test request xml file which is then used by test harness for testing.

When multiple child builder processes get created, all child builder process performs the same operation which might give different outputs as build request will be different.

Activities –

1. Child Builder dequeues comm message from receiver blocking queue.

2. If no build request in blocking queue, Child Builder sends ready message to Mother Builder.
3. If yes, Child Builder accepts build request file name sent from Mother Builder.
4. It creates temporary directory for the current build request. Each child builder process creates different temporary directory for different build request.
5. It then copies that build request file from given path to temporary directory path usign upload model [writing the file block by block].
6. Child Builder passes that build request file to the XMLParser.
   XMLParser – It uses the XMLDocument class to load that xml file & to parse it. It then gives the list of dependency file names listed in that build request back to the Child Builder.
7. Child Builder uses the Process class to build the build request i.e. to create library [.dll] file.
8. If build is successful, it calls the Serializer class to generate the test request for generated library file/s.
9. It sends generated test request xml file to Test Harness & build log file to Repository through comm message.
10. If build is failed, it only sends build log output file to Repository.

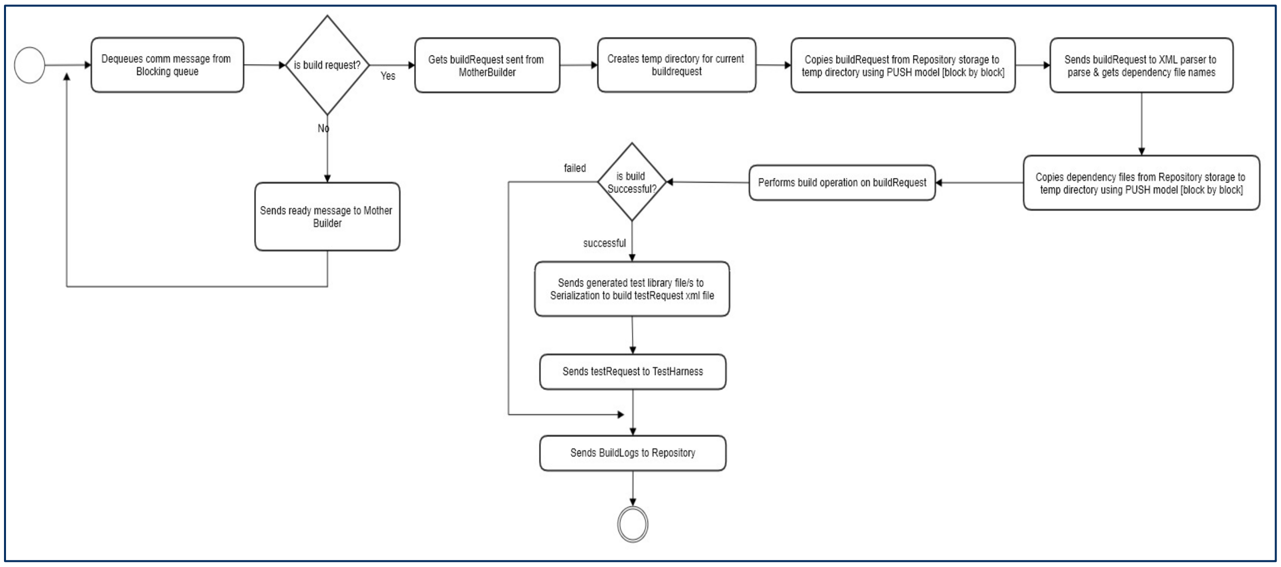

Dequeues comm message from Blocking queue
is build request?
Yes
No
Gets buildRequest sent from MotherBuilder
Creates temp directory for current buildrequest
Copies buildRequest from Repository storage to temp directory using PUSH model [block by block]
Sends buildRequest to XML parser to parse & gets dependency file names
Sends ready message to Mother Builder
Copies dependency files from Repository storage to temp directory using PUSH model [block by block]
Performs build operation on buildRequest
is build Successful?
failed
successful
Sends generated test library file/s to Serialization to build testRequest xml file
Sends testRequest to TestHarness
Sends BuildLogs to Repository

Figure 5.5. [a] Child Builder Activity Diagram

<u>Structure</u> –

The structure here defines the classes used by Child Builder and relationship between them.

1. Child Builder – This is the main class which handles all the Child Builder activities defined above. It owns some part of comm, Serializer and XMLParser class.
2. Comm – Child Builder class owns some part of this class, so relationship is Aggregation. This class is used to create sender and receiver channel for communication. Child Builder sends

messages to Test Harness, Repository and receives from Mother Builder through this message passing communication class only.

3. Serializer - Child Builder class owns some part of this class, so relationship is Aggregation. This class is used by Child Builder to create test request after generating the test library files for given build request.
4. XMLParser – Child Builder class owns some part of this class, so relationship is Aggregation. This class is used by the Child Builder to parse the build request sent from Mother builder in order to get the dependency file names listed in that build request xml file.

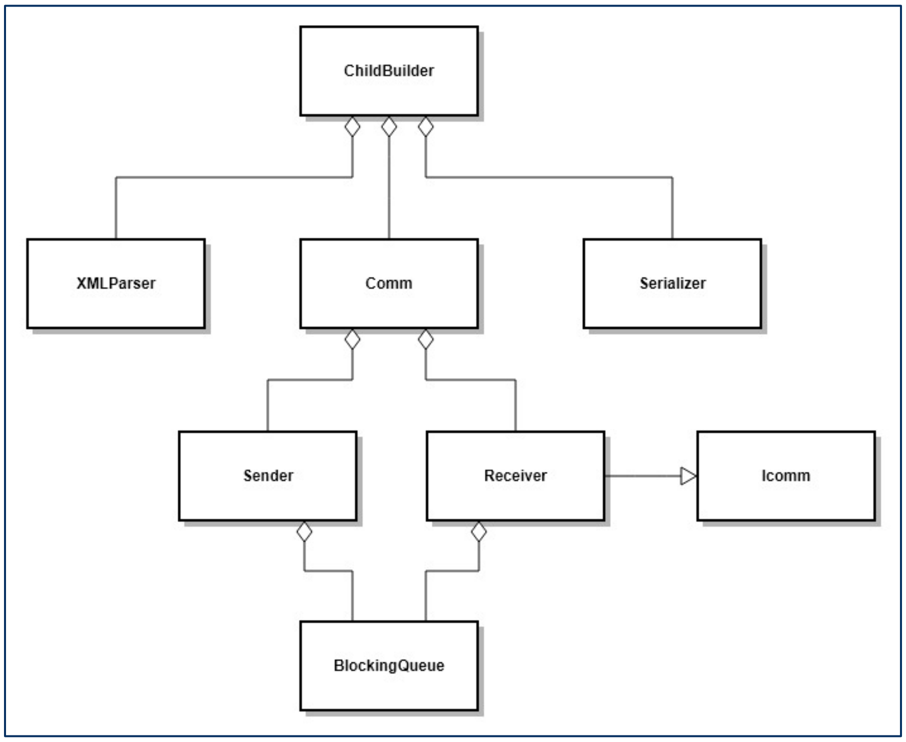


Figure 5.5. [b] Child Builder Class Diagram

5.6. Test Harness –

Concept –

Test Harness uses App Domain concept to load and unload the test library file/s sent from the Child Builder. It then executes that test library file by finding the test driver and ITest interface using reflection and gives the output of the test.

Activities –

1. It dequeus the comm message from receivers' blocking queue and gets the test request xml file name sent from the Child Builder.
2. It calls XMLParser to load this test request file from given path and parse it.
3. It gets the test library file name/s as a result from XMLParser.
4. It uses App Domain to load and unload test library file/s from given path.

5. It then loads test driver and tested code i.e. dependency files for that particular test library using assembly.
6. It then creates the instance of the that assembly uisng reflection ti get all types. And searches for the ITest interface to start running test.
7. It then run tests by getting particular method of that test library using reflection and gives the outpur as Passed or Failed.
8. It also generates test log for each test request and send it to the Repository storage for user.
9. If test is passed, it deletes the temporary directory created at the Child Builder storage path for that particualr build request. This is to avoid cache.

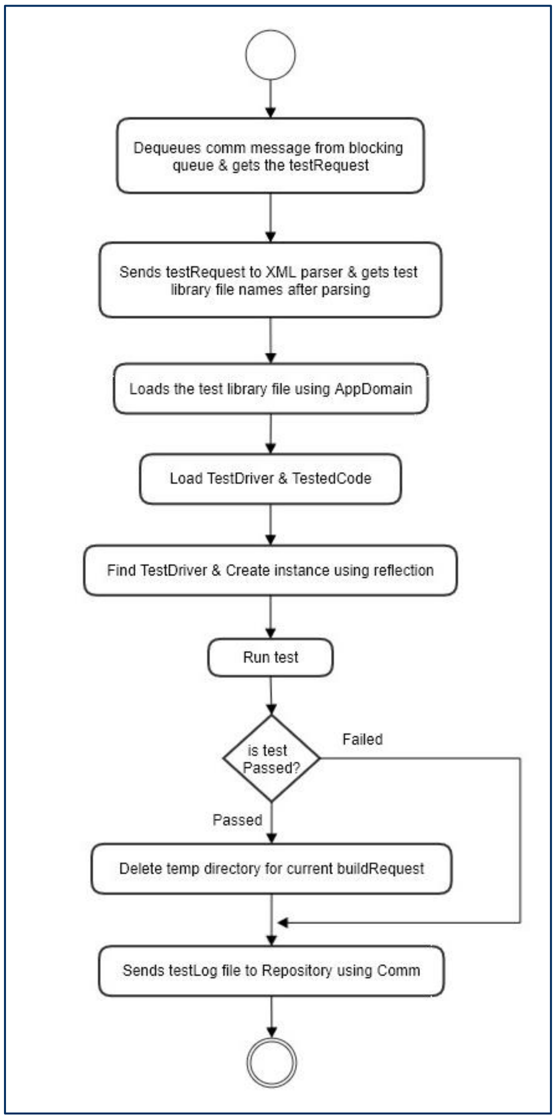

Dequeues comm message from blocking queue & gets the testRequest
Sends testRequest to XML parser & gets test library file names after parsing
Loads the test library file using AppDomain
Load TestDriver & TestedCode
Find TestDriver & Create instance using reflection
Run test
is test Passed?
Failed
Passed
Delete temp directory for current buildRequest
Sends testLog file to Repository using Comm

Figure 5.6. [a] Test Harness Activity Diagram

<u>Structure</u> –

The structure here defines the classes used by Test Harness and relationship between them.

1. Test Harness – This is the main class which handles all the Test Harness activities defined above. It owns some part of comm and XMLParser class.
2. Comm – Test Harness class owns some part of this class, so relationship is Aggregation. This class is used to create sender and receiver channel for communication. Test Harness sends messages to Repository and receives from Child Builder through this message passing communication class only.
3. XMLParser – Test Harness class owns some part of this class, so relationship is Aggregation. This class is used by the Test Harness to parse the test request sent from Child builder in order to get the test library file name/s listed in that test request xml file.

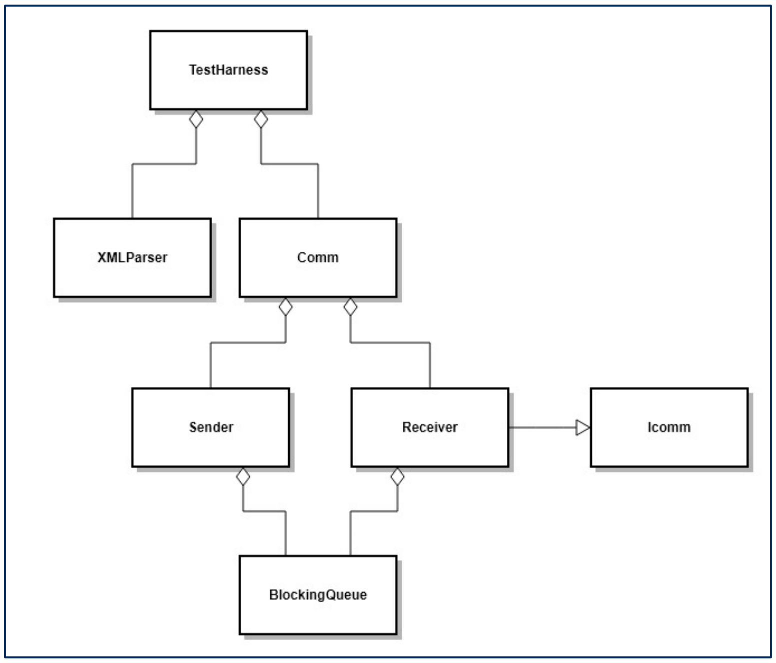


Figure 5.6. [b] Test Harness Class Diagram

5.7. Comm [WCF] –

Concept –

Message-Passing Communication (MPC) establishes a channel between processes to communciate by sending messages. In WCF, each message will consist of a Simple Object Access Protocol (SOAP) wrapper around a serialized instance of a data class that defines the request, the to and from addresses, and any parameters needed to execute the request.

Structure –

The structure here defines the classes used by Comm and relationship between them.

1. Comm – This class is main class which handles all the communication happening between packages. This is a communication channel which has a sender and receiver. Using this comm we create comm channels for each packages with their own sender and receiver.

2. Sender – Comm class owns some part of this class, so relationship is Aggregation. This class is used to perform the senders' operation i.e. sending/posting messages from one point to destination point.
3. Receiver – Comm class owns some part of this class, so relationship is Aggregation. This class is used to perform the receivers' operation i.e. receiving/getting messages from Source point to its own point.
4. IComm – This is the interface used by Receiver class. It defines all the service contracts and data contracts that comm channel uses.
5. Blocking Queue – Comm class owns some part of this class, so relationship is Aggregation. This class is used to create blocking queue for both sender and receiver. And as each package has its' own sender and receiver, they have their own blocking queue too.

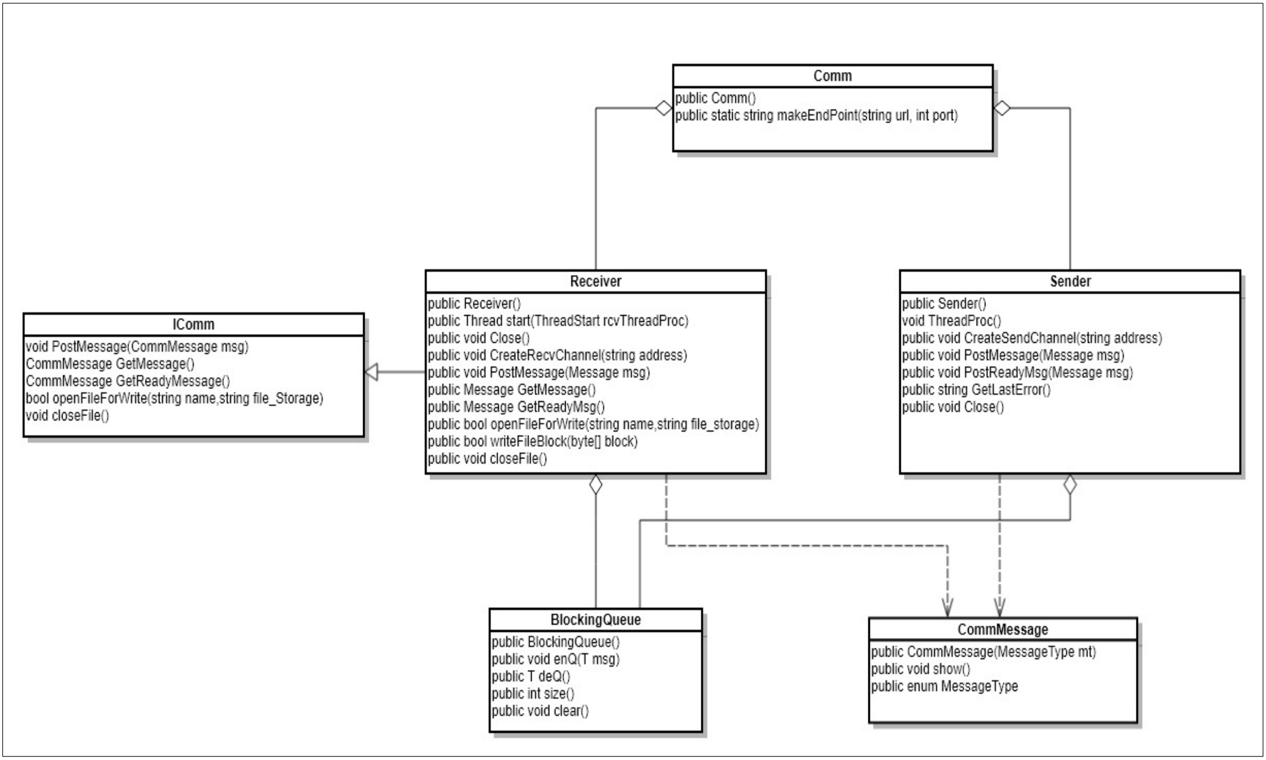

Comm
public Comm()
public static string makeEndPoint(string url, int port)
IComm
void PostMessage(CommMessage msg)
CommMessage GetMessage()
CommMessage GetReadyMessage()
bool openFileForWrite(string name,string file_Storage)
void closeFile()
Receiver
public Receiver()
public Thread start(ThreadStart rcvThreadProc)
public void Close()
public void CreateRecvChannel(string address)
public void PostMessage(Message msg)
public Message GetMessage()
public Message GetReadyMsg()
public bool openFileForWrite(string name,string file_storage)
public bool writeFileBlock(byte[] block)
public void closeFile()
Sender
public Sender()
void ThreadProc()
public void CreateSendChannel(string address)
public void PostMessage(Message msg)
public void PostReadyMsg(Message msg)
public string GetLastError()
public void Close()
BlockingQueue
public BlockingQueue()
public void enQ(T msg)
public T deQ()
public int size()
public void clear()
CommMessage
public CommMessage(MessageType mt)
public void show()
public enum MessageType

Figure 5.7. Comm Class Diagram

5.8. Comm Messages –

Comm Messages are used for communication between two end points. A comm message contains has a specific format –

1. type – type of comm message e.g. FileRequest or BuildRequest
2. to – a destination address so that the sender can connect to that endpoint
3. from – a source address so that receiver can know from where message came
4. command – a command for operation

5. body – actual message a sender wants to send to another endpoint
6. author – author name of this communication channel

If the receiver will eventually reply, then, since there will be multiple senders, the message needs a return address. The message also needs to define the requested operation and provide any parameters needed to carry out the requested action.

The following chart identified all types of messages that can be passed throughout the Build Server, and the implementation details for their contents. These messages facilitate all inter-process communications needed.

Message Type	Sender	Receiver	Command	Content
FileRequest	GUI	Repository	Send files	File names
FileList	Repository	GUI	Returns file list	File names
ProcessCount	GUI	MotherBuilder	Start Process Pool	Process count
BuildRequest	GUI	Repository	Save build request	Build Request String
SendBuildRequest	GUI	Repository	Command	Command
BuildRequest	Repository	MotherBuilder	Send XML file name	Build Request
BuildRequest	MotherBuilder	ChildBuilder	Send XML file name	Build Request
Ready	ChildBuilder	MotherBuilder	Notification	Ready status
FileRequest	ChildBuilder	Repository	Get File	File name
File	Repository	ChildBuilder	Send file	File contents
BuildLog	ChildBuilder	Repository	Send build log	Log file name
TestRequest	ChildBuilder	TestHarness	Send XML file name	Test Request
FileRequest	TestHarness	ChildBuilder	Get File	File name
File	ChildBuilder	TestHarness	Send file	File contents
TestLog	TestHarness	Repository	Send test log	Log file name
Complete	TestHarness	ChildBuilder	Notification	Complete status

Figure 5.8. List of Comm Messages

5.9. User Interface -

User interface helps clients to load files into the repository which they want to get built from build server and then tested by test harness. Having a GUI makes a user-friendly interface.
The designed user interface is –

1. Tab 1 – Main

- Browse Button – Opens a window to choose the directory path and loads .cs files from that directory into the ListBox#1

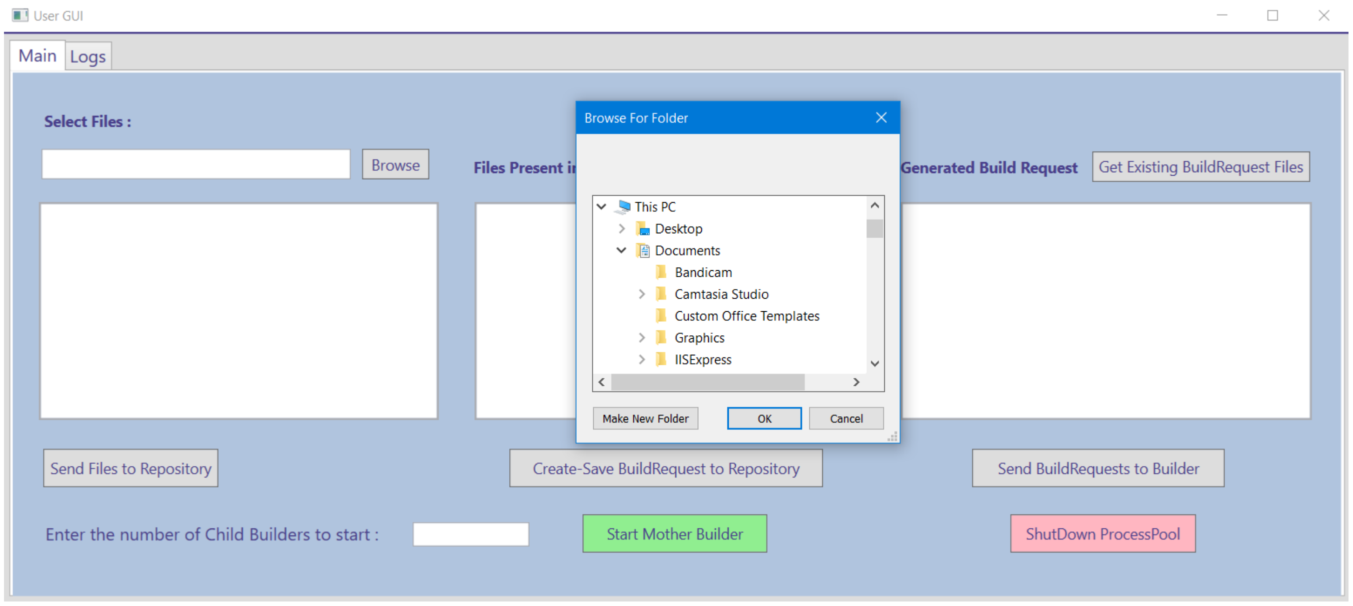


Figure 5.9. [a] Browse Button GUI [Tab 1 – Main]

- ListBox#1 – Shows the .cs file names selected from local storage directory
- Send Files to Repository Button – Sends selected files from ListBox#1 to Repository storage directory
- Get Files in Repository Button – Gets files name of files present in Repository storage directory
- ListBox#2 – Shows files name of files present in the Repository storage directory
- Add test element - Keeps adding the test elements into the single build request structure till user asks to build the build request.
- Create-Save BuildRequest to Repository Button – Creates saves a build request of selected files from Listbox#2 and saves it to Repository storage directory
- Get Existing BuildRequest Files Button – Shows currently existing build request files in repository in the ListBox#3
- ListBox#3 – Shows the files name of generated/existing build request file.
- Send BuildRequests to Builder Button – Gives the command to repository to send the selected build request files from ListBox#3 to Mother Builder

- TextBox – Gives the number of child builder processes to spawn
- Start Mother Builder Button – Sends the number got from TextBox to Mother Builder
- ShutDown ProcessPool Button – Gives command to shut down process pool i.e. all child builders

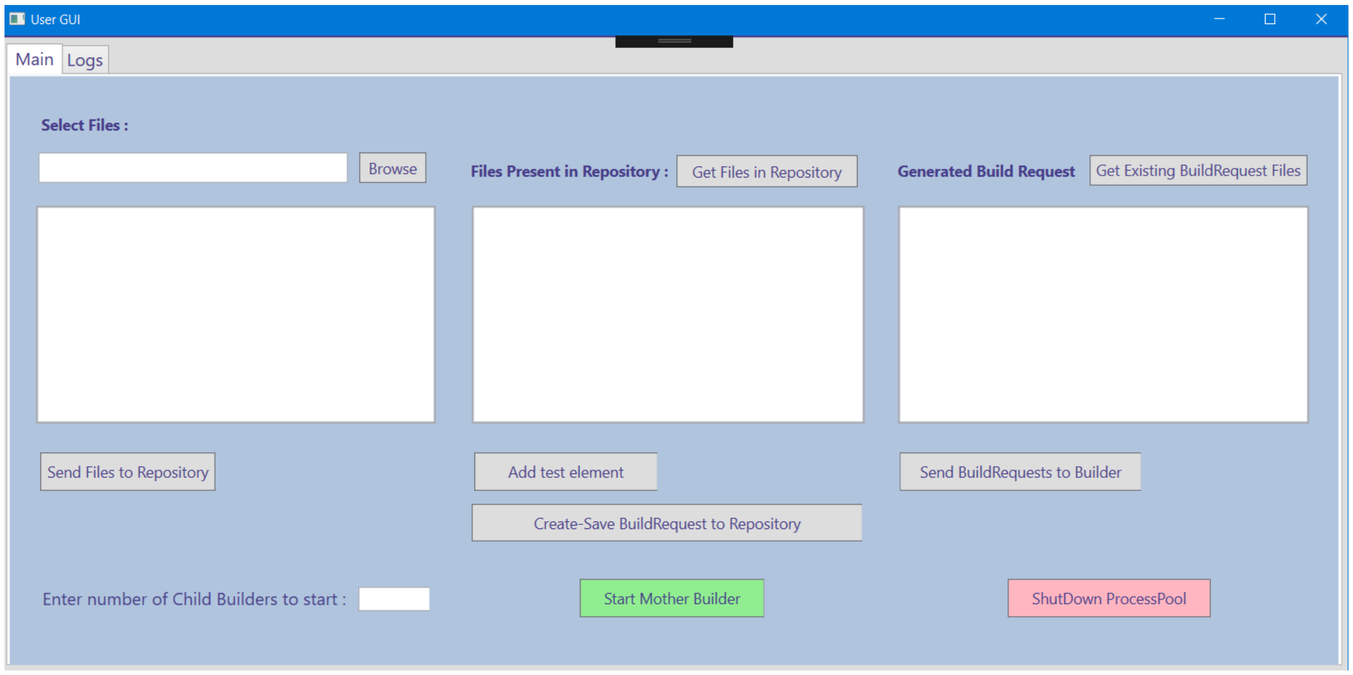
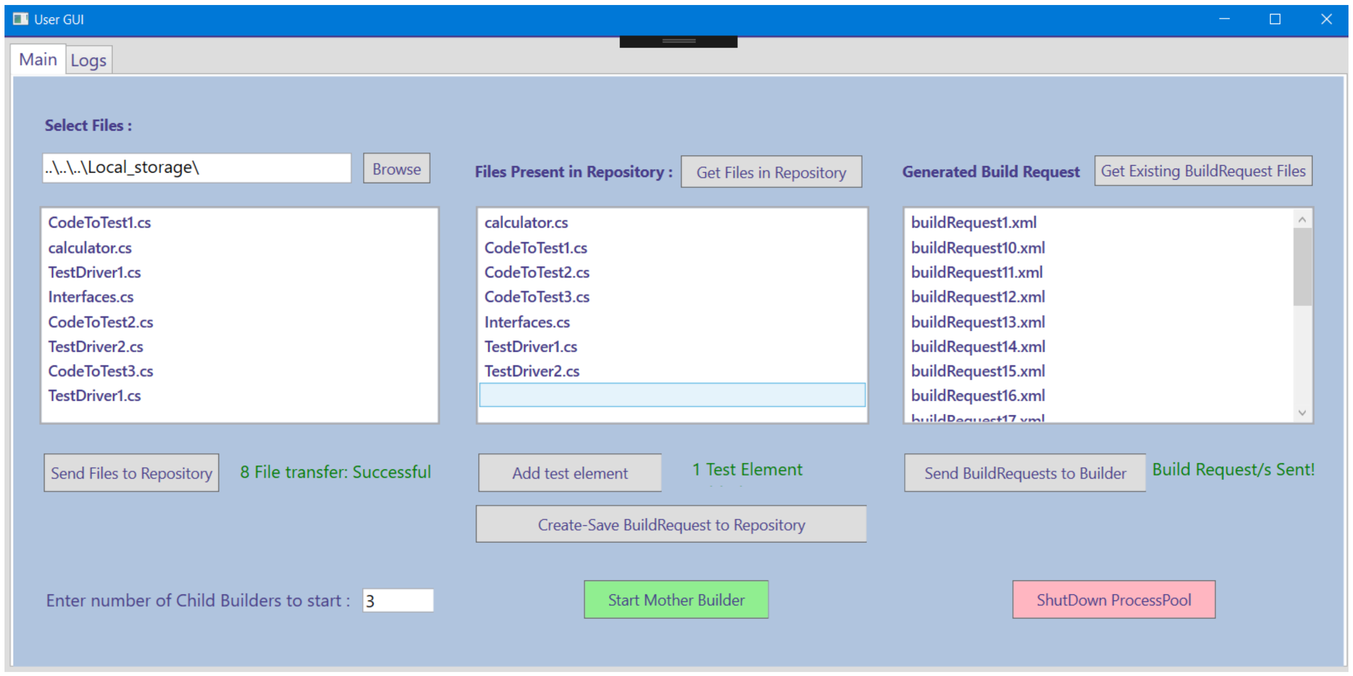

User GUI
Main
Logs
Select Files :
Browse
Files Present in Repository :
Get Files in Repository
Generated Build Request
Get Existing BuildRequest Files
Send Files to Repository
Add test element
Send BuildRequests to Builder
Create-Save BuildRequest to Repository
Enter number of Child Builders to start :
Start Mother Builder
ShutDown ProcessPool

Figure 5.9. [b] Clean GUI [Tab 1 – Main]

After performance –


User GUI
Main
Logs
Select Files :
Browse
Files Present in Repository :
Get Files in Repository
Generated Build Request
Get Existing BuildRequest Files
Send Files to Repository
Add test element
Send BuildRequests to Builder
Create-Save BuildRequest to Repository
Enter number of Child Builders to start :
Start Mother Builder
ShutDown ProcessPool

..\..\..\Local_storage\

CodeToTest1.cs
calculator.cs
TestDriver1.cs
Interfaces.cs
CodeToTest2.cs
TestDriver2.cs
CodeToTest3.cs
TestDriver1.cs

calculator.cs
CodeToTest1.cs
CodeToTest2.cs
CodeToTest3.cs
Interfaces.cs
TestDriver1.cs
TestDriver2.cs

buildRequest1.xml
buildRequest10.xml
buildRequest11.xml
buildRequest12.xml
buildRequest13.xml
buildRequest14.xml
buildRequest15.xml
buildRequest16.xml

8 File transfer: Successful

1 Test Element

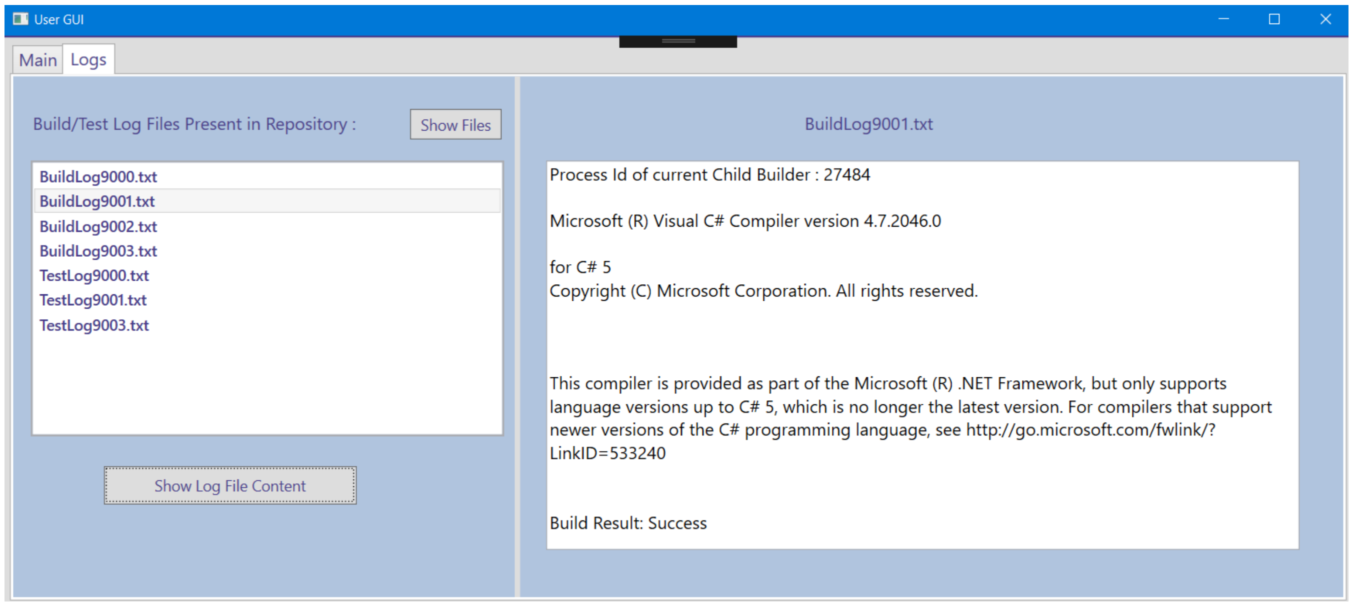
Build Request/s Sent!

3

Figure 5.9. [c] GUI [Tab 1 – Main]

2. <u>Tab 2 – [Logs]</u>

- Show Files Button – Shows all build/test log files generated by child builder/test harness that are present in the repository
- ListBox#4 – Shows all .txt file names present in the repository
- Show Log File Content Button – Shows the content of selected log file from listbox#4 into Textbox
- Textbox [Name="log"] – Shows the content of selected log file from listbox#4 into Textbox

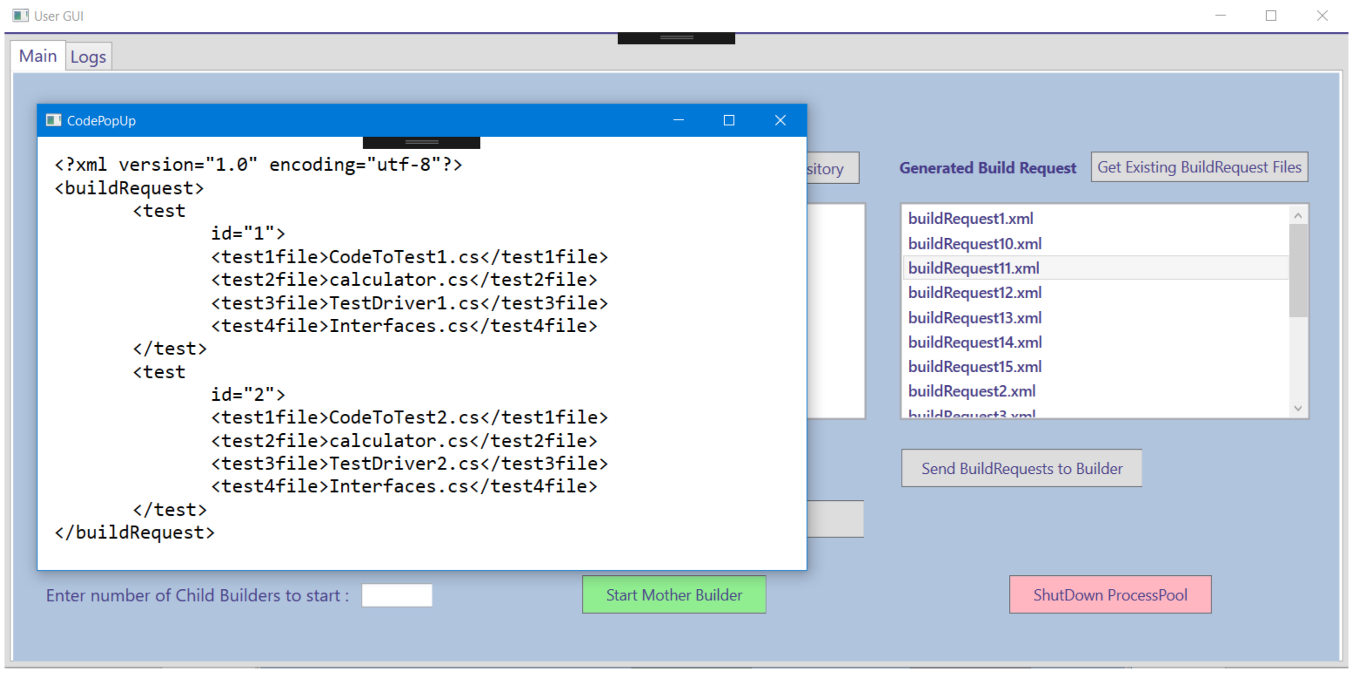


```
Process Id of current Child Builder : 27484

Microsoft (R) Visual C# Compiler version 4.7.2046.0

for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.



This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports
language versions up to C# 5, which is no longer the latest version. For compilers that support
newer versions of the C# programming language, see http://go.microsoft.com/fwlink/?
LinkID=533240


Build Result: Success
```

Figure 5.9. [d] GUI [Tab 2 – Logs]

3. CodePopUp GUI –

   It shows the content of the build request file into CodePopUp GUI on double click.


User GUI
Main
Logs
CodePopUp
sitory
Generated Build Request
Get Existing BuildRequest Files
Send BuildRequests to Builder
Enter number of Child Builders to start :
Start Mother Builder
ShutDown ProcessPool

```
<?xml version="1.0" encoding="utf-8"?>
<buildRequest>
       <test
              id="1">
              <test1file>CodeToTest1.cs</test1file>
              <test2file>calculator.cs</test2file>
              <test3file>TestDriver1.cs</test3file>
              <test4file>Interfaces.cs</test4file>
       </test>
       <test
              id="2">
              <test1file>CodeToTest2.cs</test1file>
              <test2file>calculator.cs</test2file>
              <test3file>TestDriver2.cs</test3file>
              <test4file>Interfaces.cs</test4file>
       </test>
</buildRequest>
```

```
buildRequest1.xml
buildRequest10.xml
buildRequest11.xml
buildRequest12.xml
buildRequest13.xml
buildRequest14.xml
buildRequest15.xml
buildRequest2.xml
buildRequest3.xml
```

Figure 5.9. [e] CodePopUp GUI

6. EVOLUTION

The development of remote build server accomplished in total 4 stages. The very first stage of this development was writing an Operation Concept Document [Similar to what is written in this document except for more focus on concept than on design].

The concept I represented in my first OCD differs from what I have designed in my final project with necessary modifications and addition of new concepts. The main reason behind this difference is critical issues I faced when I started designing the project. Following are the major changes made in the design to make it work for all most all critical conditions –

1. Windows Communication Foundation –

   The simplest way to communicate between packages is giving the reference of that package to the needed package and handing the methods by creating class objects. This might go in wrong way if both packages are dependent on each other creating circular dependency.
   To solve this issue, I modified my design by implementing WCF. Here, the communication is done by Comm messages (one specified structured message) from one endpoint to another endpoint.

2. Process Pool –

   The main functionality of build server is to build & test the build requests sent by the user. As per explained in the Section 3 [Users and uses], there might be multiple users sending multiple build requests at the same time to build server. My design in OCD 1 didn't support the handling of multiple build requests feature.

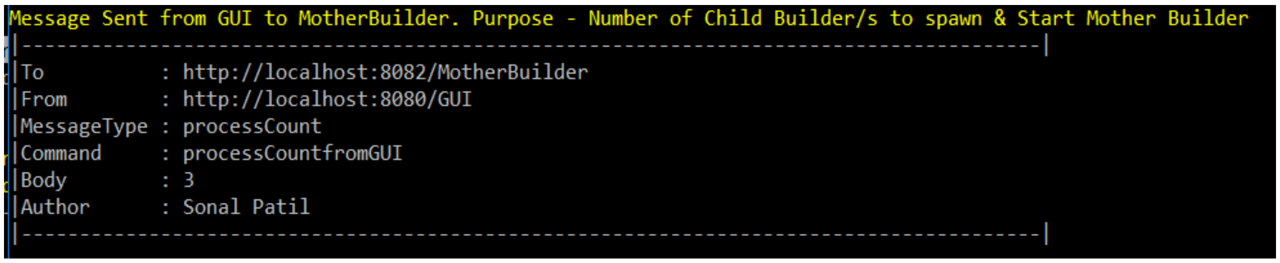
   To handle multiple build requests, I modified my previous design by implementing Process pool concept. In this concept, when users send multiple build requests, Mother builder counts the number and spawns that much number of child builder processes. Each child builder handles one build request at a time.

7. CRITICAL ISSUES

7.1. Message structure used in communication –

Issue – We are using Windows Communication Foundation to make all packages communicate with each other by sharing messages. The issue here is how to define a single structure that works for all messages sent in the federation and not to write the structure every time.

Solution – Defining a specific structure for all messages in the Data contract with its data members solves this issue. A message that contains To and From addresses, enum Message type, Command string, a string body to hold message information needed for operations.



```
Message Sent from GUI to MotherBuilder. Purpose - Number of Child Builder/s to spawn & Start Mother Builder
|--------------------------------------------------------------------------------------|
|To          : http://localhost:8082/MotherBuilder
|From        : http://localhost:8080/GUI
|MessageType : processCount
|Command     : processCountfromGUI
|Body        : 3
|Author      : Sonal Patil
|--------------------------------------------------------------------------------------|
```

7.2. Building and testing multiple language source code (C#/C++) –

Issue – For this project, we have built test libraries for C# source code and implemented testing operation on generated dll files. The issue is what if user gives C++/Java source code to build.

Solution – Adding a tool chain packages which will check every time it gets a build request to decide the building operation will solve this issue. Every time, when a test library gets loaded and the tool chain will generate and exception if it's not C# test library and by trapping this exception we can then call C++ test harness to complete the testing.

7.3. Complex Builds –

Issue – Performance of the system will get reduced if the builds are more complex and taking too much time. Software systems usually have many developers working on million lines of code using various components like open source libraries, framework etc. In this situation building process becomes complex and sometimes cause the error.

Solution – To avoid the complexity because of long builds, we must make sure that the communication between all packages occurs in the efficient manner without any deadlock conditions. Use of Windows communication foundation with blocking queue for both sender & receiver solves this issue.

7.4. Handling multiple build requests –

Issue – As per discussed in section 3 of this document, our project supports multiple users. There is a strong possibility that build server will get multiple build requests to proceed.

Solution – Use of process pool will solve this issue. It means when build server gets multiple build request in its blocking queue, it will spawn that many number of child builder processes to build them concurrently.

7.5. Versioning -

Issue – When we'll do the versioning for the build log files so that the user will get to know the latest version of successful/failed build log. In real world, while doing the builds on million lines of code, more than hundred build logs will get created. This might create a problem while storing all those build logs. It will take a lot of memory to hold those build logs.

Solution – The solution for this issue is, we can develop a package in our system which will keep the track of build test request, if the same test request comes to the build server due to previous failed build or due to some new changes, this new package will check whether the test request is same to the previous one, and if the result gets true, it'll delete all previous versions except for the current one and the last successful build log version.
The reason of storing the last successful version is, build logs gets created and stored in repository without knowing build result. So, the current build might get failed and if we delete all the previous versions' of build logs then it will cause us to lose last successful build log version.

7.6. Build Request passing using Blocking Queue –

Issue – Blocking queue solves the problem of concurrent access. But sometimes build server might get a lot of build requests more than expected. Build server couldn't process those requests as queue is fully occupied. Unaware of the status of the queue, users keep sending the build requests. This might raise problem where repository is sending the files and build server is unable to catch it because Queue is full.
Solution - One solution to this issue is sending a notification to user to wait till the queue gets vacant space to accept the next requests. Another solution is to provide dynamic blocking queue for the system with the heavy build requests.

8. CONCLUSION

The Build Server finds its utility in users like Software Developers, Managers, and Quality Assurance Team etc. working on distributed development projects. It is a useful tool to automate, stabilize and version control the software development process. In this Operational Concept Document of our build server, we have described the key idea and design of our remote build server using designed user interface, activity, class and package diagrams and a few critical issues occurred during designing and resolved in order to successfully implement the concept.

9. REFERENCES

1. http://ecs.syr.edu/faculty/fawcett/handouts/webpages/BlogOCD.htm
2. http://edn.embarcadero.com/article/31863
3. http://deviq.com/build-server/
4. https://martinfowler.com/articles/continuousIntegration.html#AutomateTheBuild
5. https://www.joelonsoftware.com/2001/01/27/daily-builds-are-your-friend/
6. http://www.doublecloud.org/2013/08/parsing-xml-in-c-a-quick-working-sample/