Theerut Foongkiatcharoen

SUID 286170217

BUILD SERVER OPERATIONAL CONCEPT DOCUMENT (ocd)

CSE681 – SOFTWARE MODELING AND ANALYSIS

Table of Contents

[1 Executive Summary 2](#_Toc500378419)

[2 Introduction 3](#_Toc500378420)

[3 Uses 3](#_Toc500378421)

[3.1 Developers 3](#_Toc500378422)

[3.2 Quality Assurance (QA) 4](#_Toc500378423)

[3.3 How will the QA interact with the application? 4](#_Toc500378424)

[3.4 Project Manager 4](#_Toc500378425)

[3.5 How will the project manager interact with the application? 5](#_Toc500378426)

[3.6 Instructor and Teaching Assistants (TA) 5](#_Toc500378427)

[3.7 How will the instructor and TA interact with the application? 5](#_Toc500378428)

[4 Application Activities 5](#_Toc500378429)

[5 Partitions 8](#_Toc500378430)

[5.1 Client 9](#_Toc500378431)

[5.2 Main Window(GUI) 9](#_Toc500378432)

[5.3 IMPCommService (Comm) 9](#_Toc500378433)

[5.4 MPCommService (Comm Message) 9](#_Toc500378434)

[5.5 Blocking Queue 10](#_Toc500378435)

[5.6 Repository Mock 10](#_Toc500378436)

[5.7 Mother Builder 10](#_Toc500378437)

[5.8 Child Builder 10](#_Toc500378438)

[5.9 Build Configure 10](#_Toc500378439)

[5.10 Test Harness 10](#_Toc500378440)

[5.11 AppDomainMgr 11](#_Toc500378441)

[5.12 Logger 11](#_Toc500378442)

[5.13 File Manager 11](#_Toc500378443)

[5.14 XML Handler 11](#_Toc500378444)

[6 Critical Issues 11](#_Toc500378445)

[6.1 Need and Value 11](#_Toc500378446)

[6.2 Solution 11](#_Toc500378447)

[6.3 Performance 12](#_Toc500378448)

[6.4 Solution 12](#_Toc500378449)

[6.5 Development 12](#_Toc500378450)

[6.6 Solution 12](#_Toc500378451)

[7 Changes 13](#_Toc500378452)

[Figure 1 – Remote Build Server Activity Diagram 6](#_Toc500366174)

[Figure 2 – Remote Build Server Package Diagram 7](#_Toc500366175)

[Figure 3 - Build Server Class Diagram 8](#_Toc500366176)

# **Executive Summary**

In general, software development is complex, especially as applications, teams, and deployment infrastructure grow in complexity themselves. Also, software projects involve lots of files consist of a great number of packages and million lines of codes so keeping track of all of these becomes an intolerable burden, and it is arduous and time-consuming when developers attempt to merge their accumulated code changes. These factors combined makes it harder to deliver updates to customers quickly. Hence, with the Remote Build Server concept that efficiently and strongly supports continuous integration, it considerably enhances software quality and development process, and developers can catch bugs early, spend less time debugging, and deliver software more rapidly.

Our Federation of servers contains Client Mock, Repository Mock, Test Harness Mock, and Build Server, each providing a dedicated service for continuous integration, and top level of eleven major packages are the following: Client Mock, Repository Mock, Test Harness Mock, Sender, Parser, Builder, File Manager, Logger, File Handler, Environment Manager, and Build. Each package is accurately described in the Partition section. In addition, users of the Remote Build Server are Developers, Project Managers, Quality Assurance (QA), SMA Instructor and Teaching Assistants. How they will interact with the application is described in detail. Rational and technical explanation for each user is also provided in Uses section. Moreover, the major activities of the Build Server are well represented in the Activities Diagram section, and critical issues regarding performance and need and value arise, but ultimate solutions are provided in detail.

# **Introduction**

This document represents the Operational Concept of a Remote Build Server, an automated tool that locates and fixes potential issues quicker, improves the quality of the software, and decreases the time it takes to release new updates. In traditional software development, developing and releasing software can be a complicated process, and the process of integration generally took place at the end of a project after each person in the team had completed their work given that the big software systems might have a lot of packages, and million lines of codes, and it is extremely difficult to develop, maintain, and publish packages. Also, as the scope of testing increases with each new release, and developers have to verify that new changes are working as expected and ensure that existing features have not broken, it no doubt can quickly become a significant burden. As a result, integration generally took weeks or months and could be very painful and long and tense integration. To solve this problem and successfully implement big systems, continuous integration comes into play and efficiently supports continuous integration by automating the process. It is the process of thoroughly automating the build and testing of each code before developers insert their codes into the software baseline so that they can catch integration bugs early and accelerate collaborative development. To efficiently support continuous integration, we use a Federation of servers that consists of Client Mock, Repository Mock, Build Server, and Test Harness Mock, each providing a dedicated service for continuous integration. Each check-in from Client Mock is then verified by an automated build from Test Harness Mock, allowing teams to detect problems early because you can spend less time debugging and keep the build fast.

# **Uses**

User and Uses basically define whom the users of an application will be and how they will use or interact with the application. The major consideration in software development is the users of the application, and each has unique role and different perspective. Therefore, Developers, Project Managers, Quality Assurance (QA), SMA Instructor and Teaching Assistants will be the users of this application. Below describes the way in which each user uses the system differently in detail:

## Developers

Developers usually write and submit a lot of codes as a result, undoubtedly, to merge their codes as often as possible, they will be requesting tests (at least once a day) to ensure that new changes are working as expected and existing features have not broken, meaning the new parts are compatible with the baseline so those parts will become part of the current baseline without any functional or integration errors.

**How will the developer interact with the application?**  
Developers will check-in (commit) their codes by selecting driver and tested files to Repository Mock so that the repository mock will contain all the codes after they check in their changes to it, meaning pushing local development code changes to the repository, they build an XML Build Request, specify the number of child builders to be started and send build request structures to the repository for storage and transmission to the Mother Builder to build test libraries, and command the test harness to execute tests if the build is successful, and receive build and test results from the Repository Mock. Also, they can shut down Pool Processes after they finish building and testing or start again at the beginning.

## Quality Assurance (QA)

Quality Assurance (QA) plays an essential role in software development because the QA ensures that products meet certain standards of quality, and he/she focuses on accomplishing satisfying user experience within the given time frame. To avoid common causes of bugs, the QA manages the baseline of a large software system to efficiently support continuous integration. In short, the QA can evaluate the project and estimate the quality of projects by checking that the project functions properly and efficiently by ensuring all development tasks meet quality criteria through test planning, test execution, quality assurance and issue tracking.

## How will the QA interact with the application?

Roles and responsibilities of quality assurance and developers are somewhat similar. The QA will use the whole application by submitting their codes including driver and tested files to Repository Mock in which a source code control system keeps all of a project's source code in as well as generating an XML Build Request, specifing the number of child builders to be started and sending build request structures to the repository for storage and transmission to the Mother Builder to build test libraries, and command the test harness to execute tests if the build is successful. Then, the QA will receive build and test results telling if their latest code builds are successful or not. However, a common mistake might occur when developers forget to include everything in the automated build or in the repository so the QA will check the completed logs and focus largely on some test metadata such as developer names and dates, test scripts, properties files, database schema, install scripts, and third-party libraries given that these code elements might cause some issues in the baseline. In a similar manner, the QA might make their build self-testing because it is an effective way to catch bugs more quickly and efficiently by including automated tests in the build process

## Project Manager

Everybody has responsibility for the release process, and project manager is one of them because the project managers should plan projects with attention to deployment and details such that they might develop new techniques for scoping, planning and de-risking projects, and the delivery of services. Typically, they know what developers and QA are working on and can see their progress to keep things on track. They continuously develop and communicate the vision to the development teams and define the features of the system. Obviously, the role of product manager can be a very stressful one. It is quite essential for product managers to completely understand the impact their choices have on software operability, and for engineering teams to have a good understanding of how their choices significantly affect product. Meanwhile, it is frustrating since their roadmap of new features can be randomized by technical debt or operational issues. It is more to keep in mind when planning work and running team like process control, project planning, managing relationships as well as team staffing.

## How will the project manager interact with the application?

They can check if the specific module is perfectly coded to make sure everything is working as the client requested. For example, they can see the test logs that which component of the project has issues and will need extra resource in order to visualize workflow to better understand what is going on and what can be improved.

## Instructor and Teaching Assistants (TA)

The main purpose of SMA Instructor and Teaching Assistants for using this application is to evaluate its functionality and finish grading This will involve applying to unique QA, developer, and project manager roles.

## How will the instructor and TA interact with the application?

SMA Instructor and Teaching Assistants will use the whole application as same as developers and QA do so they can check whether the system works in an expected fashion without any issue. Also, they can ensure that the code is clean and simple although it might require minimum efforts.

# **Application Activities**

The client can select files from GUI to upload from local storage to Repository or select files in Repository for packaging into a test library, a test element specifying driver and tested files, added to a build request structure. The GUI will display which selected files client added to build Then, the client builds a build request to generate an XML build request, and the GUI will display the build request structure that the client has just created. After that, the client selects newly generated build requests and send them to the Repository Mock, and the GUI will display that client has transferred the build request structure and XML build request files successfully. Then, the Mother Builder will receive a build request message forwarded by Repo and check the ready message queue to see which child processes are available. Once it found that there’s a child process available, it will send a build request message to that child process. When the child builder receives a build request message, it will deserialize the XML build request and asks repo for the cited files. Once it receives those required files, it will build test libraries if there are more than one test element, generate an XML test request and send the log to Repo and test libraries to Test Harness. Test Harness will then check a test request message, and if there is a test request message, it will deserialize the XML test request and loading test assemblies into application domain, recording pass status, logging execution details, sending test result to Repository. After that, the client can view the test logs and build logs from GUI.

For detailed information, it is represented in the form of an activity diagram as illustrated below.

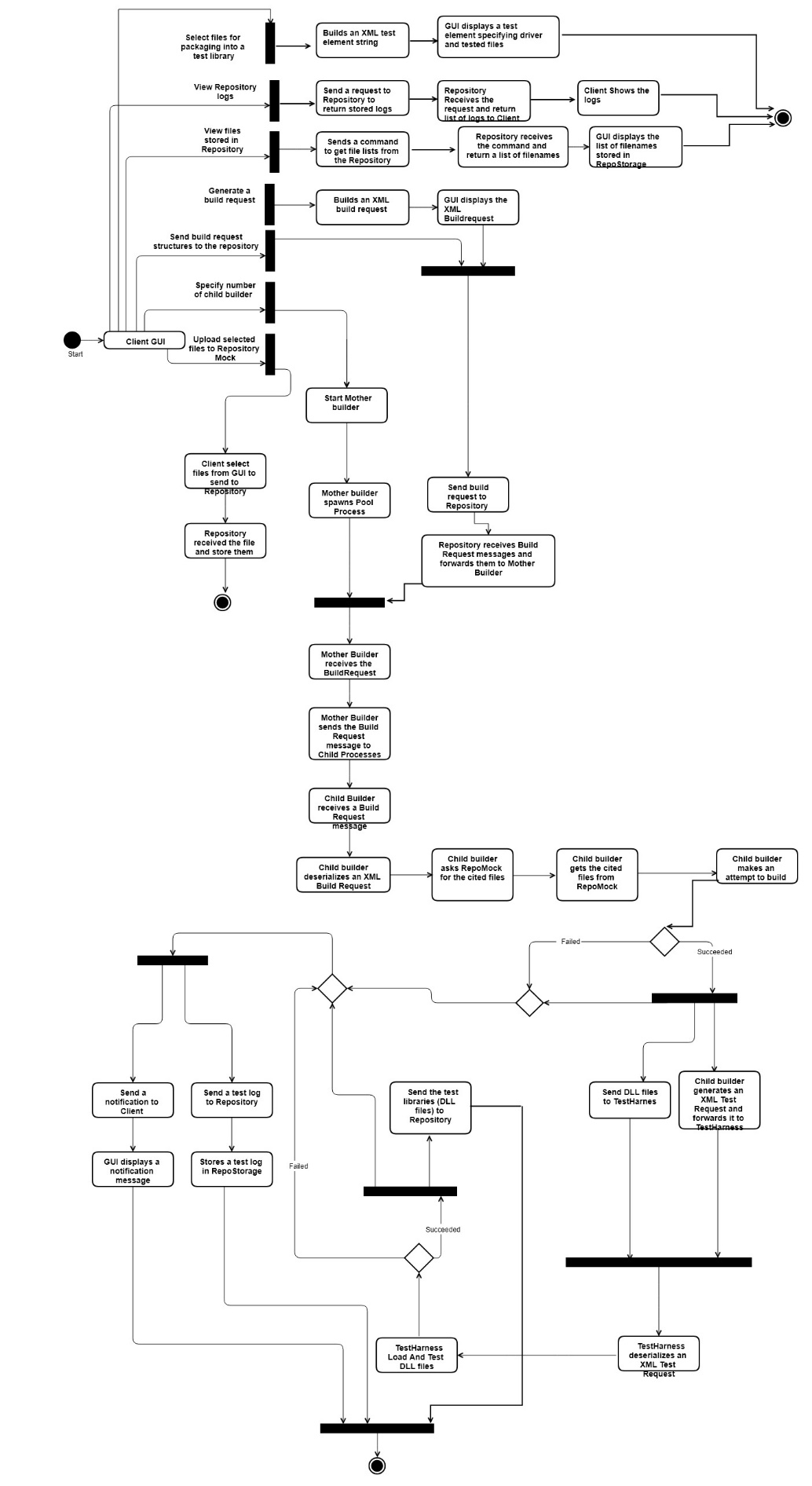
****

Figure 1 **– Remote Build Server Activity Diagram**

# **Partitions**

This section will layout the technical breakdown of required functionality for the build server and each task becomes a package and class candidate. Here are the following eleven major packages and build server classes: Client, Main Window(GUI), IMPCommService (Comm), MPCommService (Comm Message), Blocking Queue, Repository Mock, Mother Builder, Child Builder, Build Configure, Test Harness, File Manager, Logger, XML Handler, AppDomainMgr, MpCommService and IMPCommService comprise of multiple packages, meaning I will not go into lower level these two packages. However, we are talking about a Remote Build Server so I will give a clear explanation regarding what all functionalities it will require in detail.

The figure below illustrates package diagram for the Remote Build Server.

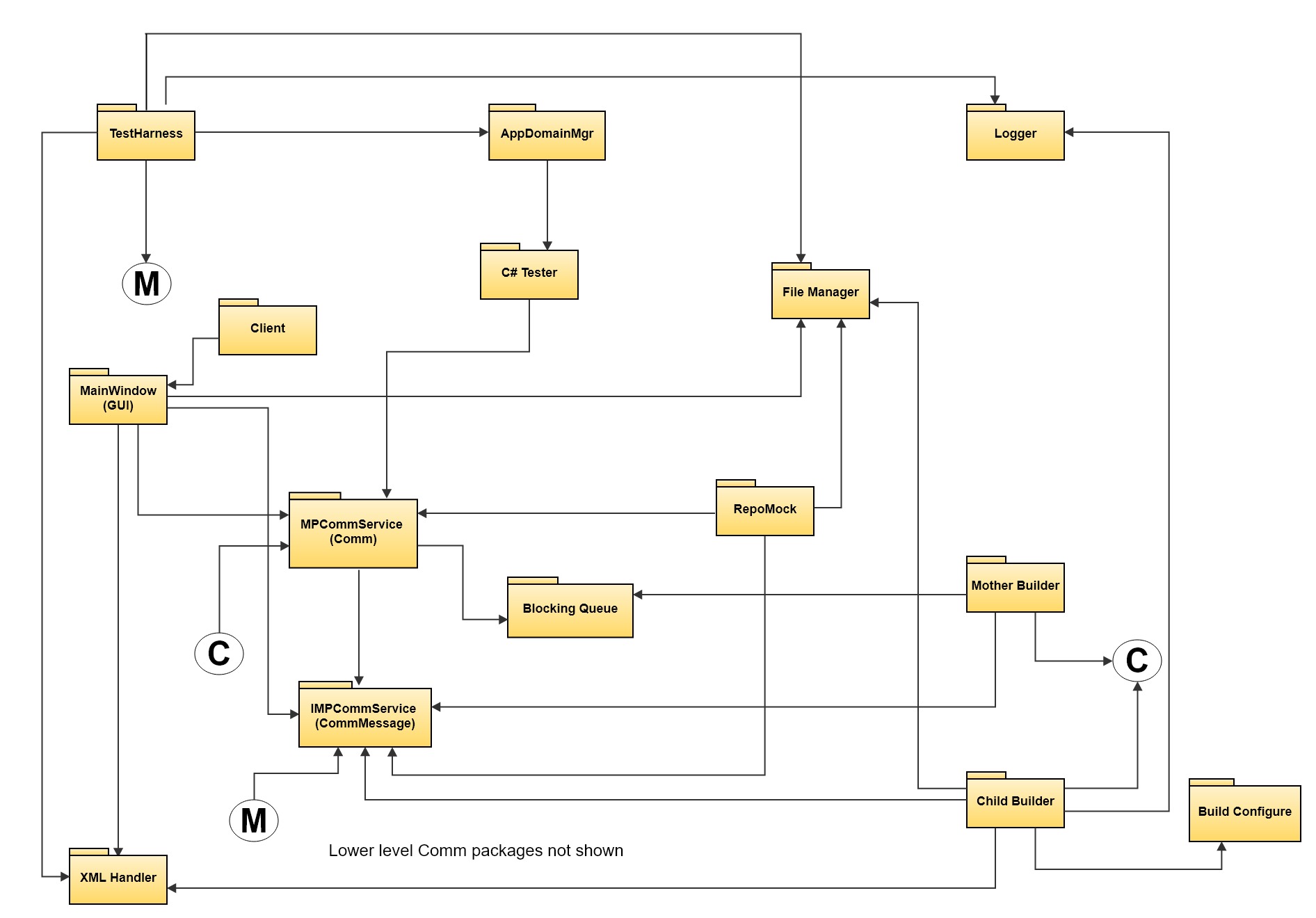


Figure 2 **– Remote Build Server Package Diagram**

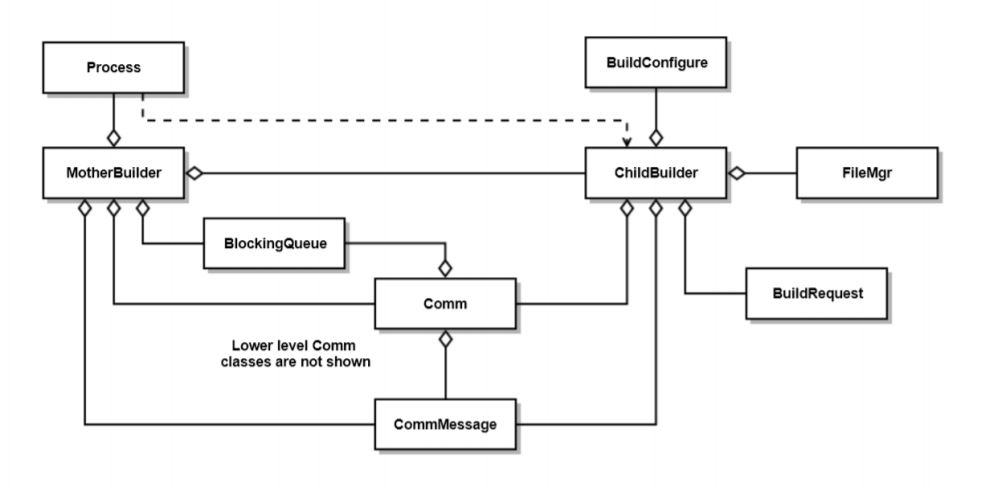


Figure 3 - **Build Server Class Diagram**

## Client

Client Mock can select files to upload from local storage to Repository or select files for packaging into a test library, a test element specifying driver and tested files, added to a build request structure. Also, it can specify the number of child builders to be started, shut down Pool Processes, and send build request structures to the repository for storage and transmission to the Mother Builder.

## Main Window(GUI)

Main Window is implemented with WPF and using message-passing communication (WCF). It gets file lists from the Repository and enables building an XML Build Request by selecting file names, uploading selected files, driver and tested files, as well as browsing to find files to build from the Mock Repository, and provides mechanisms to start the main Builder (mother process), to specify the number of child builders to be started. It also provides the facility to ask the Mother Builder to shut down its Pool Processes, to send build request structures to the repository for storage and transmission to the Build Server, and the capability of repeating that process to add other test libraries to the build request structure.

## IMPCommService (Comm)

IMPCommService is service interface for MPCommService including interface used for message passing and file transfer and class representing serializable messages. It also defines a service contract given that the service contract specifies what operations the service supports. Each method in the IMPCommService corresponds to a specific service operation.

## MPCommService (Comm Message)

MPCommService implements service interface. It implements postMessage, getMessage, and postFile functions using Sender and Receiver instances. A postMessage function sends CommMessage instance to a Receiver instance. A getMessage function retrieves a CommMessage from a Sender instance, and a postFile function is called by a Sender instance to transfer a file.

## Blocking Queue

This package implements a generic blocking queue using a Monitor and lock, which is equivalent to using a condition variable with a lock. It also demonstrates communication between two threads using an instance of the queue. If the queue is empty when a reader attempts to dequeue an item, the reader will block until the writing thread enqueues an item. Therefore, waiting is efficient.

## Repository Mock

Repository Mock is a storage house of the whole system due to the fact that it is a central location in which data is stored and managed, meaning this is where the source code lives and it contain all the code after the user checks in their changes to it. It is a simple server, running it own process, using our Message-Passing Communication, to send and receive requests and replies. Also, it uses File Manager package to handle file retrieval/storage to accept files including source codes, XML build requests, and Test Libraries.

## Mother Builder

Mother Builder provides a queue of build requests and ready messages so that it can manage Child Builder by spawning the specified number of child processes given by client as well as passing them Build Requests when it receives ready messages from them.

## Child Builder

Child Builder, spawned by Mother Builder, uses a Message-Passing Communication Service built with WCF to access messages from the Mother Builder process by sending a ready message to inform a mother builder that it is ready to retrieve a build request.

Once the Child Builder receives a build request message, it will load files, matching a retrieved Build Request, from Repository after performing XML deserialization. After that it will build those files into libraries, send them to the Test Harness, and uses to Logger to alert the client whether the build is successful or not and sends the log to Repo.

## Build Configure

Since the toolchain, containing Debugger, Compiler, Linker, and another tool for a specific programming language to produce executable files and shared libraries for the target, comes into play immediately after the merge commit happens, Build configure package sets up the environment that supports development and operation tasks and includes a specific set of tool integrations to prepare for build.

## Test Harness

Test Harness Mock provides the system for automated integration testing. It runs tests based on test requests and libraries sent by the Child Builder if the build succeeds through a Message-Passing Communication Service built with WCF. Generally, it runs a library consisting of test driver and a small set of tested packages, loading test assemblies into application domain, recording pass status, logging execution details, sending test result to Repository.

## AppDomainMgr

It gets the domain manager that was provided by the host when the application domain was initialized so that test harness can use it to load test assemblies.

## Logger

Logger is in charge of recording logs for test libraries built by Build server and test executed by Test Harness Mock owing to keeping track of all the test results. The user can review logs to ensure tests passed and troubleshoot when the tests failed. As a result, logging increases efficiency of codes.

## File Manager

File manager provides file management capabilities. It creates, delete temp directory, searches for files with a specific file extension or name.

## XML Handler

XML Handler is responsible for serialization of objects in XML format and deserialization of an XML file back to an object. Generally, serialization is a process by which an object's state is transformed in some serial data format, such as XML or binary format and is the process of converting an object into a form that can be readily transported. Deserialization, however, is used to convert the byte of data, such as XML or binary data, to object type.

# **Critical Issues**

## Need and Value

Although a remote build server is the need for today’s organization and reduces integration problems allowing us to deliver software more rapidly, the initial cost of implementing this including initial installations and configurations is costly and can create disruption if it makes no difference in minimizing the cost of integration or freeing developers from manual tasks. It is not worth it that the organization spends a significant amount of money on a basic remote build server that is not helpful and does not overcome the learning curve before implementing this, meaning all utilities of application will be useless. For instance, if it does not help us identify our integration problems earlier or it does not save us time and verify code that much, the organization waste the money for nothing. This could be one of the critical issues in terms of ease of use and would rather be federation issues than the developers.

## Solution

The business stakeholder should contemplate if it is worth for this type of project or the organization really needs this so they can minimize the cost of integration or freeing developers from manual tasks

## Performance

Build Server in the model will have lots of build requests on it, and if something goes wrong, the whole system will go down and crash, and this could be a critical issue. In terms of time consumption, if one task throws an unexpected exception only the process that was running the task will go down instead of having the whole system going down. Similarly, there will be an issue storing and processing that can throw an unexpected exception when the user is sending too many test requests to the Repository Mock. Also, if C++ headers include each other in the absence of an include guard, a file will need to be processed multiple times and can cause significant build delays in large systems. Also, the compiler will endlessly preprocess the header and use CPU time until the computer gives in and halts.

## Solution

For the Build Server issue, providing process-pool helps increasing performance and reliability because the process-pool helps dividing tasks on different processes which can be used to efficiently delegate work over multiple CPU cores.

For the Repository Mock issue, using a blocking queue approach to deal with test requests can solve the problem due to the fact that a thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue. A thread trying to enqueue an item in a full queue is blocked until some other thread makes space in the queue, either by dequeuing one or more items or clearing the queue completely. In a similar manner, applying multithreading to this code can enhance performance and concurrency on multi-processor machines by paralyzing the tasks.

## Development

Defining a single message structure that works for all messages used in the Federation is quite difficult, and managing Endpoint information for Repository, Mother, and Test Harness.

## Solution

A message that contains To and From addresses, Command string or enumeration, List of strings to hold file names, and a string body to hold logs will suffice for all needed operations in case of defining a single message structure that work for all messages used in the Federation. Also, storing Endpoint information in XML file resident with all clients and servers and load at startup can help managing Endpoint information for Repository, Mother Builder, and Test Harness.

# **Changes**

The first, Project #2, implements a core Build Server functionality that communicates with a mock Repository, mock Client, and mock Test Harness, all residing in the same process using Message Passing. Its purpose is to allow the developer to ensure that it functions as expected.

The second, Project #3, develops prototypes for a message-passing communication channel implemented with Windows Communication Foundation (WCF) because all members of the Federation have to use Message-Passing Communication to communicate. Also, between child builder need Message-Passing Communication to communicate with parent Builders, and a WPF client that supports creation of build request messages. Basically, the Remote Builder will be accessed remotely from a GUI built using Windows Presentation Foundation (WPF), and the GUI enables building test requests by selecting file names from the Mock Repository, provide mechanisms to start the main Builder (mother process), specifying the number of child builders to be started, and provides the facility to ask the mother Builder to shut down its Pool Processes by sending a single quit message.

Finally, the third stage, Project #4, completes the build server. It 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 Mother Builder, and each Pool Process shall attempt to build each library, cited in a retrieved build request, logging warnings and errors. If the build succeeds, the child builder will send a test request and libraries to the Test Harness for execution, and shall send the build log to the repository. Once, the test Harness receives a test request message, it wll attempt to load each test library it receives and execute it and then it submits the results of testing to the Repository.