SMA-681 Software Modelling and Analysis

Instructor- Jim Fawcett

Project# 4

Remote Build Server-OCD

Submitted by- Nishant Agrawal

SUID- 595031520

Submitted on- 6th Dec. 2017

Index

Figures

1. Executive Summary

The purpose of the project is to implement a remote build server that is going to accept build request from a mock repository, start child build processes and send each one a build request when they are ready. They parse the xml file to get the names of the test drivers and the tested files from the repository. If the build succeeds, they send a test request and libraries to the Test Harness for execution, and the build logs to the repository. Harness parses the test request and executes the tests, and sends the logs to the repository. There is also an interactive user interface which will be used by the client. All the interactions between different servers is done using WCF communication channel. The main objectives can be put in bullet points as under:

- Supports build request access by Pool Processes from mother Builder process, source file access from Repository.
- Provides a Repository server that supports client browsing to find files to build, builds an XML build request string and sends that and the cited files to the Build Server.
- Provides a Process Pool component that creates a specified number of processes on command.
- Uses message-passing communication to send and receive messages and files between all the components of the federation.
- Each Pool Process attempts to build each library. On success, sends a test request to the Test Harness, and the build logs to the repository.
- The Test Harness attempts to load and execute each test library it receives, and sends the result logs to the Repository.
- Includes a GUI client as a separate process, implemented with WPF that gets file lists from the Repository, and packages the selected files into a xml request string.

The intended users of this project are Developer, Project Manager, Software architect, customers and finally the QA and the maintenance team. In the further sections, their uses have been elaborated.

Below are some of the critical issues associated with this project, their solutions have been expatiated ahead in this document.

- Sending test requests directly to the build server or through repository.
- Load on the test harness server when running a very high number of test requests simultaneously.
- Risk of modifying files while they are being used by multiple users.
- Keeping track of building events when there are a lot of test requests being sent from different users/actors.
- Handling the cases when test libraries have multiple references to other libraries.
- Redundancy caused by running the same test again and again.
- Ease of use, logging and security.

2. Introduction

This document explains the concepts for building a Remote Build Server. In today's tech-savy world, with the advent and requirement of better and robust AI's and IOT's, which can easily be controlled with a very small device using a simple software/application, softwares have pretty much become part and parcel of everything. And in the future, looking at the ongoing efforts towards restoring concsciousness after death and integrating bots with humans, that day is not far when softwares will be integrated with our body too.
Such softwares can be from a range of ten lines to thousands of pages of complex code. The smaller ones can be easily developed by a single developer with minimalistic user interface, documentation or architecture. On the other hand, in a large scale software system, there can be multiple simultaneous users(actors/softwares) that access the core functionality through some kind of network. In order to build such softwares, one of the the best practices is continuous integration, where newly generated code is thoroughly tested and once all the tests are cleared, that code becomes part of the software baseline. For this there are multiple services that need to executed on a federation of servers. A build server is one such server where the building service is executed.

2.1. Objective and Key idea

The key idea of the project is to use WCF between process pools to implement the process of continuous integration. In order to implement this, a federation of servers running services of a client, repository, builders and test-harness is necessary. Hence, the objective is to create a remote build server with complete functionality and other mocks to support the entire process. A user interface is implemented for the client to interact with other services.

2.2. Obligations

The obligations include implementing a remote build server, part of the federation of servers where a user can create test requests, build and compile these tests into libraries and finally execute the libraries. The test results, build logs and test logs need to be maintained in the repository and all the communication and file transfer between different servers must be done via WCF. A interactive client GUI is also implemented with the project.

2.3. Organizing Principles

- Use DLLs to execute the tests by compiling the test files into such libraries.
- Use WCF to send messages and files between the remote servers which have different services running on them.
- Integrate a GUI client using WPF to ease the process of building and processing test requests.
- Use a process pool of builder processes to process each build request and a mother builder to handle these processes.

3. Use Cases

3.1. Developer

A Developer who has to write code and make builds numerous times in a day and run tests on a regular basis would have the most advantage using this tool and would be able to save a lot of time using this automated tool. A developer would want to:

- Update and modify the code files- An organized repository server would help maintain this fluidity.
- Write new small code snippets, build and run quick tests in order to ensure software rigidity- Using the test harness, adding new code files and running tests on them is going to be quick and easy using this tool.
- Will get logs and reports in case of failures- This tool would ensure that building and testing of multiple build and test requests is not blocked by failures by processing the other requests.

3.2. Project Manager

The manager who has to lead the projects can use this tool and its parts in the following ways:

- Monitor and keep track of the progress of the project.- The build and test logs can be accessed by the manager, thereby giving an absolute idea of the project advancement. This will in turn help to draw better and realistic deadlines, also to know when to motivate and push the employees.
- Send timely notifications to superiors and customers about the schedules and estimated time frame.

3.3. QA

A QA can have maximum benefits by using this tool by getting timely test logs and thereby be able to put more time towards building better test-cases and better functional docs and BRDs. A QA can use this tool to:

- Build executable or library with multiple test drivers in a single test request.
- Access previously build test requests and libraries logs stored in the repository.

3.4. Maintenance Team

A Maintenance team can benefit by using this tool by:

- The maintenance team would need to ensure that everything is up and running. In case of failures or complaints from clients, they need to be able to run tests on all the versions of the software.
- Running tests in order to resolve issues real fast whether it may be on customers' or their end. They can also identify the problems way faster by checking the old log files.

3.5. Software Architect

A software architect would need to:

- Review the project progress in a fast and reliable way.
- Check all the implementations and whether those implementations go hand in hand with the project requirements.

3.6. Customers

If this tool is made open to the customers, their developers can resolve their issues faster without needing to wait for the product company to find the problem and respond back.

- For trivial issues, they would want to run tests on their end and save time on the communication back and forth.
- Check out what new implementations are being done and get those upgraded in their software version.

3.7. Future work extension

In the future, proper file versioning can be done in order to implement the functionality of a cache storage at builder and test harness server. Also functionality for the user to choose to run the same test again or get the results of the previous tests can also be implemented. This will reduce the time taken to execute the builds and tests and thereby saving resources and reducing costs. This will also reduce the server load.

4. Application Activities

The key task of the application is to implement a remote build server as part of a federation of servers. When projects are huge, relying on continuous integration is highly dependable. Whenever there is code modification, it is imperative to test it against the previous test standards to ensure that nothing is breaking. This build server allows a user to send test requests that are built into Dll files and tested at the test harness. All the main activities are classified as under:

4.1. GUI client browses the files in Repository and creates build request

The GUI client can send a message to the repository to get all the source files stored. Then the client can select and add the drivers and test files to an XML string and send that string as a message argument to the repository over the communication channel, where the build request will be created and stored for future use. Each build request is an XML file with multiple test drivers i.e tests to run, basically a piece of code and the source files, other pieces of code, that need to be tested. 'Get files' button asks the repository to get all the source file names. Build request can be built using the 'Add driver' and 'Build Request' buttons.

4.2. GUI client browses xml test requests and commands repository to start processing

The GUI enables the client to get the names of all the xml requests in the repository by sending a message to the repository. Later, a command to process the selected xml requests can be sent to the repository, which in turn sends the requests to the mother builder. 'Get Xml' button displays all the xml requests saved in the repository and 'Start Building' button asks the repository to send selected files for further process.

4.3. GUI client shuts down build process pool

The GUI also enables the client to shut down all the started builder processes by sending a message to the motherbuilder, which in turn sends 'quit' messages to the processes and shuts them down.

All the activities of the mock client are displayed below in the activity diagram.

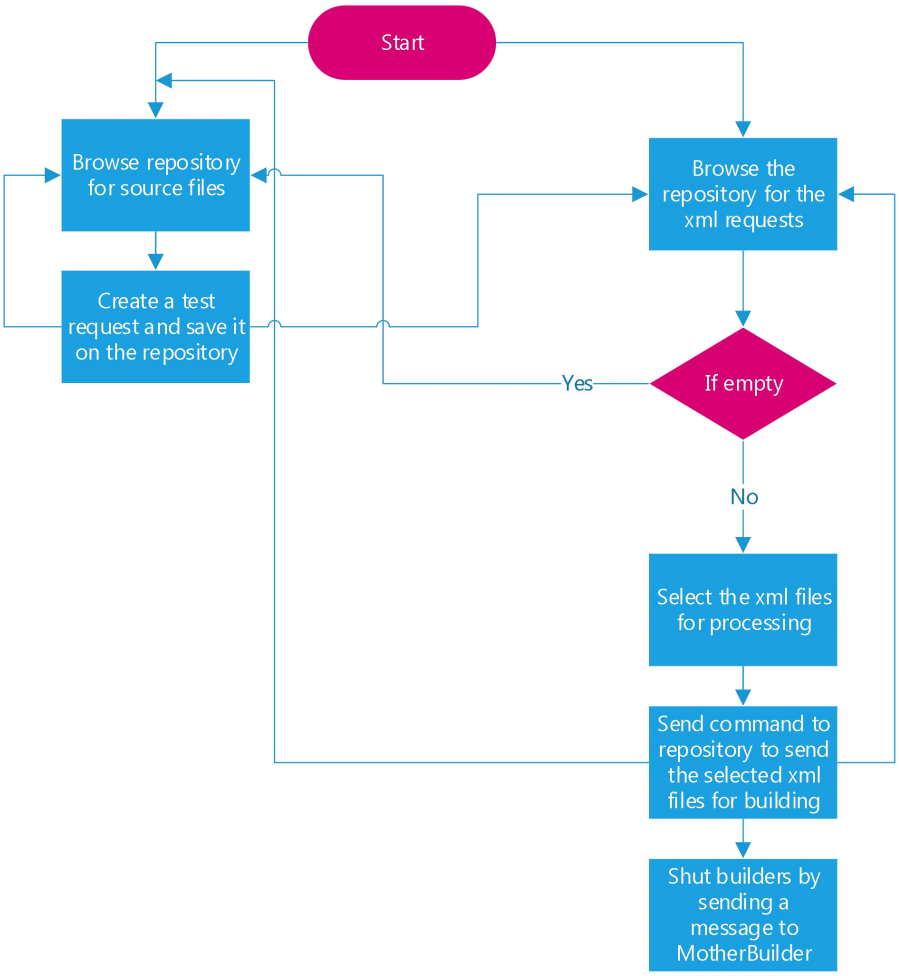


Fig.1 GUI client activity diagram

4.4. Repository gets its files for GUI client and stores the build requests

A repository is a storage base which sits on a family of directories. For this project we are using only a single directory in accordance with the small scale. Once it gets the command from the client, it gets all the stored files and sends their names to the GUI client as message arguments. After it receives a xml build string from client, it creates and saves the build request. The associated files to the build server are sent when client requests for it.

4.5. Repository gets xml files for GUI client and sends them to mother builder

The repo also sends all the xml file names to the client as message arguments, where client can select the xml files to process. As the command from client to process comes, it sends the respective files to the mother builder for building.

4.6. Repository gets source files and posts to builder processes

Repository gets the messages from the builders to send the source files back to them. It gets the names of the required files from the message argument and then posts those files to the respective builder addresses.

The activity diagram below mentions step by step activities of the mock repository.

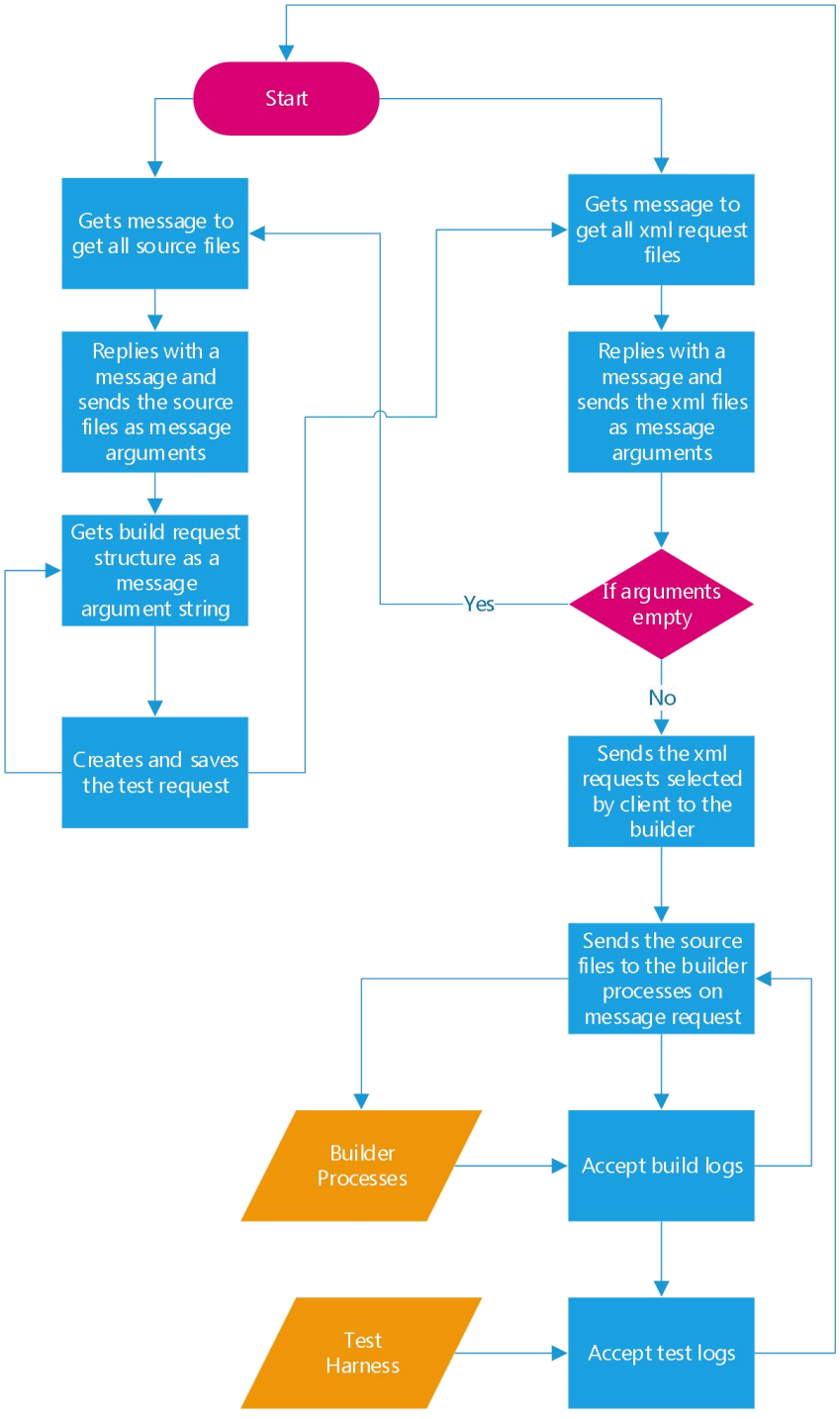


Fig. 2 RepoMock activity diagram

4.7. Mother Builder accepts and adds the requests to a blocking queue

The mother builder accepts all the build requests coming from the repository to a blocking queue, to be dequeued later to be sent to the 'ready' builders. It is responsible for managing the builder process pool.

4.8. Mother Builder starts the builder processes and sends them build requests

As soon as the client clicks the Start building button, the selected xml files are sent to the mother builder from the repository and it starts the builder process pool. It enqueues all the 'ready' builders on a blocking queue, which are dequeued one by one and a build request is posted to each of them. The process is repeated until all the build requests are processed.

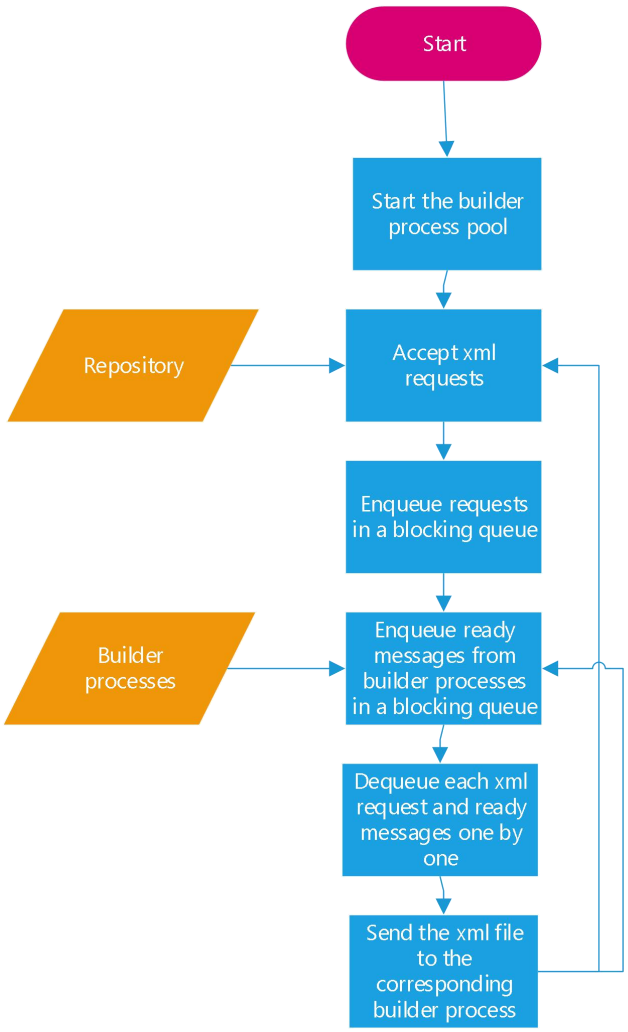


Fig. 3 MotherBuilder activity diagram

4.9. Builder process sends ready message to the mother builder

As soon as a builder process is started it sends a 'ready' message to the mother builder and then waits for an xml request. Once it processes the build request, it again sends the 'ready' message and waits for another request to be posted in its local storage. A receive thread ensures this works without any complication.

4.10. Builder process parses the build request and check its cache for the source files

Each builder process parses a build request and then checks its cache for the source files needed to build the request. If all the files are already there, it starts building the request. Whichever files are not there are sent as message arguments to the repository asking for them.

4.11. Builder process asks the repository for the source files

The builder sends a 'sendSourceFiles' message to the repository asking for the source files.
Once the repository gets the message, it checks the message arguments and get those files and then posts them to the builder local storage.

4.12. Builder builds the build request into dll files

It starts building the build request once all the source files are in its local storage and compiles each test into a dll library file.

4.13. Builder shows build succes and fails, and sends build logs to repo

After the build process is finished, the builder displays the output, warnings and errors and logs the same into a log file. This log file is then posted to the repository for storage so that it can be uesd for future reference.

4.14. Builder process sends the successful builds to the test harness

If the build is successful, i.e all the tests in the build request are successfully built, the builder creates a test request and adds all the dll file details to it. Later it posts the test request and the respective library files to the test harness.

Activities of the build server are ordered in the correct sequence below on the next page.

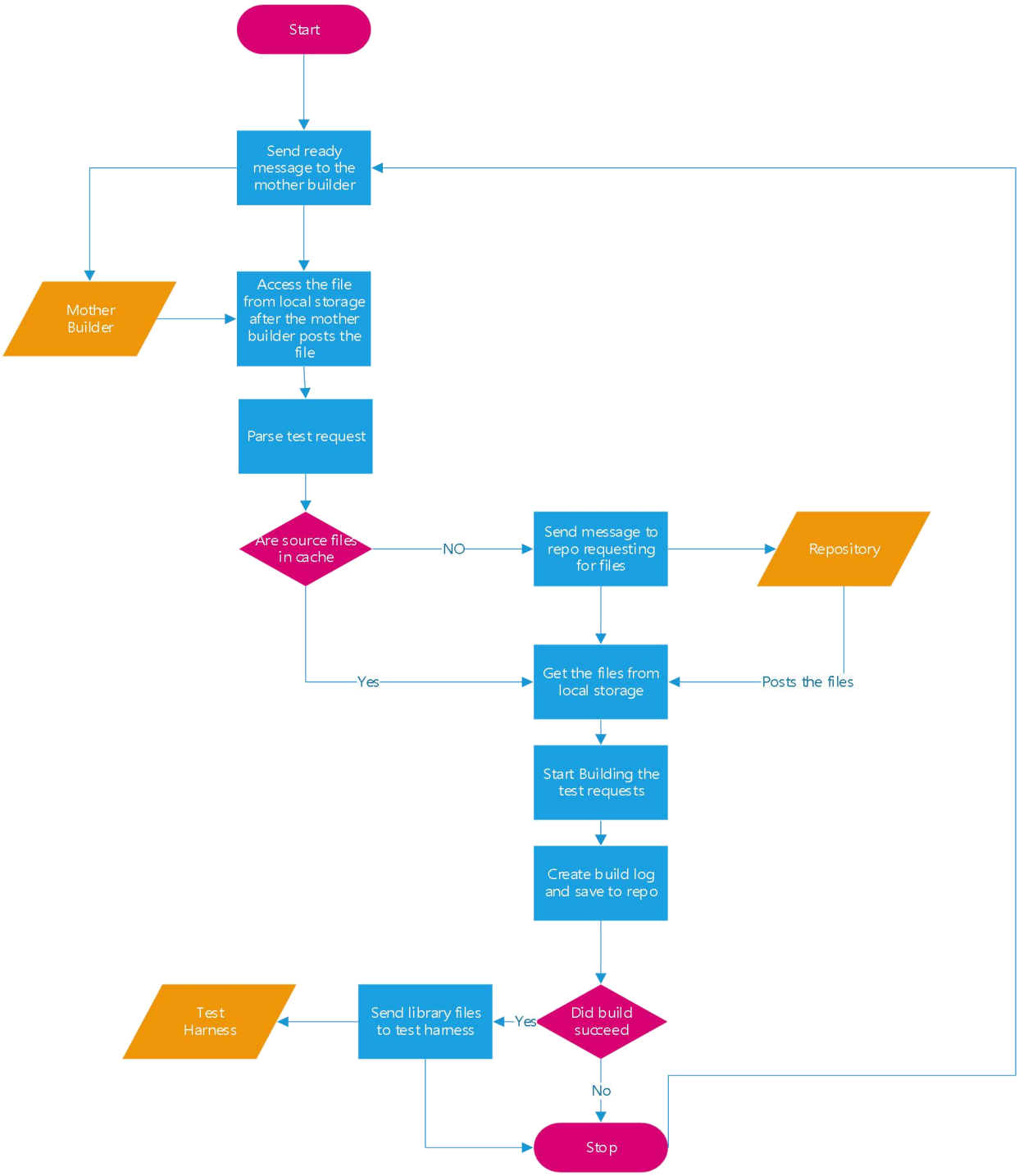


Fig. 4 Builder process activity diagram

4.15. Test harness accepts and enqueues the test request

The test harness accepts the test requests and the libraries from all the builder processes and enqueues them into a blocking queue. Each of the test request is then dequeued one by one for processing.

4.16. Test harness parses the test request and executes the tests

The test harness parses the test requests one by one and then executes the respective dll files. Once it is done with each, it dequeues the next test request and repeats the process until the builders stop sending the test request, thereby emptying the blocking queue.

4.17. Test harness shows the test results and logs the same to repo

Once the tests are executed, the test harness displays the results and also logs the same into a log file. This log file is then posted to the repository for storage.

Activities of the test harness are displayed in the correct flow in the activity diagram below.

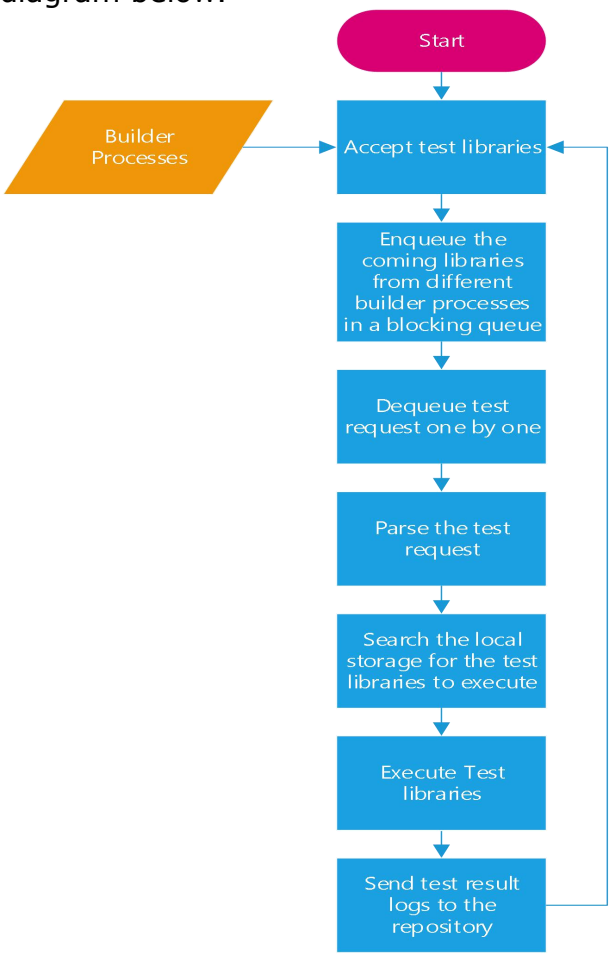


Fig. 5 Test harness activity diagram

5. Partitions

The package partition based on the tasks have been described in this section. A package diagram below displays the interactions of these packages in a clear and fluid manner.
The package partition details are explained after package and class diagram.

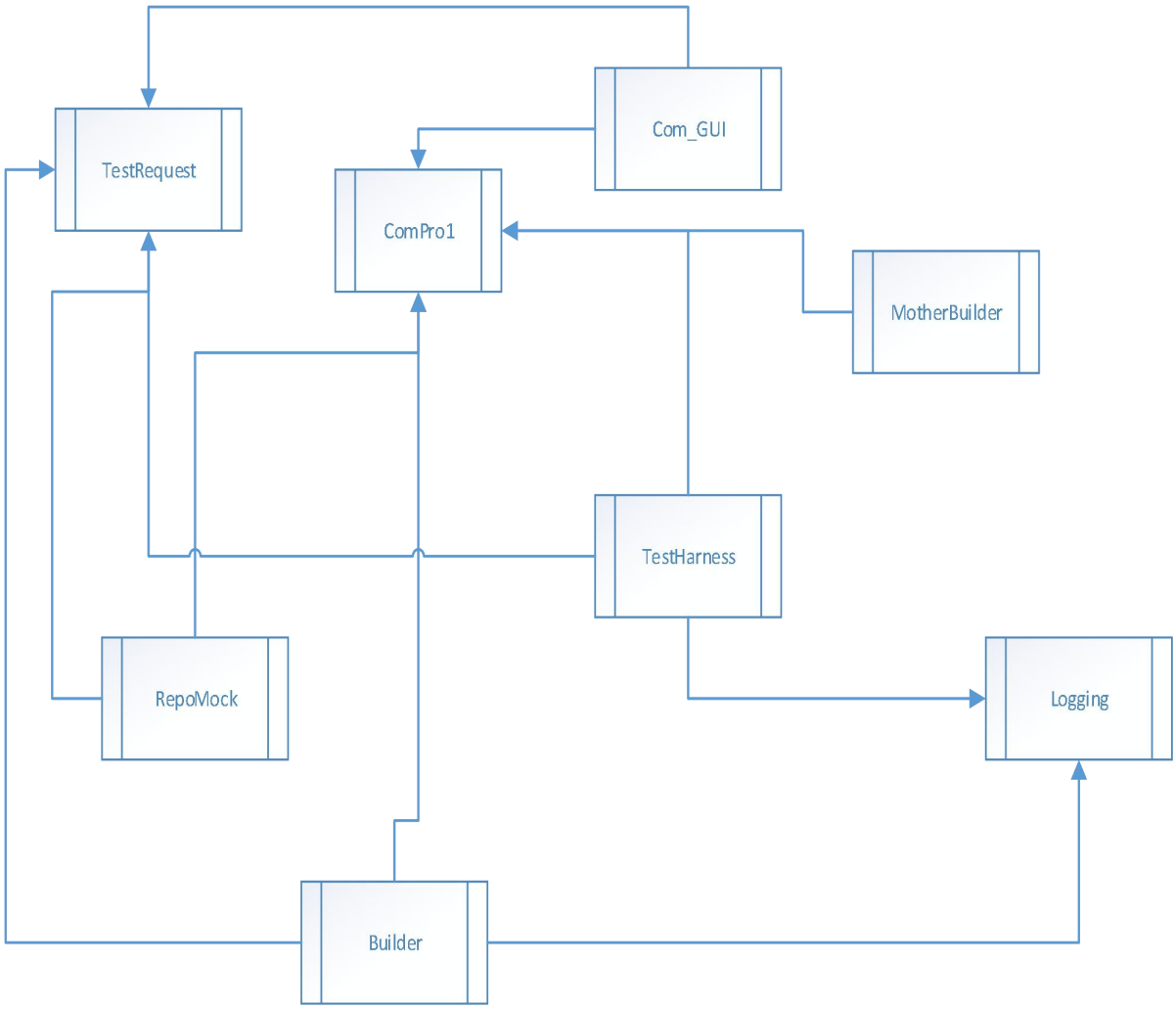


Fig. 6 Package diagram

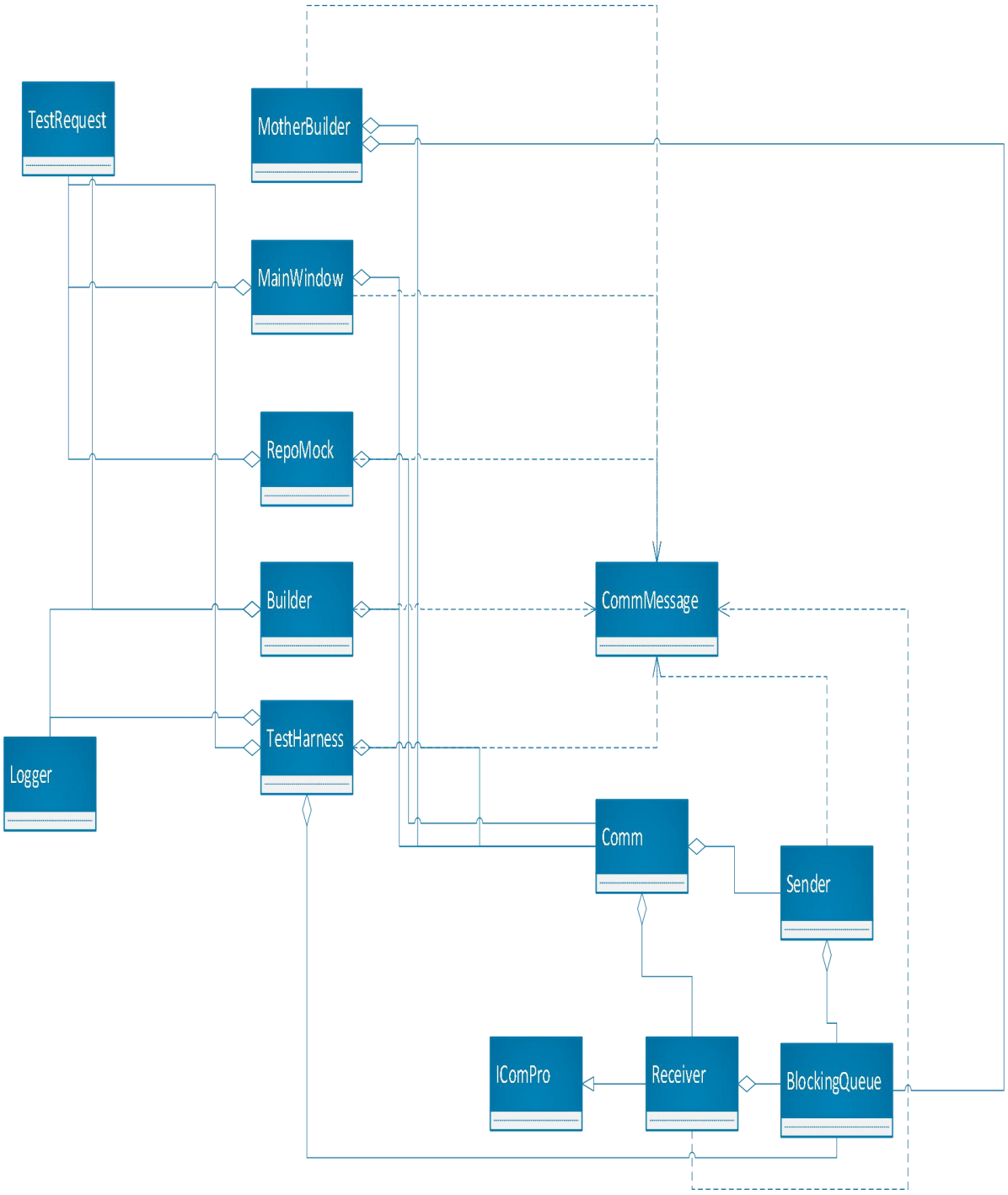


Fig. 7 Class diagram

5.1. Com_GUI

This package implements the GUI, built using WPF, and all its functionalities. This package uses ComPro1 and TestRequest packages to create its private comm and create build requests respectively. It has a receive thread which gets the message from the comm, displays the message and then transfers the control back to the main thread by using the message dispatcher. It also has the testExec function where the main control flow of the application enters through. So, the execution of the program starts here. This function helps to show that the requirements for the project have been met by doing a complete run of the application, using other helper functions. The getfiles() function sends the message to the repository asking to get all the file names, which are displayed in two list boxes each for test drivers and tested files. There is another similar getRequestfiles() function that requests the repo for the xml requests, which are displayed in another list box in 2nd tab. Then there are xmlString() and buildrequestfunc() that help create the build requests by sending a xml string to the repo. The function sendXmlToBuilder() gets the selected xml files and requests repository to process them further. Rest are the button functions required to do some action when the buttons on the GUI are clicked. These buttons execute below sequence of actions

- Button_adddriver - adds selected driver and tested files to the xml string
- Button_buildrequest - creates the build request
- Button_getfiles- gets source files from repo
- Button_getXml- gets xml files from repo
- Button_startbuilder- starts processing the selected xml files for building
- Button_shutbuilder- sends the message to mother-builder to shut process pool

5.2. TestRequest

This package helps create build requests and test requests by loading and saving it to XML file. In this project, we need to parse the test requests at two places, once at the builder processes and a second time at the test harness. It also provides functions to load and parse both kind of xml requests.

5.3. RepoMock

This package is simulates the basic functionalities of a repository. It allows accessing and saving files to repository. This package uses ComPro1 to create a private comm and TestRequest package to create build requests. It also has a receive thread to handle all the incoming messages. It also has functions to get files, send messages to client, motherbuilder and builder processes for getting files at client, sending xml files and sending source files respectively.

5.4. MotherBuilder

This Package again uses the ComPro1 package for communication with different servers. Here, the process pool is started by getting the client inputted no. of processes to start. A function to close all the builders by sending a quit message is also implemented here. It also has 2 blocking queues, one for build requests and other for 'ready' messages from the builder processes, which are both dequeued one by one and each process is sent a xml request to deal with.

5.5. Logging

This helper module helps to create log files both at builders and test harness and then save them to the repository. We need to create two types of logs: first, the build logs which will be created after the test requests are built into libraries and second, the test logs which will be created after the libraries are executed. Therefore this package is going to interact with builder as well as the test harness.

5.6. Builder

In this package, we implement all the core functionalities of a builder. It will interact with TestRequest to parse build requests and create test requests, and with ComPro1 to use the WCF communication. Finally it will use the Logger package to create build log files. It implements functions to send ready messages, build source files into dll, check cache for source files otherwise ask repo, send build logs, build test reqeusts and send them and libraries to the test harness.

5.7. TestHarness

This package is where we will execute the test libraries and then use the Logger to create test logs, and ComPro1 to receive test requests and send the logs to the repository. The TestRequest package helps to parse the test requests. It will also display the tests results at the command prompt and send the test logs to be saved in the repository.

5.8. ComPro1

This package implements the WCF comm, sender and receiver classes , and also the blocking queue needed by the comm. This allows to send and receive messages and files over a communication channel.

6. Critical Issues

6.1. Test Request Direction

One of the ways to design this tool can be to send the test request directly to the build server which has the primary job of building the test request instead of the repository. But, since the end goal is to have a dedicated server for each type of task, which includes storing files, request building and running the test libraries by remote access, it seems smarter to save everything to the repository. Also it is safer to do all the storage including files and requests on a single server. This not only provides better security and consistency, but also helps in maintaining test request history. Therefore, here the program is designed in a way that the request will be sent and stored in the repository first and then only will these requests be sent to the build server by the command from the client.

6.2. Load on test harness server

Server-load is one of the biggest issues that may need to be handled in case too many tests are running on the test harness server at the same time. The chances of getting too many requests to run tests is quite viable as this tool has an array of uses for different users. The best way to resolve this issue is to scale out the task on multiple servers/machines. Maximizing the hardware ensures operation no matter the volume of the upgrade demand.

6.3. Repository config management issue

The biggest risk while using this tool is the chance of file modification which can happen in a number of ways.
If the same file is being updated by multiple users, there can be huge inconsistency and therefore, it can result in a catastrophic failure.
This can be resolved by storing the files with proper meta-data, for example author's name and date.
Another issue can be created if the older files get deleted as the newer versions arrive. This can be a cause of concern when a customer is still using the older version of the software. Because of the unavailability of the older version, it might not be possible to run tests to find issues in case of a software failure. Proper versioning of the files not only resolves this issue but also the former one to some extent. It means that once a file is versioned, it becomes immutable and cannot be modified any longer by any user.

6.4. Keeping track of build events

In large software systems, there are huge teams dedicated to work on the project which means that on a regular basis there may be hundreds of build requests in a single day. This can result to inconsistencies if there are multiple builds with same name. This can also be resolved by storing the build requests in a proper way by using more intuitive meta-data like user's name, date and time grouped together.

6.5. References of other libraries in a test library

In a large project there are bound to be multiple references to other libraries in a test file. In such a scenario, while running test, the test harness is not going to know the names of those libraries. Therefore, it can result in many inconsistencies. Such issues can create binding errors that can be resolved by using delegate, which gets assigned a target method at run-time and acts as a delegate for the caller.

6.6. Redundancy while sending same test requests on same files

Another issue can be raised when the same test requests, which have been run before, need to be run again. This can result in serious redundancy and create unnecessary load on servers and thereby increasing the costs. This can be prevented by building cache storage in both build and test harness servers, where recently run test requests and results can be stored. Since cache storage is faster than sending remote requests, this will reduce the effort and also save a lot of time.

6.7. Logger

Generation of log files is important to be created in a way that it is clear and concise and user can see what's going on in one look.
Solution: The logger can be implemented in a way to put in a lot of metadata clarifying everything and should be written in an organized manner.

6.8. Security

Not everyone should be able to access all the files, builds and tests. This can be implemented by giving different privileges to different officials, e.g. a manager should not be able to access files or tests that are only meant for developers.

6.9. Easy to use

The remote builder should not be cumbersome to use. A clear and easy to use GUI should provide users fast and effective methods to test their code against the baseline.

7. References

- http://te.ieeeusa.org/2010/Jun/backscatter.asp
- https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/presentations/ProjectCenterUseCases.pdf
- Project helper code by Prof. Jim Fawcett

8. Appendix

8.1. Message flow

This figure displays the message flow between the different services involved in the entire application. Below is the description of all the CommMessages.

GUI_client:
1. getAllfiles- request to RepoMock to get all source files in repository.
2. getXmlFiles- request to RepoMock to get all the xml files
3. sendSelecXml- request to RepoMock to send the selected xml files to the MotherBuilder. The file names are sent as message arguments.
4. createRequest- send request to repo to create a build request. The selected file names are sent as an xml string in the message argument
5. startBuilders- send request to the MotherBuilder to start the builder process pools.
6. killBuilders - send request to MotherBuilder to kill the process pool

RepoMock:
1. setAllFiles- to client with all the source file names as message arguments
2. setXmlFiles- to client with all the xml file names as message arguments
3. sourceFilesSent- message to builder processes notifying that all the required source files have been posted in their local storage
4. allxmlsent- message to MotherBuilder notifying that all the xml files selected by the client have been posted to MotherBuilder's local storage.

MotherBuilder:
1. xmlFileSent- message to builder process notifying it that the xml file request has been posted to the builder process's local storage
2. Quit- message to builder processes to quit themselves

Builder:
1. ready- message to the MotherBuilder to let it know that builder is free to start processing another build request
2. sendingBuildLog- to RepoMock notifying that a build log has been posted in repository
3. sendsourcefiles- request to RepoMock asking for the source files that are sent in the message arguments.
4. executeTests- to TestHarness notifying that the test request and the libraries have been posted to it.

TestHarness:

1. sendingTestLog- notification sent to the repository letting it know that that test log has been posted there.

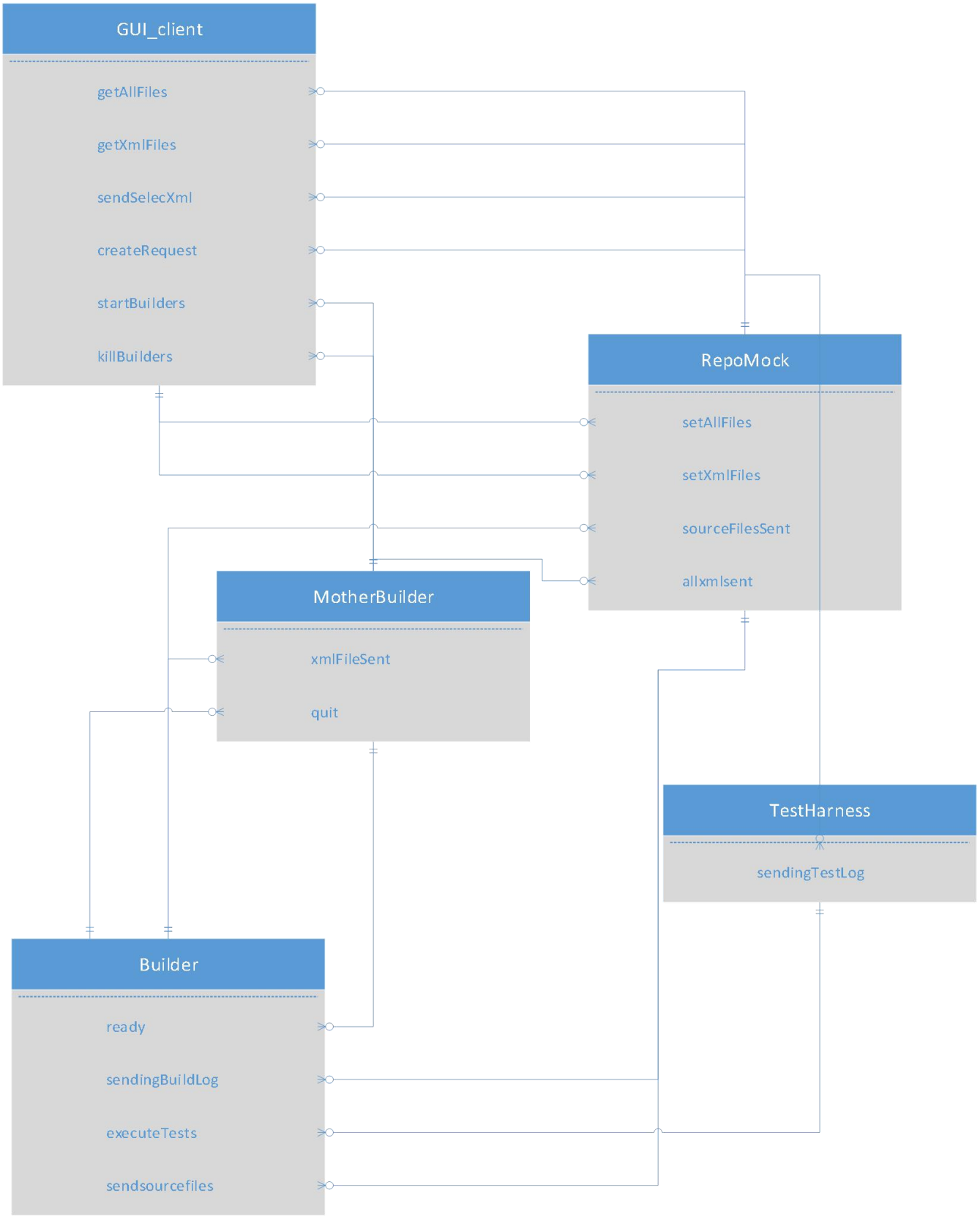


Fig. 7 Message flow diagram

8.2. GUI elements- user interface

All the elements of the GUI and their functionality has been discussed in the below points.

1)Firstly, there are 2 tabs in the GUI window:

1. **Create Request**- to help select source files and create and save a build request on the repository. Has two list boxes, each for Test Drivers and Tested files, and 3 buttons- Get Files, Add Driver and Build request.
2. **Start Building**- to help select the xml files and then ask the repository to send the files to the mother builder for further process. Has a text box for getting the user input for the number of build processes to start and a list box for displaying all the xml request files in the repository. 3 buttons are there to help with the functionality- Get Xmlfiles, Start Building and Shut Pool Processes.

2) **TextBox**- processnumber text-box to get the user input for the number of pool processes to start.

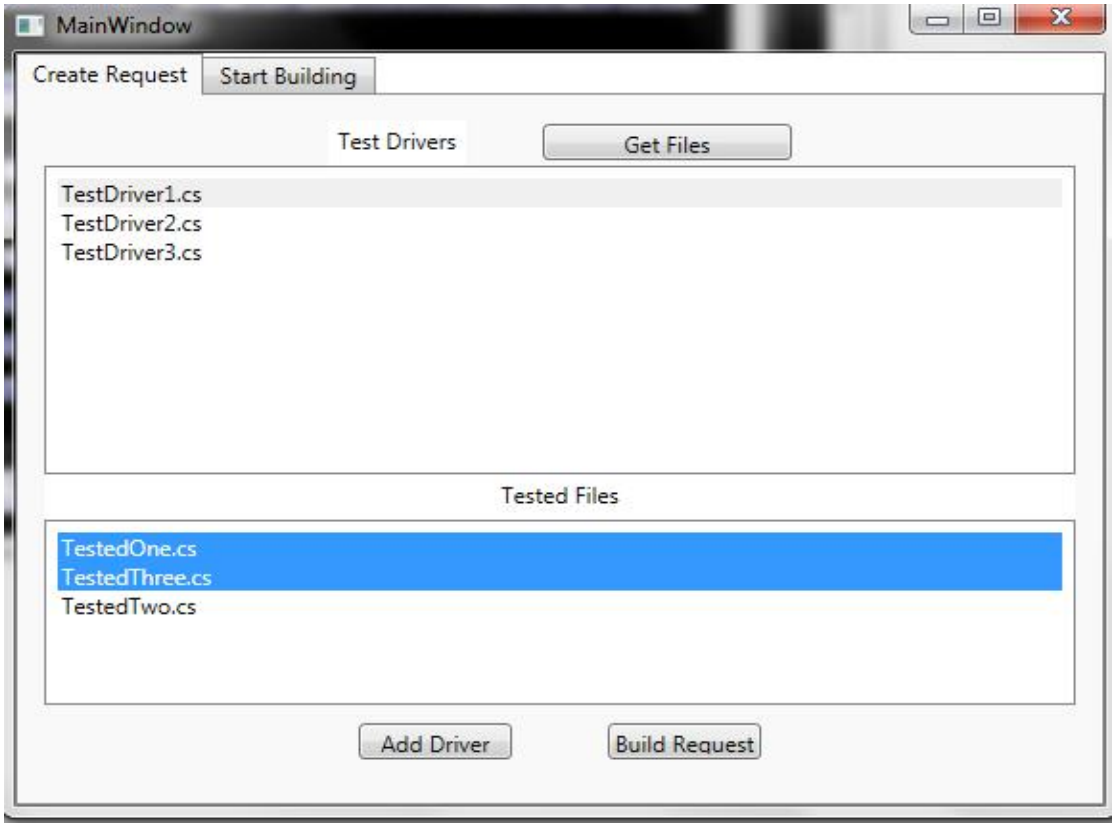
3) **ListBoxes**

1. driverListBox- to display the list of the test drivers, single selection is allowed in this box.
2. testedFilesListBox- to display the list of all the tested files in the repository, multiple selection is allowed, so that user can add multiple files to a test driver.
3. xmlListBox- to display all the xml request files stored in the repository, multiple selection is allowed so that user can select all the requests needed to be processed.

4) **Buttons**

1. Get Files- populates the list boxes for the drivers and tested files with their names.
2. Add Driver- After a driver and some tested files have been selected, this button adds them to a list.
3. Build Request- Once at least one test driver is added, this button creates and saves the build request on the repository with the respective selection.
4. Get Xmlfiles- populates the list box for the xml file requests after getting it from the repository
5. Start Building- once at least one selection is made from the list of xml file names, this button starts the builder processes and starts the processing of the selected xml files.
6. Shut Pool Processes- this button shuts down all the builder processes.

The snapshots of the GUI are displayed under.



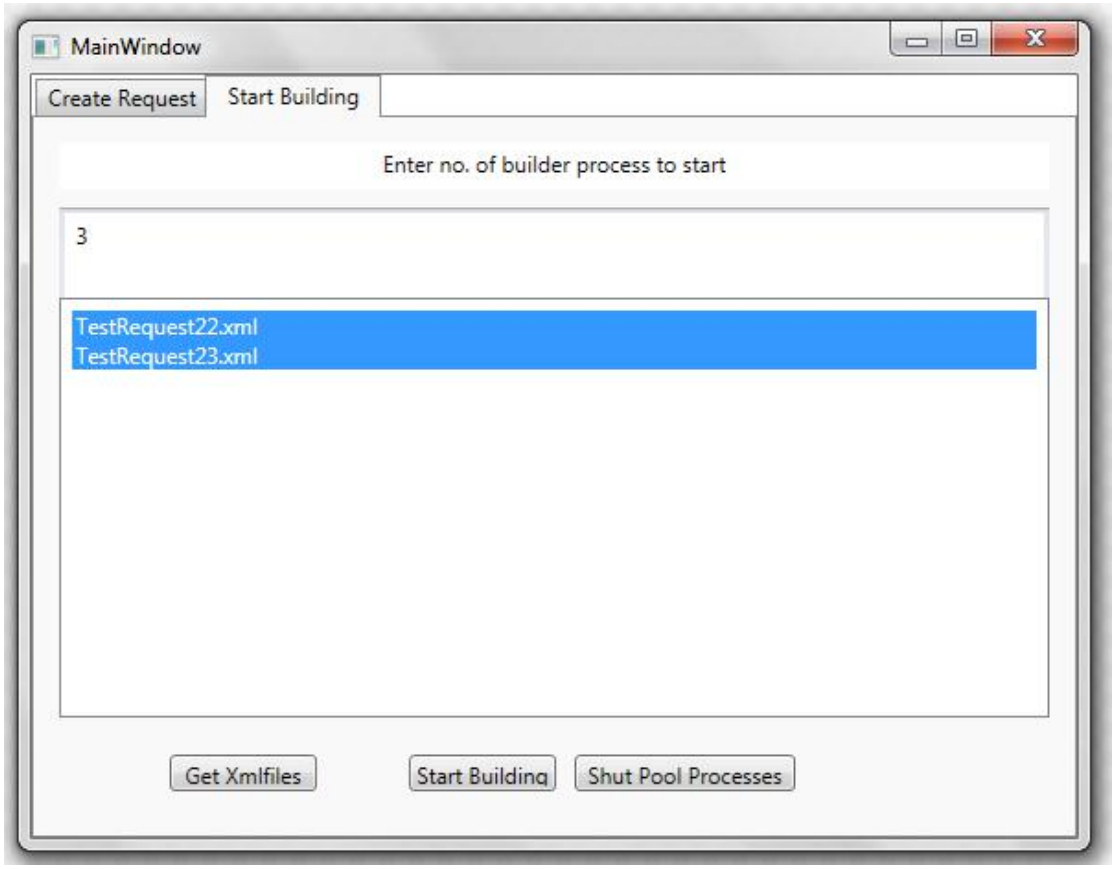


Fig. 8 GUI snapshots