Build Server
Operational Concept Document

CSE681-PROJECT #1

By Quanfeng Du (SUID: 373746579)

Instructor: Jim Fawcett

Date: 1th December 2017

Contexts

Figures

1. Executive Summary

In the procedure of software development, there always be several teams to work on deferent part on the whole project, then the different part will be integrated at once. It causes the consumption of time, if the module broken each other's code, it will be detected only when after the integration. In early days, this problem is detected by using continuous integration, this builds on the principle of "always keep it working".

The main purpose of this project is to build an automatic build server to build the specific source code. This includes the test driver and the necessary tests. The build server will build test driver and the tests into a single dll, it also generates the log file which indicates the building status. In this project, also include the mock repository and mock test harness.

The main user of this project is me, graders and instructor. In such a way, the project is designed easily evaluated and all requirements are well demonstrated. The idea build server will be designed with remote access and multithreading facilities can be used by developer, Quality assurance and manager.

Here list several certain significant issues that need to be addressed while developing a build server:

- One of the major problem is that the how to make sure the performance of the build server, there could be hundreds of build requests. The time to complete the build process will increase dramatically with the increase of the number of build requests. This can be addressed by open multiple build process by build mother, those build child can process the build concurrently.
- It is very important to provide a simple interface to make sure it is easy to use.
- how to design the communication process between different parts.

Introduction:

Software Collaboration Federation is used to partition code into relatively small parts and test each part before inserting them to the baseline when implement the big system. When new part added, or error happens, we need to re-run test sequence for these parts. It is reasonable to make this process as automate as possible since there might be a great number of packages. For each part, it will need a separate test library. Build Server is designed to help us solve this issue.

Uses:

Build Server is designed to meet the requirement for five types of people. Include me, graders, instructor, Software developer and QA.

Application Activities:

When the project running, it provides a Graphical User Interface, from the GUI, user can get the source files in the repository, and select the source file to build a build request in the form of XML. Then, user choose the request files and send to repository. The repository will parse the xml file and find the source files associated with the request file, send them to mother build. Mother build starts to parse the request file and create the temporary directories for sources files. User now should input the number of process they want use to build the requests. After getting the number of process, mother build will open child process and send a request and the associated source files to a child. After a child done the builds process, the child will send the ready message back the build mother. Mother builder will assign new build request to the child until all build request are built.

After the build process, the build mother will command the build children to send the log files to repository and dll files to the test harness. Test harness will test the dll and generate the test result log, which will be sent back to repository.

Partition:

Build Server will be implemented in C# using .NET framework on visual studio 2017. The system will be divide into 15 modules. Main modules are as below:

- GUI: it controls the main data flow in the system.

- Repository: stores all source files, communicate with GUI, Build mother and test harness.
- Builder Server: the main part for this project, it stats the number of build process and send files to child.
- Test Harness: used to test the dll files build by the build server.

Critical Issues

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

- Requests handling- in which Build Server get test requests and change into application readable format.
- Performance- how to deal with the extreme situation, like hundreds of test requests flood in a single XML file or Several XML files contains hundreds of requests.
- Exception handling- how to handle the exceptions occurred in both Build Managerand Build Executive.

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

2. Introduction

There is a desire to create the huge projects with the complex structure to meet the various requirements since the hardware technology and software engineering is remarkable developed.

In a large system, when there is a demand to add new functionality or fix the current errors, we have to test the new code before we insert them into the baseline. we need to extract the property files and generate the test library for the new parts. Build Sever is designed to build test library for files.

2.1 Application Obligations

The primary requirement of Build Server is generating test library for the correspond files. So that the new part will be tested using the library by testHarness, then insert the new part into the baseline. The main obligations of the application should be:

- Generate several request files.
- To accept one or more Test Requests, each in the form of an XML file.
- Accept the corresponding files and create the temporal directories.
- Build test library for each file.
- Build log and send it to the Repository.
- If the build process succeeds, send the test library and test Request to test Harness.
- Test harness test the dll files and send the test result back to repository.In real life scenario, the functionality should also include:

- This should be able to accept the requests from multiple clients.
- This should process the requests form multi clients concurrently.
- This should provide the remote access to the clients.

- This should have a use friendly user interface.
- The time to build a great number of requests should be reasonable.

2.2 Organizing Principles

Most of the functionality listed above will be achieved by splitting them into several small tasks. Each task will be the separate package which will make is more understandable. The various packages that make up the application and will include packages that control the program flow, test the various functionalities, process test requests, perform building of dll's and log the results and execution.

2.3 Key Structural Ideas

When we build a large system successfully, the code implemented must be divided into several small modules and these small modules need to be thoroughly build and test before inserts them into to the baseline or the subset of it. Build server automates the build process. It builds the source code into the dll and generates the corresponding log files. manual build process is very expensive, as the code have a complex dependencies between the solution side and the application side. Automatic build the source code and makes this process concurrently indeed saves developer a lot of time. All the code the module depends on will be tested and make sure the module does not break any code.

The clients can choose source code in the repository, and generates the build requests. The build requests include the build details, such as test driver and all tests needed. The repository will send the request file and the corresponding source to the build mother.

Build mother accept message send from the client, know how many child processes they what to open. Build mother starts send the source files and requests file to child process. After the child process done the build process, mother builder will send new build request to the same child. Until the build requests all done.

After the build server build all the source files and generates the corresponding dll files, they send the dll to test harness and log file to repository. Test harness will auto test the dll file and generate the test result, send the result file to the repository.

3. Uses

The current project focuses on designing and implementing a builder server, and as such, its primary uses will be restricted to build the dll files and logging the build results, and execution summary.

Users and impact on design are listed as below:

3.1 Myself

Use the project to learn how to structure complex code, handle threading problems, and use the WCF framework to pass messages and WPF framework to develop a GUI.

Impact on Design: Will build the communication messages between these servers.

3.2 Quality Assurance Engineers

Uses the project to evaluate student performance, e.g., check requirements, evaluate structure and design, and analyze code metrics.

Impact on Design: I will pay particular attention to demonstration and code structure.

3.3 Instructor

Will examine code structure, look over demonstrations as recorded by TA grader, and look at code documentation and structure.

Impact on Design Process: I come to help sessions and ask questions, test the submission in a new empty directory, and examine each package for proper documentation.

3.4 Developer

Developers are the primary users of build server. Integration tests are a fundamental part of continuous software integration and are used by each developer. They need to build their modules against the entire baseline or some part of it, before integrating it into the software baseline.

Impact on Design Process: Ease of use is the major design impact for the Build Server because the developer may want to avoid using the tool when the UI is terrible. I therefore must provide a nice GUI.

3.5 Managers

Managers can look at the build logs before the build delivery dates to see if the project is in good shape. They can also view the test activity data to see if the deadline would be met.

Impact on Design Process: Managers need the build activity data. Thus, the build server should provide really good logging facilities with proper time and date stamp.

4 Application Activities

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

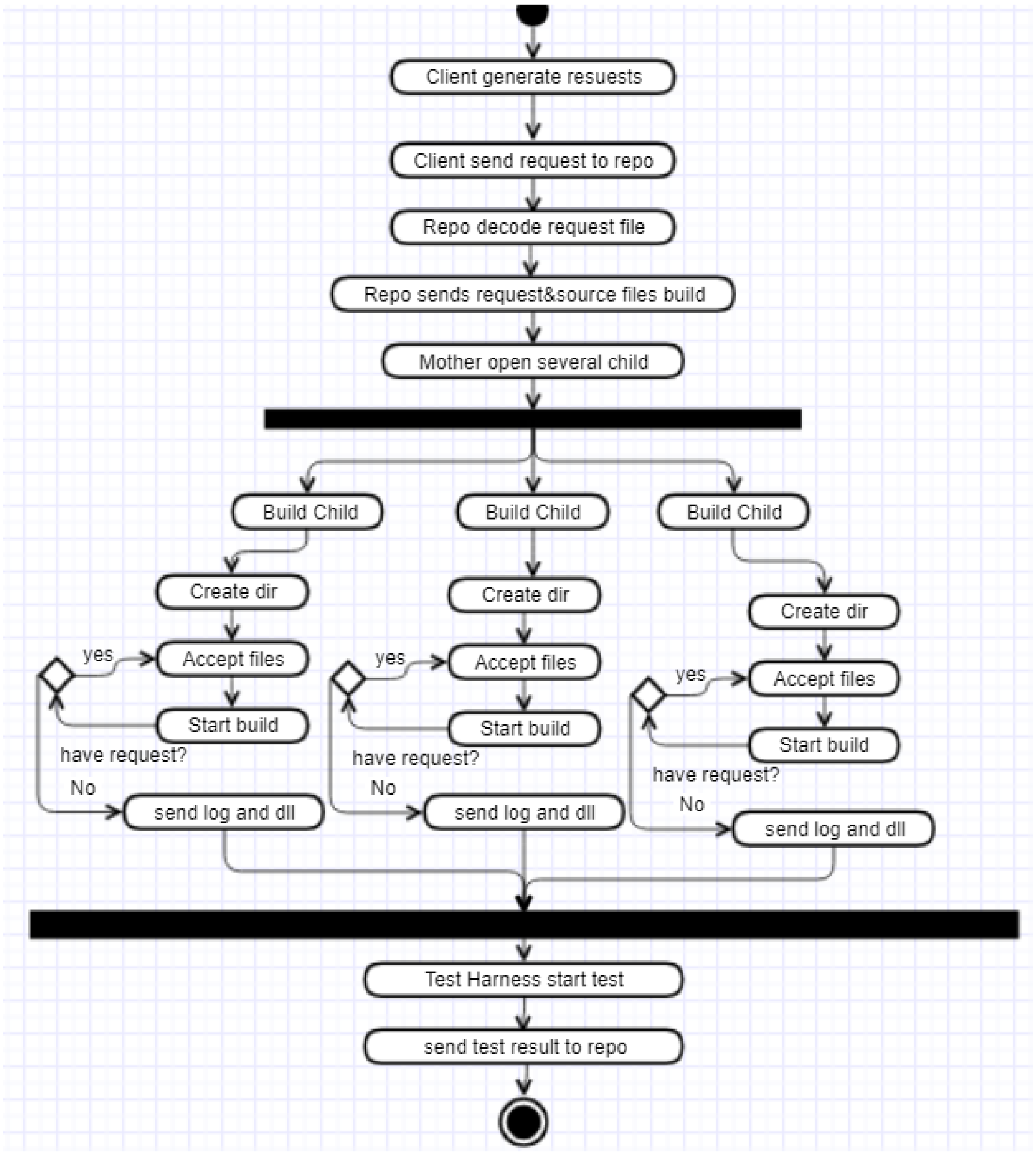

Figure 1. High level activity diagram of Build Server

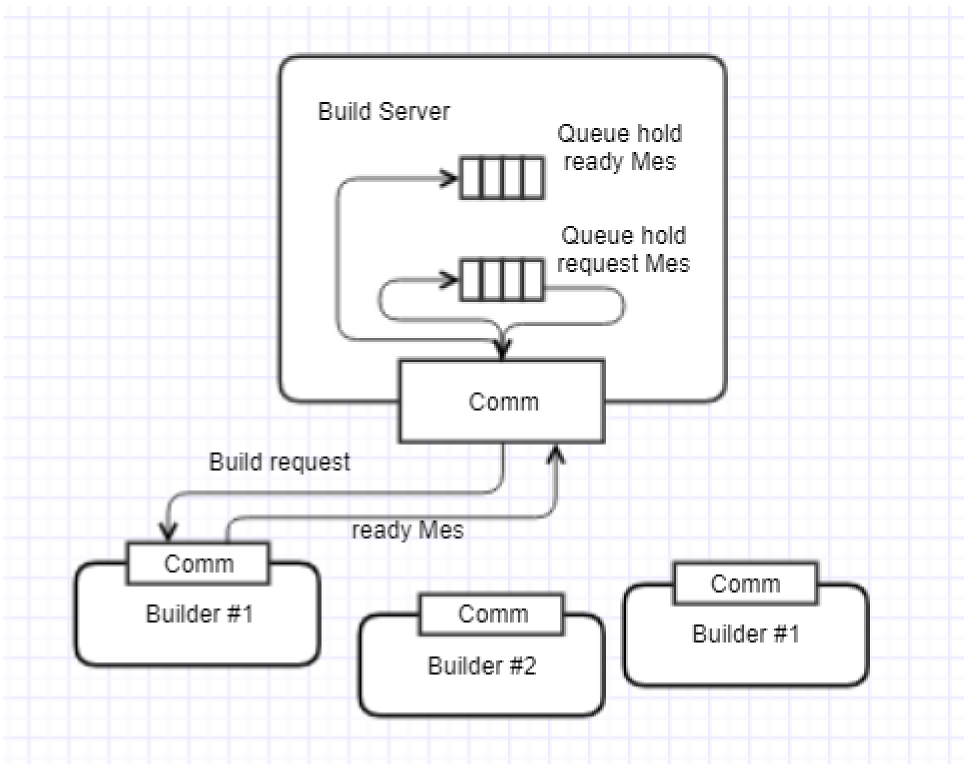

Figure 2. Communication between build mother and child

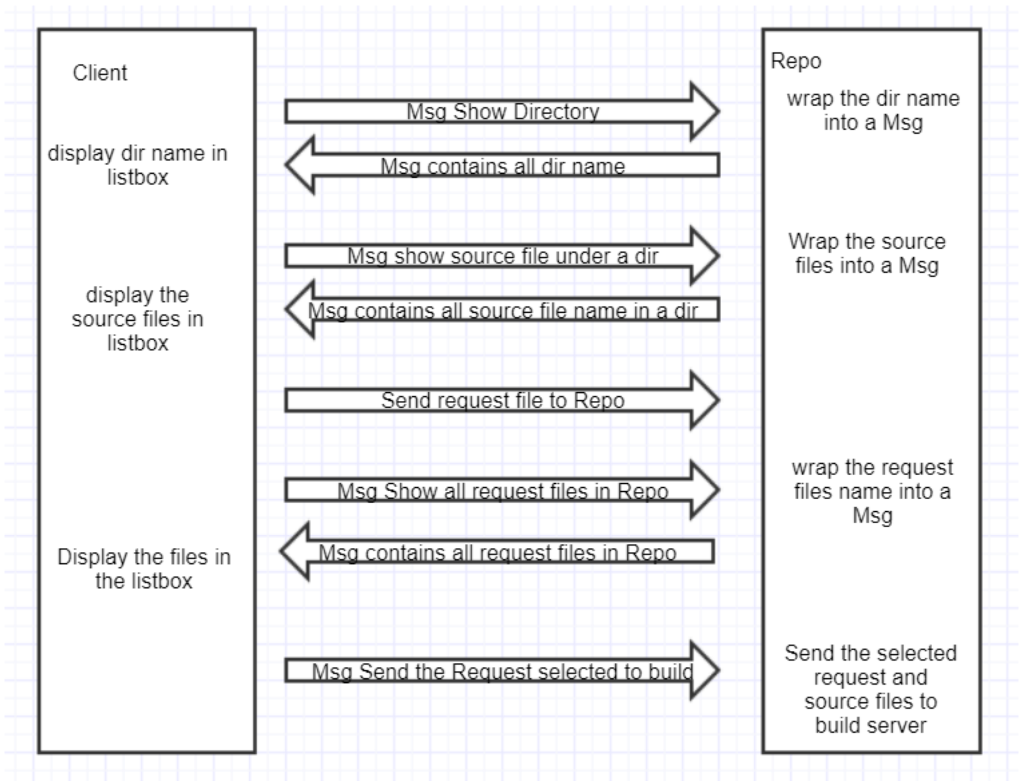
Client
Repo
Msg Show Directory
wrap the dir name into a Msg
display dir name in listbox
Msg contains all dir name
Msg show source file under a dir
Wrap the source files into a Msg
display the source files in listbox
Msg contains all source file name in a dir
Send request file to Repo
Msg Show all request files in Repo
wrap the request files name into a Msg
Display the files in the listbox
Msg contains all request files in Repo
Msg Send the Request selected to build
Send the selected request and source files to build server

Figure 3. Message flow between client and Repo

4.1 Client generate requests

At the beginning, the client sends a show directory message to the Repository. Repository will wrap all the directory which include the source files into a message and sends it back to the client. The name of all the directory will be displayed on the list box. Client chooses one directory and wrap the name of this directory into a message, send to the Repo, Repo wrap all the source files name into a message and forward it to client. The source files' name will be displayed in the list box. Client can choose all the files in the list box and generate a xml request file.

4.2 Client send request files to Repo

Client can choose a request file it generated and send it to repo using windows communication channel.

4.3 Repo decode the request files

After accepting the request files, Repo needs to deco the files to get the source file information.

4.4 Repo sends the request file and source file to build server

After parsing the content of the request files, the repo sends the request files and the corresponding source files to build server

4.5 Build server open several child build processes

Client sends the number of process it wants to open. Mother builder open the number of child build processes.

4.6 Create dir

Child builder will create the corresponding directory to store the source files and the request files.

4.7 Accept files

After the directory created, the child build accepts the source file and request file.

4.8 start build

After accepting all the files, child builder starts to build the source files, and generate the log file and dll.

4.9 send the log to repo, dll to test harness

After build all the source files, the child build sends all the log files which indicated the build status back to the repository, sends the dll to test harness.

4.10 Test harness start test

After accepting all the dll files from child server, the test harness will test all the dll files.

4.11 Send test result back to repository

When the test process done, test harness will generate a test result file, and send it to the repository.

5 Partition

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

Figure 4 is Package Diagram of the system.

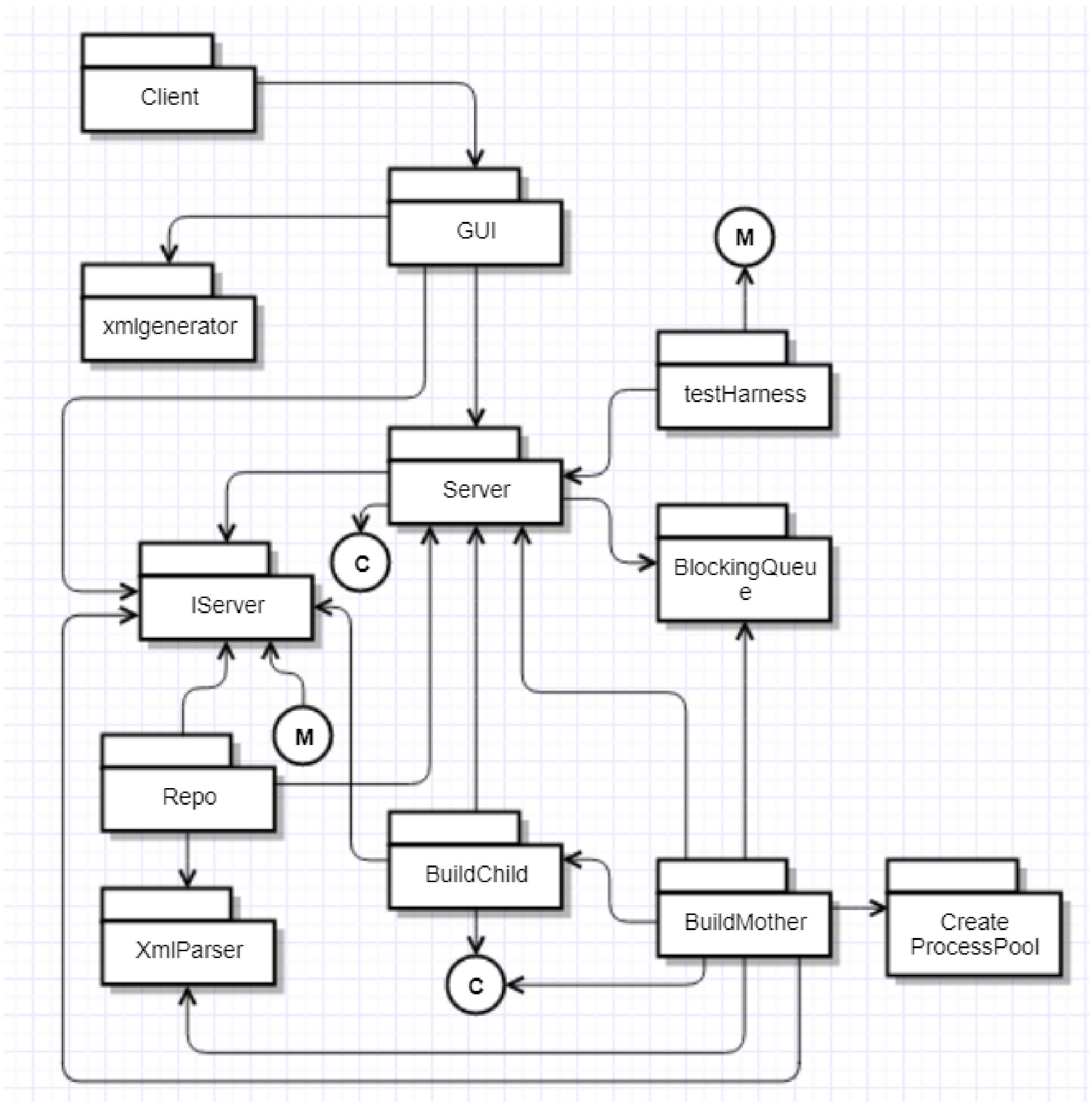

Figure 4 Package Diagram of this system

5.1 Client

Client is the primary user of this system, it mainly used to demonstrate the system meets all the requirements for grader.

5.2 GUI

GUI is a package which mainly controls the data flow and package interactions of this system. The GUI send sends the request message ask for the name of all directories which store the source files in repository. After the name of the directories displayed on the list box, when a directory's name is selected, GUI send request message to Repository ask for the name of source files under this directory. Then, the GUI can select these source files to generate a build request file in the form of XML. Also, the GUI can send the request files in its own directory to repository. When GUI send message to request all build request files in the repository, the name of request files will be displayed on the list box. GUI can send request to ask the repository send the request fills and associated source files to the Build Mother.

5.3 XMLgenerator

This package is used by GUI to generate the request files in the form of xml. It Include generate a request form and save the xml request file under a specific path.

5.4 IServer

Interface used for message passing and file transfer and representing serializable messages.

5.5 Server

Server package provides sender and receiver classes, is the main part for WPF. In the sender class, it has connect, which used to connect with the end point, close, which close the channel, postMessage, which send a message to the receiver, postFile, which send a file to the receiver. Receiver includes start method, which start to listen, getMessage, which get the message from the sender, openFileForWrite, writeFileBlock, closeFile used to accept a files send from a sender.

5.6 Repo

Repository hold all the sources needed to be built. Repository send the sources files name to the GUI used to build the build request files. And send the request files and source files to build mother. Accept Log files from build child and test result file from test harness.

5.7 XmlParser

The XmlParser package used by Repo and build mother. This package can be used to parse the xml files, get the content in the xml file, such as test driver, tests, author, and source directory.

5.8 BuildMother

This package is the main package for the project of build server. It accepts request files and the corresponding source files. Create the temporary directory for the storage of source files. Run a certain number of build child processes. Send the request files and corresponding sources to the child builder.

5.9 BuildChild

This package is the process pool, it will be run by the build mother, and accept request file and source file from the build mother. The most important part for this package is that it will build the source code into dll files and generate the log files which indicate the status of the build. Send the log files back to the repository and the dll to the test harness.

5.10 CreateProcessPool

This create the certain number of child process.

5.11 TestHarness

This package is used to test the dll built by the child builder. After test all the dll files, it will generate a test result file in the form of xml, and send it back to the repository.

5.12 BlockingQueue

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 deQ an item, the reader will block until the writing thread enQ an item. Thus, waiting is efficient. This blocking queue is implemented using a Monitor and lock, which is equivalent to using a condition variable with a lock.

6 Critical Issues

6.1 Test Requests handling

The first question is that what kinds of test request will be accepted by the Build Server? If the request is accepted, how to process the request? Here, the Build server will take the responsibility to accept the request as input, the request should only in the form XML file. Then, it will decode the XML file into application readable format.

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

Solution:

- To ensure the input is readable, request files should be generated only in XML file.
- The XML Decode package will parse the XML files and extract the important information stored to a queue.
- To find the path of the files in the Repository, use the System.IO to search the root the directory.

Test driver example:

```
<testRequest>
   <author>Quanfeng Du</author>
   <time>09/09/2017</time>
   <test name="First Test">
      <Driver>td1.cs
          <tested>tc1.cs</tested>
          <tested>tc2.cs</tested>
      </Driver>
  </test>
  <test name="Second Test">
      <Driver>td2.cs
          <tested>tc1.cs</tested>
          <tested>tc2.cs</tested>
      </Driver>
  </test>
```

</testRequest>

6.2 Performance

Performance issue is the most important consideration when implement the large system. How many the test files a XML file will contain, how many XML files can the Build Server accept, how long it gone take to generate the build library.

Solution:

- The GUI can specify the certain number of child process to build the requests concurrently.
- Build server will allocate enough space to deal with a Build request.
- When XML files are decoded, their valuable information will be stored in string format. Because the string format need less space than the XML files.

6.3 Exception handling

Build Server is the most important part in the federation system, it must be robustness and exception-tolerant enough. How to handle the exceptions and prevent Build Server processing from being stopped?

Every successful system should consider solutions to this issue.

Solution:

- Test request's execution will be isolated that the Build Library processing will not be bothered by the exceptions in execution of build request.
- In the Build Library process, when any unexpected or exceptional situation occur, Exception Monitor package will use The C# language's exception handling features to deal with exceptions.

Exception Monitor uses the **try**, **catch**, and **finally** keywords to try actions that may not succeed, to handle failures when you decide that it is reasonable to do so, and to clean up resources afterward. Exceptions can be generated by the common language runtime (CLR), by the .NET Framework or any third-party libraries, or by application code. Exceptions are created by using the throw keyword. [1]The results will be displayed on the console.

[1]"C# Programming Guide". https://msdn.microsoft.com/en-us/library/ms173160.aspx (Sep. 13,2016)

6.4 Ease of use

Ease of use is a very important feature for a large system. The rigid application is what we should avoid, which means the system only work properly under certain condition and will break when the using environment is changed. Also the complicated interface will discourage the use of the system.

Solution:

Carefully design the code structure, the use cases and their impact on the system should be considered. Make sure a well design Graphical User Interface.

6.5 Unreadable Input

Build Server only accept the XML files as input. When some illegal inputs is occurred, Build Server must handle the unreadable input.

Solution:

Build Server should declare that the input test requests should be in the form of XML files. The get() method in Build Manage will not accept other forms input except XML files.

7 Conclusion

In conclusion, Build Server can receive test requests, build the library successfully and demonstrate results clearly.

The OCD mainly illustrates the package and activity and what users can do with Build Server. Different modes that meet different requirements of users are discussed. Build Server is divided into 14 packages to perform the function.

8 Appendix

8.1 Change of core concept

As the process of implement the Build Server project. The core idea changes a lot from the beginning. In the first OCD, the repository simply copies the request files generated by the client to its own directory, the build server copy the request files and corresponding files to the build server folder. After the build process done, copy dll files testHarness, the build log back to repository. In the final project, the build server, repository and test harness all use message communication with WCF to communicate with each other, sending messages and files. But the main message flow is the kind similar with the beginning, the client generates request files, send them to repository, repo parse the request files, send the request and corresponding source files together to build server.

The other major change is that at the beginning, I simply suppose all source files in the same directory. And clients can choose the source files in the same directory, this is very fragile. In the final project, I create a separate folder for all related sources files, which includes the test driver and all necessary test files.

And in the first OCD, I provide two queues to store the request files and the source files respectively, which is nonsense and inefficiency. In the final project the mother builder store all request files in a list, when mother builder get a request files in the list, it will parse the request files get the name of directory which stores all the necessary files, and send them to a build child.

One of the most important progress in the final project is that the using of process pool. It is a very crucial part in this project. Which can assist the builder server build several requests concurrently. Build Mother will send request to each child builder. After child builder finish the build process, it will send a ready message back to the build mother to get another request files to build, until all the requests files are built.

8.2 Deficiencies

1. It does not support to add multiple test requests in a single request file. Which does not satisfy one part of requirement 11. Since the parse and build request part will need more work, also the GUI part.
2. The test harness is a mock server, it does not use AppDomain.
3. The build server only supports for MSBuild.

9 Reference

https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project1HelpF2017/

https://msdn.microsoft.com/en-us/library/ms173160.aspx