SOFWARE COLLABORATION FOUNDATION

Saisumanth Gopisetty
831608197
sgopiset@syr.edu

Dr Jim Fawcett
Systems Modelling and Analysis
FALL 2016

Table of Contents

Table of Figures

Software Collaboration Foundation

Executive Summary

Software development is a process oriented designed based execution which involves many people from multiple locations. Earlier, when the teams were small and the projects were relatively tiny when compared to now, there were no obligations on software development. But, with the advent of huge usage of software all around the world in every aspect of life, it is mandatory to choose a process oriented approach while developing software to coordinate hundreds of developers and future maintenance of the developed software. We need a federated system which is collections of federated servers in which each federated server does a predefined task and communicates with the other servers accordingly. One of the solution to this is Software Collaboration Federation(SCF).

Software Collaboration Federation (SCF) is a collection of clients and servers and associated software designed to support activities of a software development team.

In software development, which is an agile process, there is a need to track the continuous development as well as to perform integration test on the modules developed by developers from multiple locations. SCF manages a team in helping them plan the development, collaborate work done among the team members, set up meetings, track the progress of the development, ensure the quality of work being done etc. This document gives insight into how each of this task is accomplished by the federated servers of the SCF.

Architectural Ideas

Software Collaboration Federation(SCF) is designed for a software development team. It is highly possible that developer contribute for the project from multiple locations. So, managing the whole system with one servers is impossible as it builds load on the system. So, SCF will have several distributed virtual servers. All the servers collaborate through efficient message passing communication to provide full-fledged services to the SCF users. Even though the servers are distributed they appear as a single instance working in virtual environments. As all the servers are interconnected, they can be authenticated at the main server and can be routed accordingly. The architecture of SFC is accommodated to have the following federated servers

Organizing Principles

We abide by some core organizing principles while we build SFC. They help us to design the system which accomplishes all the tasks that we intend to do as a part of SFC.

CTAI and Deployment

In an agile project where there are strict timelines for the mile stones, it is highly impossible for the serial accomplishment of Integration Testing and deployment. To achieve effective results, the code is tested for integration for every check-in and it is deployed if every branch in the module passes the integration test.

Versioning mechanism

Every code is assigned an automatic version number for every Check-In into repository. No file is overridden and every file which is checked in is assigned a new version number. In this way, all the older versions are preserved and can be backtracked easily. All the meta data of every file is maintained in a manifest file or may be in NoSQL DB like MongoDB which is identified by a distinct key.

File Locking

When multiple clients try to access the same file in either Repository or Collaboration Server, it creates chaos. To avoid this situation, file locking mechanism is employed so that only one client can access a file at a point of time.

Integrity of the code

Invalid code is not allowed to be checked-in into the repository. When a developer checks in the code, it is verified for compile time errors before checking in the code. If the compilation fails, proper error message is sent to the corresponding client who tries to check-in that code.

Caching

When Client requires a file, he makes a request to the Server. In cases when the same file is requested multiple times, the same request is triggered multiple times which creates load on the servers.
To avoid this situation, Local caching can be done on each client and server.

- When a client wants a file, the local cache is searched first before making a request to the server. If the file is found in cache, it is returned. Else, the request is made to the repository.
- When a request comes to the server, the local cache of the repository is searched. If the file is found in the cache, it is sent to the requested client. Else, the file is searched in the database and returned to the client.
- Local cache is updated regularly by efficient purge algorithms like Lest Recently used algorithm.

Asynchronous Operations

All the message passing between servers is done asynchronously to avoid wastage of time while it waits for the response. Clients send multiple messages to Collaboration Server and Test Harness for various purposes asynchronously.

Notification Services

Notifications are sent/received among various servers for the efficient understanding of the activities of each server.

1. Clients are notified about the test results when they send a Test Request to the Test Harness.
2. Clients receives notifications from Collaboration Server when they are scheduled for a meeting by their manager.

3. Repository receives notifications when a new build is tested for integration.
4. Test Harness server receives notifications when it receives Test Requests from Clients or Repositories.

Uses and Users

Customers

The customers are the people who pay for the software. They are the end customers who use the software. Customers use SCF for following purposes:

1. To upload the requirement documents to the Software development team proposing their requirements.
2. To track the status of development status of the project
3. To schedule meetings with the development team regarding the progress of the project.
4. To report the bugs after the deployment of the software on their machines.

Impact on Design:

A virtual display system is developed to upload documents, schedule meetings, track the status of the project etc.

Technical Architects

Technical architects are the channels of communications between the customer requirements and the developed software. They develop high level architectural document by closely understating the business requirements of the customer. They verify it with the customers before developing High level design document.

They use SCF to accomplish the following tasks.

1. To understand the customer requirements and accordingly hold meetings with them to discuss about the requirements in detail.
2. To develop High level documents to ensure that the project will meet all the requirements stated by the customers.
3. To understand if they can reuse any modules that are already present in the repository to minimize the effort required to develop the software.

4. To set up goals for the teams and design project milestones to ensure that the project is progressively achieving the desired results under strict timelines.

Impact on Design:

1. A virtual display system is designed to talk to the customers regarding the project requirements and status of the project.
2. Version controlling is enabled on the repository to save all versions of the documents by the architect. This helps to the understand the progression of the project.
3. A high-level scheduling system is incorporated in the Architect GUI as he is very busy managing customers and the leads.

Developers

A work package[1] is assigned to a developer. The developer reads the functional requirements given by the architect and writes a Design level document and gets it approved from the architect before starting developing it.
Developers use SCF

1. To save their code incrementally by check-in process in the repository so that they don't lose the progress as well as collaborate their work with the other team mates.
2. Extract code from the repository so that they can start working on them to fix bugs or to develop enhancements on the existing code.
3. To understand their current milestones and future milestones so that they can work accordingly.
4. To understand the progress of the component they are working and get all the required documents from the repository.
5. To query the repository to understand the status of a test case.
6. To communicate with the peer team members as a part of the software development.

[1] A Work package is a unit of development activity. It contains the description of the work to be done. Every work package has a single tangible result associated with it.

7. To test the modules built by them by submitting them to Test Harness.
8. To be able to understand which modules can be reused by looking at the source code and their corresponding documents from the Collaboration Server.
9. To fix any bugs raised by the Test Harness which are received in the form of notifications from Test Harness.

Impact on Design:

1. A virtual display system is built to cater the developers to view repository, query it etc.
2. A chatting and video conferencing is enabled on their GUI to talk with the peer developers.

Leads

Leads are responsible for the complete development of a module which is designed by the architect. They use SCF

1. To design milestones for each developer and assign work packages to them.
2. To track the status of the development of the module.
3. To set up meetings with the developers who are working on the module assigned to him.
4. To report the status of the development to the manager.

Impact on Design:

1. A visual display system is developed to set up meetings with the team members working on the module and track the progress of the development of the module.
2. The Display system will allow him to view the code in the repository and understand the analysis provided by agents in the repository. This will help ho advice the developers to work accordingly.

Managers

Project Manager manages the project in a process oriented way so that project will be accomplished in scheduled time lines.
Managers use SCF

1. To track the status of the project and understand the future requirements, road blockers and risks.
2. To manage the budget of the project and manage the resources under him.
3. To collaborate with architects and hire the essential resources with skill sets requested by the architect.
4. To hold business level meetings with the clients.
5. To analyze the progress of the project and take necessary actions if required to run the project smoothly and accomplish it in stipulated time lines.
6. To track the progress of each developer so that his quality of work can be accessed.
7. To understand the summary of Test Results which are notified by the Test Harness after every run.
8. To Hire new resources if the project requires by analyzing the required skill sets and searching for them in the job descriptions available in the Collaboration Server.

Impact on Design:

1. A virtual display system is enabled on Manager machine which helps to track the progress, manage the budget etc.
2. High level work schedules, video conference capabilities are enabled in the virtual display of the manager machine.

QA's

QA's use SCF

1. To run performance test, load test and stress test on one or many or all modules of the project to ensure the quality of software built.
2. To update the testing progress to senior managers.

Impact on Design

Test Harness is enabled to run bulk tests using efficient mechanisms like multi-threading and the results are notified to the QA.

Structure

SFC is organized in the following stated structure to enable it to meet its requirements:

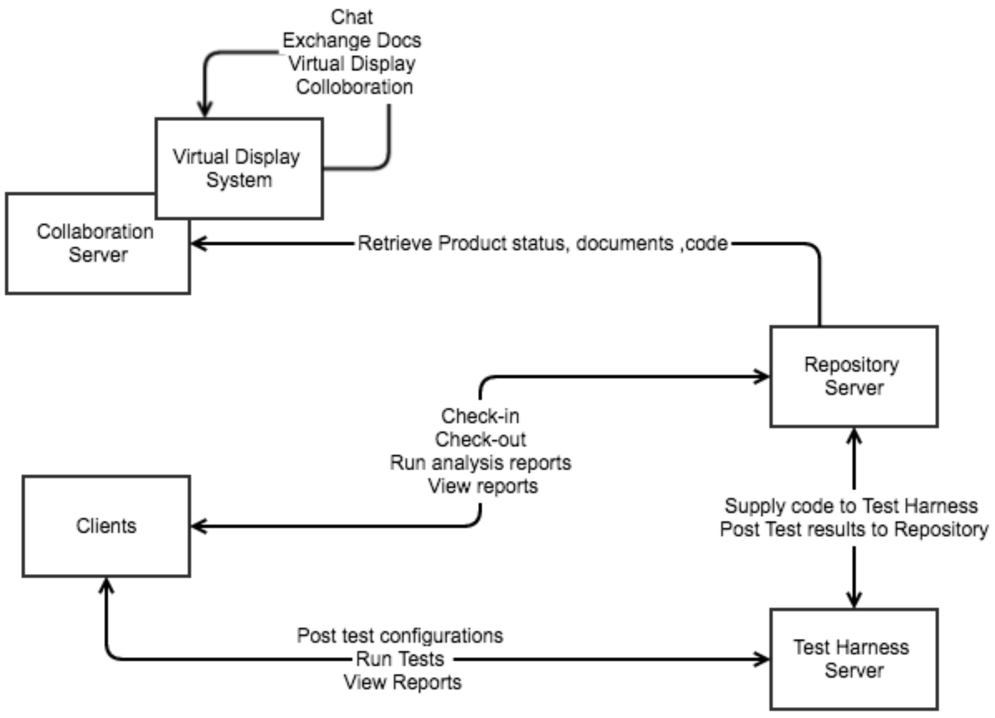


Figure 1 Software Collaboration Foundation Structure

Client

Clients are any team members who use SCF for the software development purpose. They may be Developers, Managers, Architects, QA etc. All the clients are interconnected to allow them to collaborate their work. The Client will have the following responsibilities:

1. Allows developers to check in code into repository and maintain all the code they have developed in proper versioning format.
2. Allows developers to check-out code from repository to develop their module on top of it, fix bugs or tweak the code to enhance the functionality.
3. The virtual display system on the client machine helps the developers to view documents, the results of the testing, their mile stones, code from the

repository, web cam views, various charts and tables that describe progress and the interactions between the peer developers.
4. Allows them to update status of their work to the architects as well as managers by sending the status to the collaboration server.
5. They should be able to test their code before checking in by sending the code to Test Harness.
6. They should be able to query the repository for a file/string etc.
7. They should be able to browse the code from the repository.

Repository

Repository is the warehouse of Source code of the project. It the important federated server which plays major role in collaboration of source code among the team members. Having a central server which has all the source code helps to integrate the work done by multiple developers in a process oriented way. The responsibilities of Repository are

1. It holds all the source code of the project.
2. It is a distributed across multiple virtual servers across multiple locations.
3. The code checked in by a developer can be extracted by any other developer in the team and start working on it.
4. It stores code based on dependency-based storage so that any version of the code can be pulled out without any mismatch.
5. It allows clients to query on it to extract any file from it based on the search of the client using efficient mechanisms like indexing.
6. It provides exploratory tools based on the metadata of the files stored in it.
7. It can configure agents on it which can analyze the code and draw various useful results.
8. It should have a build server which builds and catches execution images from the latest code in the repository and sends it to Test Harness.

Test Harness

Test Harness is a federated server which is core of ensuring that the code which is being developed by the developers is passing the integration system and deploys the latest build on client machines regularly. Apart from Integration testing, it should also provide Performance Testing, Load Testing, Stress Testing etc. Test Harness is delegated to do this. Test Harness loads and executes images provided by the build server. Apart from doing CTAI[2], it also does continuous deployment so that all the changes made by the developer after integration testing are reflected at the customer end automatically.

1. It is essential part of CTAI which helps in continuously performing integration tests (Regression Test) on the checked in code so that we can ensure that newly checked code is not breaking the working functionalities.
2. Apart from that, Test Harness accepts test requests from multiple clients, process their messages using efficient mechanisms like multi-threading and returns the results to them.
3. Notifies the Clients who own a module which fails an integration test so that they can fix it and check-in again as soon as possible.
4. After every test, the test results are sent to the corresponding developer who has developed it and send the logs to the repository for future reference.
5. After the code passes integration test, Test Harness deploys the code on the customer machines automatically.
6. So, Test Harness is responsible for Continuous Testing, Integration and Deployment.

Collaboration Server

A Central hub for all the project management information. The responsibilities of the Collaboration Server are

1. It supports project management, storing work packages details and the all the documents related to project management.
2. It maintains the mailing list for each activity and each developer so that appropriate mails are sent when developer updates his status to the managers.
3. It helps the managers to organize the resources and hire new resources with the required skillsets.
4. It should be configurable to schedule the deployment of new build on the client machines.

[2] CTAI- Continuous Testing and Deployment

5. It should have all the budget information of the project.
6. It helps to schedule meetings between different teams and blocks their schedules and notifying them at the time of meeting.
7. It holds all the data of the Team members, whom they report to, number of bugs fixed by the developers etc.

Virtual Display System

This federated server provides an interface for collaboration between the teams during the software development process. It is tightly integrated with Collaboration server to drive the collaboration activities. It allows he team members to connect verbally as well as visually to connect to each other.

1. It interacts with collaboration server to provide sophisticated interface for publishing all the project management data.
2. It helps the remote teams to interact with each other by providing webcams, video conferencing etc.
3. It provides visual diagrams like Pie charts, Bar graphs, sketches, diagrams etc. from the data procured from Collaboration Server.
4. It supports team meetings by allowing people from multiple locations to chat, video conference, discuss about the project status, analyze code etc.
5. Virtual Display Clients allow the clients to view the repository code, to talk to the peer team members, to view the documents from Collaboration Server etc.

Critical Issues

Knowledge of various components of SCF

As SCF is large system with a lot of federated services, it is difficult for an employee who starts working for the project to understand the processes undergoing in all the Servers.

Solution

Various artifacts must be created by the development team about the use of each system. The artifacts should extensively give information about all the features of each subsystem. Documentation should be clear and versioned accordingly so that the changes in each release can be understood easily.

Continuous Support and maintenance

SCF is a large software with lot of federated servers, interfaces and Display systems. There is huge possibility of any bugs in the functioning of the system as expected.

Solution

A continuous support team should be employed to take care of any issues in the SCF. The SCF support team should have sound knowledge of all the technicalities of the SCF. They fix the bugs and ensure that the users of SCF are comfortable.

Security

As lot of servers and clients are involved in the architecture of SCF, security is a key issue.

Solution

1. Security firewalls are enabled on all the components of the SCF to prevent malicious hackers from intruding and stealing data.
2. All the messages are encrypted before sending and they are decrypted at the destination server.
3. To achieve this, carefully designed hashing algorithms are used which suits the project
4. Secure protocols are to be used when communication happens between various subsystems.

Deployable Subsystems

How will we upgrade the software of any subsystem of the SCF?

Solution

All the software installed on the subsystems should be enabled deployment on them. So, that new versions can be easily installed on it.

Continuous access to the servers requesting for same files

Sometimes same file will be requested continuously on the server. So the same request is piled up in the Queue of the server. This creates load on the server unnecessarily.

Solution

Local caching can be enabled on all the servers. Before processing the request, the file is searched in the local cache. If the file is found in the local cache, it is served instantly without making request. Efficient purge algorithms like Least Recently Used should be used to update the cache regularly.

Huge File Transfer

Sometimes it may be required to transfer huge files. For example, log files can be immense sometimes. It quite difficult to transfer large file over remoting.

Solution

The file uploading should be made distributed activity engaging many threads on it. Each thread transfer a chunk of the file so that file can be sent easily.

File Locking

There are cases when multiple clients access same file on the Repository and Collaboration Servers.

Solution

File locking is enabled. When a file is opened by any client, the file access is locked and other clients are denied accessing the file. The lock is release when the client releases access on the file.

Repository

Executive Summary

Due to immense increase in the use of software, it is common for software systems to require millions of lines of code for their implementation. The implications of the huge size and complexity are:

- Thousands of developers spread across large teams are required to complete development in reasonable time.
- Maintaining integrity and managing development is extremely difficult.
- Tracking the progress of each module of the project could become quite cumbersome.

To address this issue, we need to have a repository which uses some techniques to overcome these issues. Repository is a code warehouse which holds all the source code of the project which is collaborated by all the project members who may work in multiple locations. A repository enables all the project members to merge their code with the already existing code in a process oriented way which can be easily tracked and backtracked. The version controlling system enabled on repository helps to store every incremental change in the project with proper version numbers so that they can be obtained back in case of requirement. A file inserted into repository is called Check-in and a file extracted from repository is called Check-out. For every check-in, a metadata file is constructed which holds all the information about the checked-in file. Repository builds dependency relations between various files so that any version of the project can be retrieved without any mismatch.

The Concept and Key Architectural Ideas

The repository holds the central code baseline which is common unit for all the servers in the project. Multiple clients(Developers) from multiple locations can check in code and extract code from the Repository[3]. The repository enables large teams to coordinate and complete work in an organized fashion. Clients can query the repository for the files they want by using various type of filters. Repository is enabled such that multiple agents can work on it for various purposes[4]. The repository should be configurable to suit the business requirements. It has some core services which are not pluggable. The core services include association with the peers[5], updating, communication between channels, indexing etc. On top of these repository support many pluggable services like browsing the files, searching for files, building, packaging, analysis of the files, Status of the repository etc. The display of the repository should allow users to browse through the files to reach the file they want with minimum effort. Various mechanisms like namespaces and indexes are embraced to address this. All the above-mentioned ideas will be discussed in detail in the following sections.

Responsibilities

1. The primary goal of the repository is to enable insertion and extraction of source code into the project's baseline without any conflicts.
2. Authentication is done at the repository when a user tries to access the repository. The ownership policy decides whom to give permissions to check-in and check-out code from the repository.
3. All the dependencies and relations between files should be tracked and displayed elegantly.
4. A proper version-control system should be used so that any version of file can be traced back without loss of data.
5. It should offer efficient search. One should be able to retrieve few collection of files with common characteristics from tens of millions of files[6].

[3] Every module can't be checked in by every client. We will have ownership of modules so that only owner of module can check in the code. But anyone can extract any module of the repository and make changes to them. But they are forbidden to check-in the code.
[4] Sample purposes include Code analyzers, Test result notifier, Code viewer etc.
[5] Peers here are Test Harness server, Client Servers, Collaboration Servers etc.
[6] To do this efficiently, we use MongoDB to store the information of files. To fasten the search process, we use efficient search algorithms

6. Exploratory Querying is enabled on it. Clients query the repository for various purposes like to know the test results, to query source code, to query for a dependency graph etc. The Queries include querying on source files, log files as well a manifest file.
7. It should build a manifest file for every file that is checked in. The manifest file contains the information of all the dependencies of the file, its version number, the information about the Test Driver files corresponding to it, the path of the file etc.
8. It should have a build server integrated with it which pokes the repository for every certain time interval (may be 2 minutes) to see if there are any changes in the repository.
9. Quality assurance and maintenance should be ensured. It should support process of maintaining a massive collection of certified components.
10. It should be distributed. It should allow people from multiple locations to contribute as well as extract from the repository[7].
11. The status of the repository should be updated whenever there is any change in the repository.
12. The repository should have pluggable policies.
13. It should allow one or many agents to access it and do some processing on the files.
14. It should have efficient communications channels built with the Test Harness and Clients as they are frequently communicated.
15. Notifications of the Test Results should be sent to the Repository and corresponding clients for every Test Request executed by it.
16. It should be extendable and changeable.
17. It should take care of release management.
18. Rule based administration is required.

[7] We use Distributed File Management system to achieve this.

Organizing Principles

The construction of repository is governed by these key organizing principles.

Versioning

Why do we need Versioning?

There are multiple factors which contribute for the inclusion of version controlling in our Repository.

Collaboration

1. Its highly impossible for a team with large number of people to contribute for the same project/module without Version Control System.
2. If there is no VCS[8], only one person can work on a file at a moment. Only when a person has completed writing into a file, the other person can make his changes in it.
3. With the help of VCS, Multiple developers can work on the same file at the same moment. VCS will show the changes and the developers can merge their files accordingly.

Storing Versions

1. It’s always important to store a version of the project before advancing with the new changes. Versioning the files manually is not only tedious but also error prone.
2. We need a Version control system which automatically increments the previous version number and assigns a new aversion number to the newly checked in files.
3. Every checked-in file is kind of safe point which can be traced back in case of any error with the newly checked in files.
4. Also, it is extremely important to provide maintenance to the older versions of the projects which have been delivered to the clients.

[8] VCS- Version Control System

Understanding the nuances of the Project:

Every time a developer check-in a file, it will ask him to provide the description of the changes the developer made. Apart from that, it also shows the history of check-ins of that file. That will make the life of developer easy as it shows all the developments that has occurred the development of that module.

Backup

In case the Repository server crashes, we can retrieve the project from any one of the Team members computer. This prevents from accidental loss of the project.

How is Versioning achieved

To demonstrate how versioning is done, let us consider a case study:
The project consists of multiple systems. Each system has multiple packages. Each package has multiple files.
In the figure shown below, F stand for the actual files (All the bold boxes are actual files). Every other file is manifest file. M represents manifest file of module. P represents manifest[9] file of Package. S represents manifest file of system.

1. Before the addition of new version of file F2(F2.2), M2.1 depends on F1.3, F2.1 and F3.2.
2. It means that manifest file of Module has information that it depends on the files F1.3, F2.1 and F3.2.
3. When a new version of file F2 is created, the repository assigns a new version number to the file and stores it(F2.2)

[9] Manifests are XML files that define Systems, Programs, and Modules, simply by linking to lower level manifests and files. Files are shown with bold outline; manifests have a normal outline.

4. It clones all the previous manifest files for all the parent file of F2[10] and creates new manifest files with new version numbers.
5. Note that M2.2 and M2.1 refers to the same set of files except the newly checked in file.
6. It's up to the interest of the RI to choose which version of the file he wants to use.
7. Note that M1.2 is dotted box. It states that the developer has not decide to use the latest version of M1 yet.
8. RI for a module may link a new version of her manifest to any file or lower level manifest. The RI may NOT link a higher-level manifest to the new version. That is allowed only by the RI for the higher-level module.
9. The versioning of M1.2 is open – indicated by dashed lines – meaning that its RI may change links in that manifest without generating a new version.
10. However, M1.2 may not be checked-out for modification until its versioning is closed.

How older versions are accessed?

When a developer seeks an older version of the project, the corresponding metadata of the file is fetched. It has all the details of the lower level systems, packages and modules it depends on.
Note that the metadata of a file has the details of the corresponding versions of the lower level files it depends on. So, the exact image of the corresponding version can be traced back.

Advantages of the system

1. Developers can access complete configurations for older products that are still in service to provide support for customers.
2. A configuration can be easily rolled back should an earlier change prove to be incorrect or lead to other problems in the developing system.

[10] Parent file is the file which depend on the child file. The manifest of parent file has all the information of all its child files (The files on which it depends on)

Using Indexing to manage dependencies

Key Concept

Indexing helps to trace all the dependencies easily without sweeping through all the files of the repository.

Problem when there is no Indexing

If a developer requests for a version of the package, the repository will look through all the manifest files, finds the files with that version number and gives it back to the developer. This is tedious as the repository must read the manifest files of thousands of files.

Solution:

The following data is included in the manifest file:

1. Path of each lower level component and file
2. Set of keywords
3. Status information
4. Brief statement of function

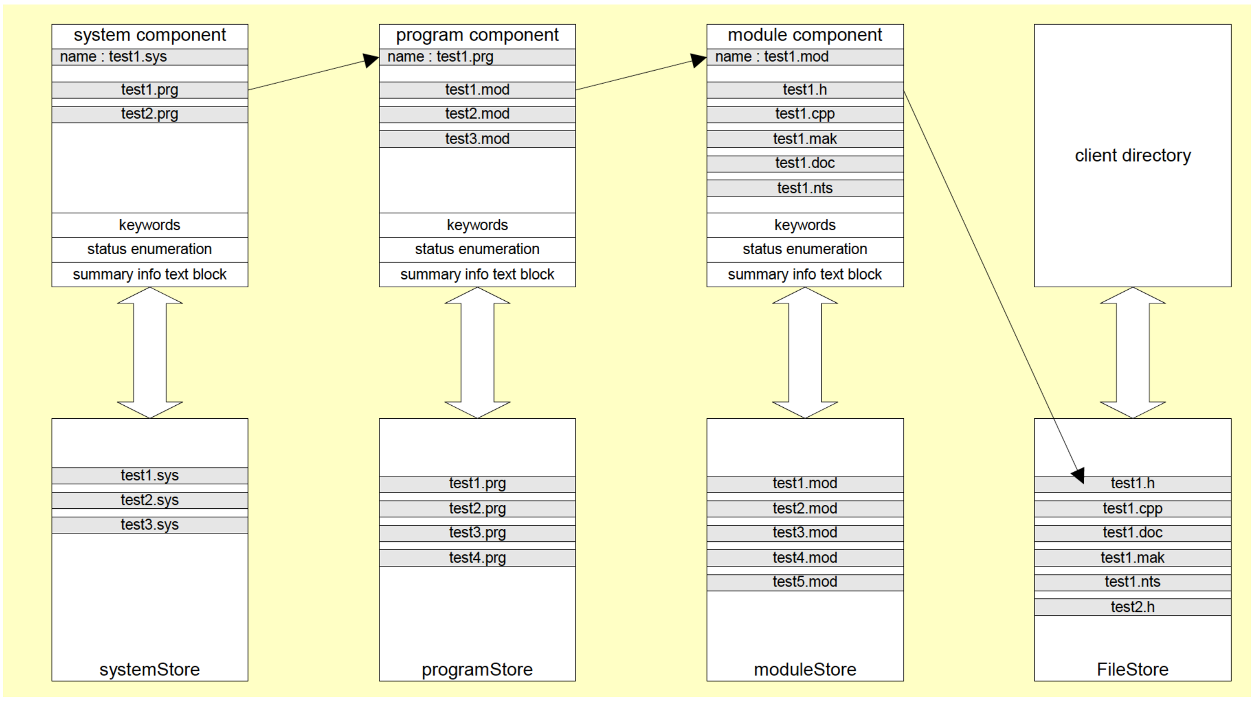


Figure 2 Indexing in Repository

The above diagram demonstrates how indexing helps to trace all the dependencies of a system.
For instance, developer requests for test1.sys, the manifest file has information of its lower level components like test1.prg, test2.prg etc. Similarly, test1.prg has all its lower level components associated with it. This helps to load all the dependencies associated to a project without any hassle.

We can use NoSQL databases like MongoDB to achieve indexing. The file name is stored as key in the MongoDB and all the associated information is stored as its value. For instance, a class file named ‘ServicePuller’ can be stored in the DB in the following format.

ServicePuller.1[11]:{
PathOfLowerLevelFiles:{

[11] 1 here indicates the version number of the file.

```
}
Keywords: [xxx,yyyy,zzzzzz],
Status:  “All Test cases passed”,
DriversPath: “/Documents/Project1/TestPackages.1.dll”
FilePath:”/Documents/Project1/Pacakge1/ServicePuller.1.dll”,
}
```

Check-in

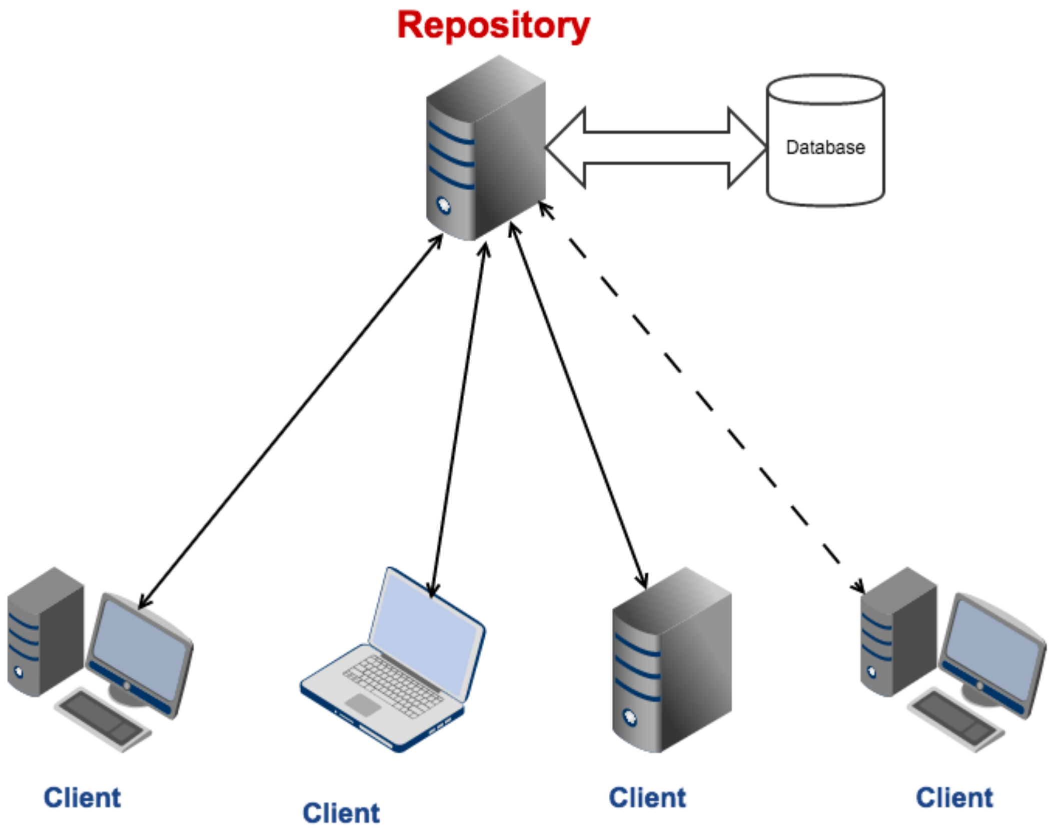


Figure 3 Check-in from multiple clients

The above figure demonstrates how multiple clients merges their code with the baseline. The dotted line for the last client signifies that there can be any number of clients like this who wants to check-in the code.

Check-in Process

1. When the Developer wants to check-in a file, he is authenticated first to verify if he is eligible to check-in that module.
2. If yes, he checks-in the code. Else, a notification is sent to him about the failure of check-in process and associated information with it.
3. The Developer is obligated to upload all the Test Drivers corresponding to the code while checking-in the code. If the files are being checked-in for the first time, he must upload the Drivers. Else, he can choose to update the Test drivers or allow the repository to bind the newly checked-in code with the old Test Drivers.
4. While checking-in the developer will be obligated to give a description of the changes made by him when compared with the older module. In that way, all the development progress of the module can be tracked by reading all the description histories of all the check-ins.
5. When the file is checked-in is successful, the repository will increment the version number according to the versioning policy and stores the file.
6. The meta data files created are bound to the corresponding dependent files.
7. Indexing file is updated for every check-in

Distributed File Management System

Consider the following case

- Number of developers in the Project: 3000
- Average software size: 5 Million lines of code

1. For example, when all the developer's start working at the same and starts browsing the repository, it is very difficult for the repository to cater the needs of all developers.
2. One of the design solution to handle this situation is using Distributed File Management system.
3. To lessen the load on repository, we must filter out the requested being sent it by some means.
4. Evert night Repository server sends all its indexes to each developer desktop.
5. Each index contains keywords, association paths, status information and a brief text summery of component.

6. When client wants to browse a file, it browses through entire repository structure without making frequent requests of the server.
7. The client gets the file location from the index search and clicks on the file link.
8. Only when file is not found on the local cache, the repository will receive a file request.
9. The cache on client is purged only when running low on disk spaces. It uses any of the purge algorithm to knock out the least recently file and replace it with the latest searched file.

This greatly reduces the number of hits on the repository server as client filters the requests before asking repository for the files.

Reusability

To decrease the effort in building software, it is very essential to use the existing modules. But it should be done in a process oriented way:

1. Reusability implies no change in documentation, code and test apparatus for the reused components.
2. Components which are reused are like black boxes which fit into the current requirement with 0 deviation.
3. To ensure that reusable components bring out the expected results, tight quality control schemes are to be employed.

Uses and Users

Developers

Developers use repository to collaborate their work to the project. Whenever developer checks in the code, it is available to all the project members. They can extract it from the repository and can work on it if required. It also helps the developers to safeguard their work against accidental crashes. If the working machine of the developer crashes accidentally, he can always get back the source code from the repository without any issues. Also, in case multiple developers are working on the same file, the developer can always update the latest copy of the file if takes it from the repository. The metadata of the repository gives the complete information of all the changes that have been made to the file in the process of its development. This helps the developer to merge his part of the file easily with that of the baseline.

Impact on Design

Version control is enabled on the repository to help the developers. They can understand the incremental progression of the development with the help of version control system. Also, they can retrieve back the older versions of the project in case they mess up with the latest copy of the project.

Managers

Managers use repository to get the progress of the development. They also use repository to track the activity of a developer. They understand the status of the current baseline by looking at the repository. They may also use repository to get the result of a test case or testing status of a module from the repository.

Impact on Design

Querying is enabled on the repository to search a file with any keyword.
For instance, Manager wants to view all the files which has test case with the name "Module1TestCase1", he can query on the repository using this string. All the file names which has this string is retrieved from the repository.
Efficient search techniques like indexing is used on metadata of files to speed up the querying process.

QA's

QAS's are concerned about the quality of code.They want to understand the metrics of the code without looking at the code directly.

Impact on Design

Various agents are employed on the repository which will analyze the code and pull out stats about the quality of code which can be viewed by the QA's directly.

Collaboration Server

Collaboration Server retrieves project status, essential documents from the repository to display them on the virtual display server.

Test Harness

The Test Harness server supports CTAI. Whenever a new package is inserted or an existing package has a new version checked into the Repository, the Test Harness runs a sequence of tests against the modified subsystem or entire project to see if anything breaks. Test Harness gets all the files of that build from the repository which are used for testing.

Impact on Design

Every time a new component is checked in, the developer must check in the test cases associated with it. Repository is enabled such that whenever it receives a new component, it is sent to Test Harness for testing. When an already existing component is checked in, the Testing is done against the existing Test cases.

Structure and Partitions

The Repository is organized into various packages which does a specific task. This also adds easy maintenance and provides abstraction. Also, the repository can be easily extended and enhanced in future.

The package diagram of Repository is:

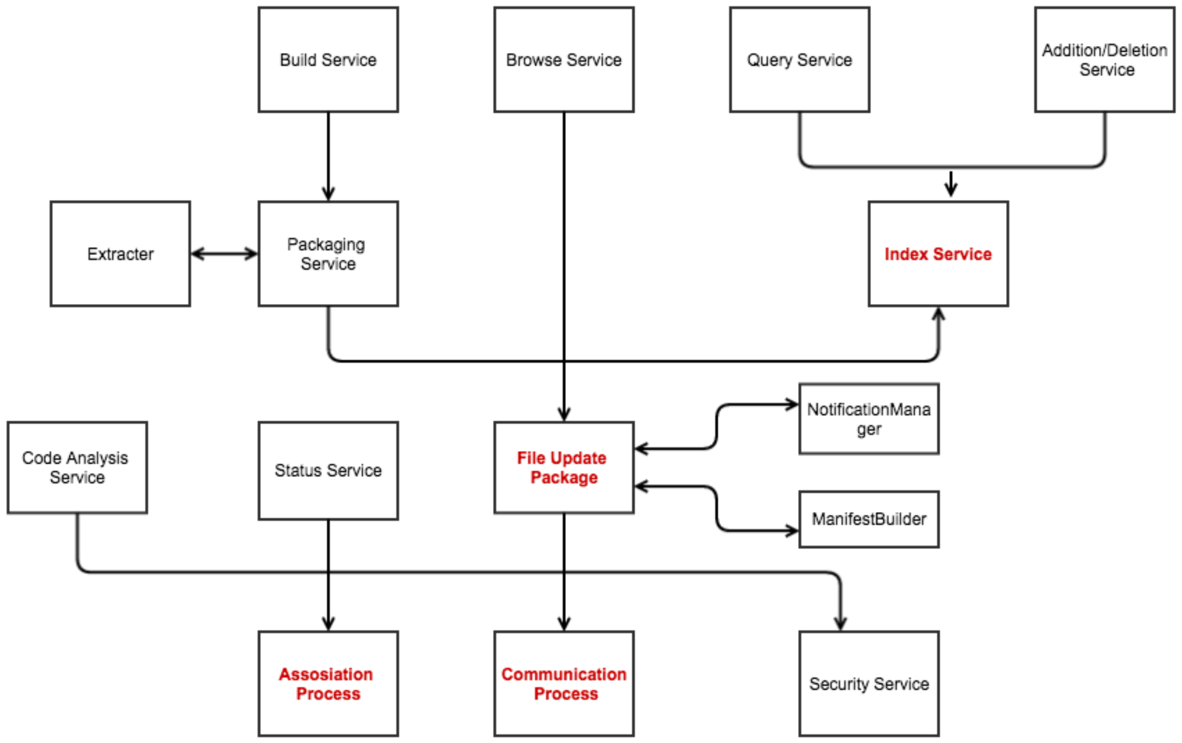


Figure 4 Package Diagram of Repository

All the Core Services are provided by the packages indicated with Red color.

Browse Service

This package is used for display purpose on the virtual display of the repository

1. It is used to display all selected components and their corresponding dependencies.
2. It is used to display the status information for the selected components.

Query Service

1. This service enables the repository to query on it.
2. Provides search services to find all components and files that match a set of key words or a name fragment.
3. Uses Index Service to speed up the process of searching the files based on the index for faster retrieval.

Index Service

1. This service is used to build search indexes when a new module/system/program is added to the repository.
2. This service is used by the File update process for retrieving the desired files fastly instead of sweeping through all the files.

Addition/Deletion Service

1. When a new file is checked in, this service sends the file to the file update process via a middleware.
2. When a new file is checked in, the indexes are updated accordingly by calling manage index process.
3. The same process is repeated for deletion of file also.
4. This service assists the Core Service (File update process) to perform the functions accordingly.

Package Service

1. When a new file is checked in , this package helps to build new package which is required for the build server.
2. Package Server uses index Server to get all the dependencies of the module to build the package.

Build Service

1. This package is the core component of the build server.
2. When a new file is checked in, this package service builds the project with the new module and package it to send it to the Test Harness for integration test.

3. Build Server builds the new package, catch the execution images and libraries needed for testing and deployment(Which is done by Test Harness)

Code Analysis Service

1. This package is a pluggable package which is used to analyze the code continuously to monitor the quality of the code.
2. Code analyzer run a whole set of processes on the code to calculate the metrics of the code such as Number of lines of code in the function, Naming conventions followed accordingly, dead code, uninitialized variables, memory leaks etc.
3. Apart from calculating metrics, code analyzers identify software security vulnerabilities of their modules.
4. They also do static analysis, defect detection, application security and IT risk management to ensure that every module of the project adhere to all the standards.

Association Process

1. This package is used to find all components and files associated with a higher-level component.
2. This package is used by Status Service to get all the lower level components of a component.

Status Service

1. Every file checked in has manifest file associated with it.
2. It has all the information about the content of the file, its association with other files,
   The test status of all the functions in the file.
3. Status service uses the services of Association service to extract all the lower level components of a component to send them to Test Harness for testing.
5. After all the modules are tested by Test Harness, the result status of all the modules is stored in the manifest file with the help of Status Service.
6. This package helps to modify the manifest file according to the test results from the Test Harness.

File Update Service

This package is used to accomplish three functionalities:

1. It extracts file from repository to the file cache on desktop if it is not present on the developer machine.
2. It is used to add a file from the repository service. It receives files from Add/Deletion Service.
3. When a developer checks-in new file, it is allocated a new version number and a notification is sent to the corresponding Developer by interacting with NotificationManager Package if the check-in is successful.
4. When a developer checks-out code, this package helps to extract all the code using dependency hierarchy and sends it to the developer machine who requested for the code.

Extractor

1. This package takes care of the check-out requests.
2. It interacts with packaging Service package which gets all the required files by interacting with Index Service package.

NotificationManager

This package is responsible to send notifications to respective servers in various cases

1. When a check-in is successful, the Developer is notified about it.
2. When a check-in is unsuccessful, an error message is sent to the Developer via Notification.
3. Notifications are sent to the Test Harness when a latest build is available for Testing.

ManifestBuilder

This package is used to build a manifest file when new file is checked-in by the developer.

1. The manifest file contains the Developer information, Test Driver information, Version number, all the dependency relations information etc.

Communication Service

1. A package dedicated for the communication channels between Repository-Client and Repository-Test Harness.
2. File sharing, message passing between the three servers (Repository, Client and Test Harness) is enabled with the help of this package.

Security Service

1. This package is used to authenticate people to access the repository.
2. When the code is being checked in by the client, this service authenticates if the developer is the owner of the module which is being checked in.
3. Provides tight security from external sources to hack the repository and steal the code.

Activities

Repository interacts with Client(Developer) and Test Harness frequently to accomplish all its responsibilities.

Client-Repository Activity

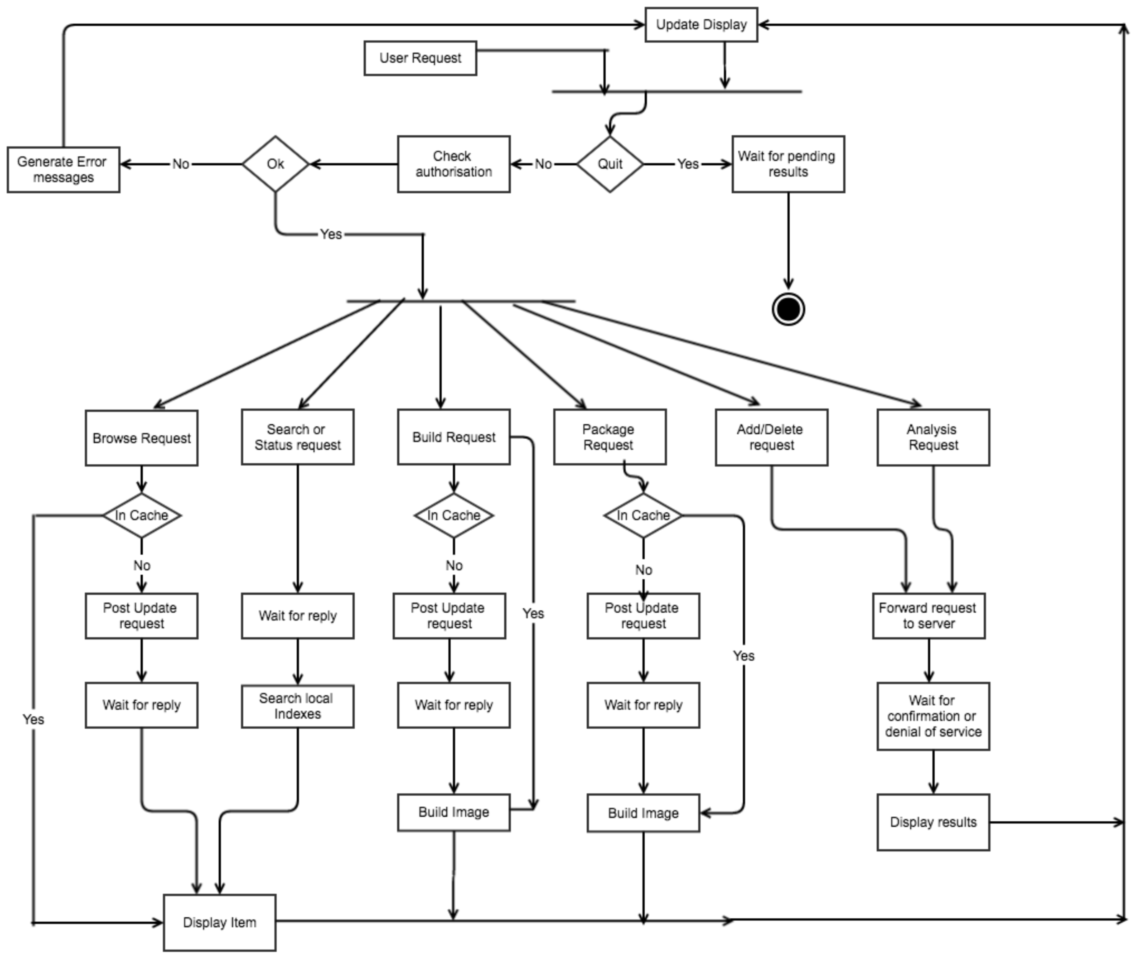


Figure 5 Activities between Client and Repository

1. When a client[12] sends a request to the repository, his authentication is verified by the repository.

[12] Client can be a developer/Manager/Lead/Architect

2. If the authentication is unsuccessful, corresponding error message is generated and sent to the client.
3. If authentication is successful, then each type of request does a specific thing.
4. If the request is browse request, the cache is first searched for the file.
   - If the file is found in the cache of the developer, the file is displayed.
   - Else, a request is made to repository.
5. If the request is of type Search/Status request, the local indexes are searched for the file and the file path is obtained.
6. If the request is of type Build request, the cache is searched to see all the dependencies are present in the local cache to obtain the build image
   - If yes, the build image is obtained.
   - If No, a request is made to the build service of the repository to get the latest build.
7. If the request is of type Package request, a request is made to the repository for package of the modules along with their dependencies.
8. If the request is of type Add/ Deletion request, the message is forwarded to the repository for authentication.
   - If the authentication is successful, the owner of the module is performing the operation, so file update request is sent to the repository.
   - The file update package assigns a corresponding version number to the file and builds the metadata of the file which holds all the information of the file.
   - If the authentication is unsuccessful, the request is denied.

Repository-Test Harness Activity

As a part of CTAI, Repository interacts with Test Harness continuously. The following activity diagram demonstrates the activity between Test Harness and Repository.

1. When a developer checks in a new version of the code into the repository, repository authenticates the developer.
2. If the authentication is successful, he is allowed to check-in the code. Else, appropriate error message is sent to the developer.

3. When the code is checked in, repository assigns the new file a version number, builds manifest file by getting info about all its dependencies.
4. Build Server constantly pokes repository for any change in any of the module.
5. If there is any new file checked in, Build Server builds the new image by getting all the dependencies of the module which is recently checked in And builds an execution image which contains all the files to be tested and the corresponding test cases associated with them.
6. The latest build is sent to Test Harness for the integration.
7. Test Harness performs whole set of integration tests after extracting the files from the image. The files contain code to be tested and the test cases corresponding to it.
8. The log results are sent to the repository and the Test Results are sent to the corresponding clients.
9. The manifest file of the module is updated with the test status that is obtained from the results of the Test Harness
10. If the Testing is successful, the code is deployed on the client machines automatically.

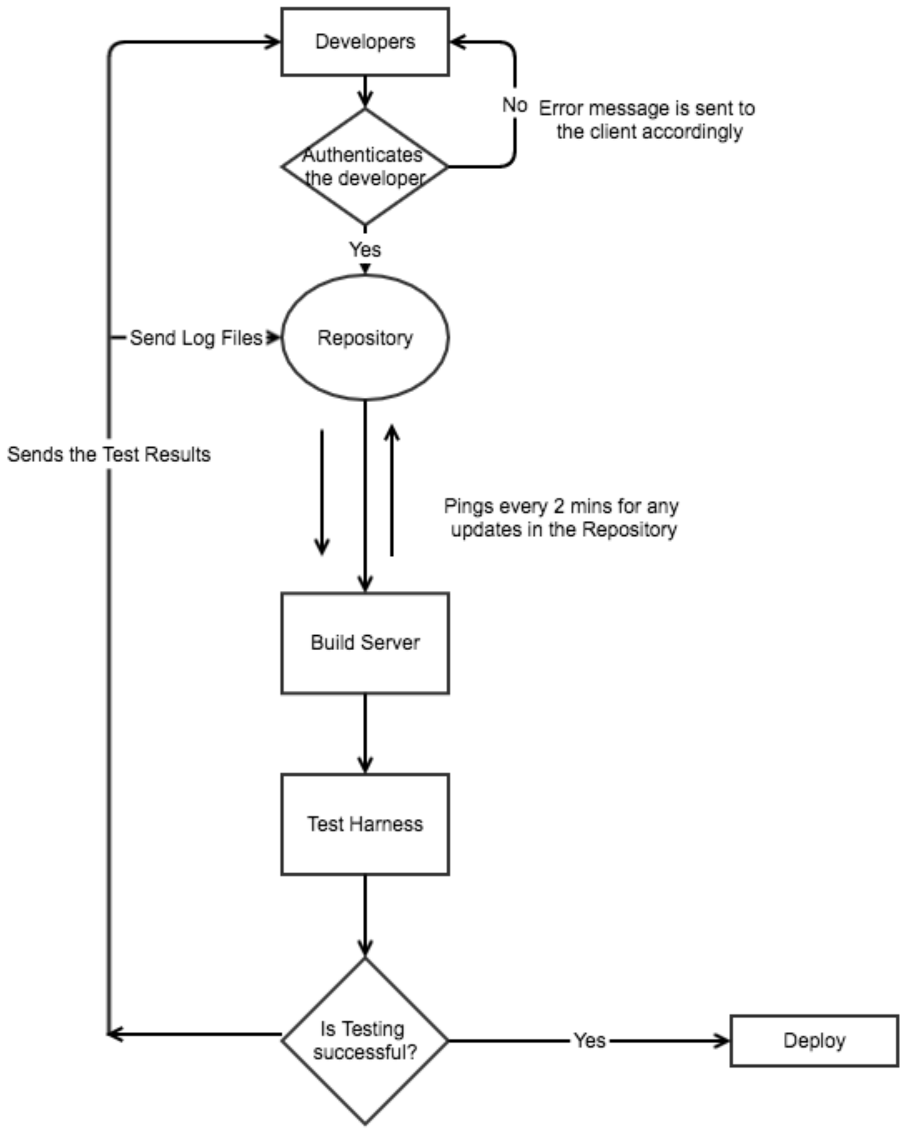

Developers
No
Error message is sent to the client accordingly
Authenticates the developer
Yes
Send Log Files
Repository
Sends the Test Results
Pings every 2 mins for any updates in the Repository
Build Server
Test Harness
Is Testing successful?
Yes
Deploy

Figure 6 Activities between Test Harness and Repository

Pluggable Policies

Installable Policies

To avoid chaos when developers check-in and test the code on repository, It always important to clearly give policy permissions on the repository access. We design repository to have tow servers: Project Server or Staging Server and Corporate Server or Production Server

1. Project Server is the server which is used for team's daily activates like check-in and testing. Developers have complete access on the Project Server. Developers can add components easily to the Project Server. The only constraint is that they should have passed Unit Tests.
2. Production Servers will make adding new components difficult. Tough review process is required to add components to it. Only when Staging Server is compliant with all the policies of the project, Code is checked in to Production Server. Build Server takes code from Production Server to build execution images to send to Test Harness.

Ownership

Ownership is a pluggable policy which is decided by the project owners how they want to use it.
Usually Single ownership policy is used by many projects

Single Ownership

1. A person who owns the module can check-in. No other project mate is allowed to check-in the code.
2. But other project members can extract code and work on it. But they are denied check-in permission.

Pluggable Ownership

1. We design a component called pluggable component which has all the policies.
2. When a new file is checked in, the pluggable policy component is scanned to verify if all the policies are being followed. If the verification is successful, it returns true and the code is checked in. Else, the developer is denied to check-in the code.

Versioning

It is up to the project owner how he wants to version the files which suits his business requirement. One of the used versioning policy is using tags.

Tagged Versioning

1. Every file has a tag associated with it which specifies a version number.
2. When a new file is checked in, it is not associated with any tag. So, a new tag is given with a brand-new version number.
3. When a new version of existing module is checked in, repository looks out for the latest check-in of the file. The version number of the tag of the file found is incremented and it is assigned as a tag to the newly checked in file.

Open Check-in

When a development of module takes a long period and the developer don't want to merge his module with the repository, he can choose to open check-in the module.

1. When any module is open checked in, the version number is assigned manually to that module by the developer. As long as the module is checked in to the repository, the developer can make any number of changes to that module. The version number is not changed for every change as it does when a version is checked in to the main repository.
2. Now the file is in open check-in state.
3. Once the check-in is closed, you cannot make any changes to the module with creating a new version of the module.
4. But, as long as the module is open no one is allowed to link to that module.

Critical Issues:

Concurrent File Access

Description:
When more than one developer tries to access the same file. One fix is to have file locking. But if the file is locked the developer tries to access the file, he is denied. If it is in between a process, the process shuts down because of this.

Design impacts:
Use the exception handling of the object-oriented languages like C# or Java

Solution:
The file access code is wrapped in a try block. When concurrent file is accessed by other developer, the exception is thrown. In the exception block, the thread which access the file is made to sleep for a while and reverted to access the file. When the developer who has the lock releases it, the second developer can get a chance to access the file.
Repository Server Performance
Description:
How can a single server cater the requirements of large community of developers? Obviously, the performance will be affected.
Solution:
Various mechanisms like Distributed file caching and indexing are being used to address this issue. Apart from that Distributed virtual servers are created which will be load balanced and are in synchronous with each other. A request from a developer is sent to the server with minimum load so that it can be processed fast.

Repository Security

Description:
We must ensure that repository is easily accessible to all the developers working from multiple remote locations. This triggers unnecessary access to malicious hackers and competitors who can get hold of the repository access. How are we going to prevent them from accessing the repository?

Design impacts:
We will provide role based administration and ownership of who can access and write into the files of repository.

Solution:
1. We can prevent hackers from stealing our data by enforcing strict firewalls on the top of servers.

2. We can provide strong conventional network security and allowing outside access only thorough virtual private networks. This will not allow hackers from outside networks to hack the repository.
3. Only module owners can change their modules. Other developers are granted read permissions, but they are denied write permissions.
4. We can provide read access only through proprietary reader software.

Ease of Use

Repository should allow easy ways to interact with it as well provide stable functionalities which can be easily tracked. Every minute information should be easily fetchable so that any issue can be resolved easily. It should offer efficient search techniques for the managers to query any part of it.

Solution

Sophisticated GUI is developed which will make repository easily browsable.
To make the search faster efficient techniques like indexing, storing files in MongoDB are embraced.

File-Loss

Developers may accidentally delete file from the repository accidentally while interacting with the Repository. It creates a great problem to retrieve back the files.

Solution

1. The Developers are denied the Delete permissions. Only administrators are given permissions to delete files.
2. The code is continuously updated to a Cloud Server (May be Amazon AWS) so that it can retrieved back if the files gets deleted on the server accidentally.

Agents

Repository is not merely a code warehouse which stores the code. Beyond that, many agents can be employed on the repository which analyses a set of things and come out with some valid analysis. All the agents are pluggable and configurable depending upon the project requirement. Sometimes we don't want an agent to work on the code. So, the agents should be enabled/disabled instantly. Every agent is desiccated for a single purpose. No agent depends on the result of the other agent so that they are independent. Some agents may start automatically and some may require manual invocation.

Test results notifier

This agent will notify about the Test results to all the people concerned with the testing activity like the Test Developer, Tester, Manager and the architect.

Dependency builder

For every check-in of a file, a manifest file must be built. The manifest file contains all the dependency graph of the file. This agent will build the dependency graph by making type based analysis.

Check-in identifier

1. It can be used to get all the check-ins done by a developer on a filter basis.
2. It reports all the open check-ins to the developer who has done it and his manager when the check-ins are open for considerably lot of time.

Extractor

This agent extracts all the dependency of a file with the help of the root node. It recursively searches for all the dependencies of a file.

Viewer

This agent displays the required information of the manifest file. For example, the developer always wants to view the dependency graph of the file, It is showed to him. It is configurable and some other developer may want to view some other information of the manifest file.

Test Harness

Executive Summary

Every code written is to be tested appropriately to make it deliverable to the client. So, it’s always challenging for any company to test their code in all possible ways to the maximum extent. To ensure the robustness of large systems, various testing procedures are employed like Unit Testing, Performance Testing, Load Testing etc. But one of the complex test is an Integration Testing. Though every module works fine, it’s very important that all the modules when integrated should work as intended. For any change in any module of the project, the whole project must undergo integration testing as any small change may break other modules. So, Integration testing is an expensive process as it involves lot of man hours when done manually. We need to automate this process as it can be repeated as many number of times for every minute change in the code. Apart from that, we want to reflect the changes on the client machines instantly after the newly made changes are tested thoroughly.

Test Harness is the important part of Software Collaboration Federation which supports Continuous Integration, Testing and Deployment. Whenever a new package is inserted or an existing package has a new version checked into the Repository the Test Harness runs a sequence of tests against the modified subsystem or entire project to see if anything breaks. If all tests are passed, the latest code is deployed on the client machines automatically by Test Harness on client machines.

Test Harness and its uses

Test Harness is an automated testing tool which will ensure that the software is stable after subsequent changes are made to any part of the code. Test harness can be used to do any type of testing like Integration, Performance, Load Testing, Regression Testing. Before code is given to Test Harness, Developers must ensure that the code has passed unit testing. Thus, continuously testing the code for any subsequent change will ensure the robustness and reliability of the code.

Brief Architecture

Test Harness will receive requests continuously from Repository for every piece of code that is checked in. Many developers will commit code to the repository and each change in the repository triggers a new build of the module which is checked in. The build contains all the modules that depend on the newly checked in module. Apart from that multiple clients may want to test their code before checking it to the repository. So, Test Harness will receive a glut of Test Requests from both Repository and Developers. All the Test Requests are queued in a blocking queue. Multiple threads will be spawned from the Test Harness Core. Each thread will create separate Child Application Domains for each Test Request and the tests are executed in Child Application Domains. The results are stored in the Log repository and sent to the clients also.

Key Issues

There are some issues which should be considered while building Test Harness.

1. Performance
2. Ease of use
3. Demonstration
4. Security
5. Flexibility and Extendibility
6. Logging and Intimating clients about the Test Results
7. Cost of Implementation

More critical issues are discussed in detail in the later sections of the document.

Introduction

Almost every software project has hundreds of packages and millions of lines of code developed by many developers. This scale may even go high for Huge projects where developers across the world contribute to the project. Every developer will test his/her own modules and ensures that they are working fine.

But the most complex task is the Integration of all the modules. Even though the developer might be quite confident about his module, he could not ensure the impact of his change when integrated with the Baseline code. So, every time a change is made to any piece of code, an integration testing[13] is required.

Test Harness is a tool which automates the integration testing so that the tests can be performed any number of times for any change in any part of the code.

The Concept and Architectural Ideas

Test Harness is a remote server which can serve multiple clients as well as the repository. One or more clients or Repository can supply Test Requests to the Test Harness. Client must send Test Requests, Test drivers and the Code to be tested in the form of dlls to the Repository server before sending the Test Request. The Test Request will have the information of the Test developer and the dlls that are to be executed. If the repository wants to perform integration test on a module which is checked in, the build server sends a build file (Which contains all the code dlls and Testcase dlls for testing). All the Test Requests from multiple clients and repository are enqueued in a blocking queue by the Test Harness. Multiple threads are spawned from the Test Harness. Each thread deques a Test Request form the blocking queue and creates a Child Application Domain to process it. This will ensure that multiple Test Requests are processed concurrently. After the execution, the test results are stored with an identity (Developer's name and Time stamp) in the repository and they are sent to the client who has sent the corresponding Test Request also. As all the Test Results are stored in the repository, they can be accessed any time in future.

In detail explanation of each functionality will be provided in the following sections

[13] *Integration Testing:*
An Integration Testing is a process in which a few or all Program units are tested to ensure that all the components work fine when integrated together.

Responsibilities

To achieve the goal of Test Harness, the following obligations are to be met:

1. Test Harness should accept one or more Test Requests from client in the form of XMLs and from the repository in the form of execution images to test the code.
2. The XMLs from the client contains the file names of the Code to be tested and the associated test drivers. It contains the developer name and the corresponding version number of the file.
3. The Build image from the build server of the Repository contains all the files needed for testing.
4. Each test driver should implement a common interface which differentiates it from the Code to be tested.
5. Implement a Blocking Queue which will enqueue the Test Requests sent by the clients and repository.
6. Implementing XML Parser to parse the Test Requests sent by the client.
7. Use Application Domains to isolate the execution of a Test Request. Each Test Request runs on its own Child Application Domain.
8. Implement asynchronous message-passing communication channels between Clients and Repository server, Client and Test Harness Server, Test Harness Server and Repository server using Windows Communication Foundation(WCF).
9. After the execution, the Test results are sent to the requesting client and log files are stored in the Repository server with a distinct identity (Probably a combination of timestamp, Developer identity and Version number associated with it).

Organizing Principles

For extending the application in future and maintainability, Test Harness is divided into various packages.

Users and Uses

Developer

1.Uses Test Harness to test his code against a set of prebuilt test cases befor check-in the code.
2.Uses Test Harness to verify code as a part of check-in process and defect analysis.

Impact on Design

1. Test Harness will accept the Test Requests from clients and parse them to process the request.
2. To serve hundreds of developers, multi-threading is enabled which processes the concurrent requests efficiently.
3. Developers are given a Virtual Display Client to give input Test Requests to the Test Harness

Quality Assurance

QA personal focuses on the testing process in preparation for acceptance testing and on maintaining code quality. They perform various types of testing on the certified code like performance testing, load testing, stress testing etc.

Impact on Design

QA will be submitting test request messages for large bodies of code. It is important that the collection of dependent code packages be automated and that the Test Harness operates as efficiently as possible. It is also important that we should be able to scale out by adding new Test Harness servers without a lot of rework to the existing Federation Infrastructure so that QA's can perform bulk testing in reasonable amount of time.

Repository

Repository is one the prime user of the Test Harness. As a part of CTAI, for every check-in that happens in Repository, Test Harness is used to ensure that the newly checked in module did not bring any inadvertent change in the functioning of the system.

Impact on Design:

A Build Server is enabled on Repository which constantly looks for any update in the repository. When a new file has been checked in, the build server builds the execution image and sends it to Test Harness for testing and deployment.

Partitions

For extending the application in future and maintainability, Test Harness is divided into various packages.

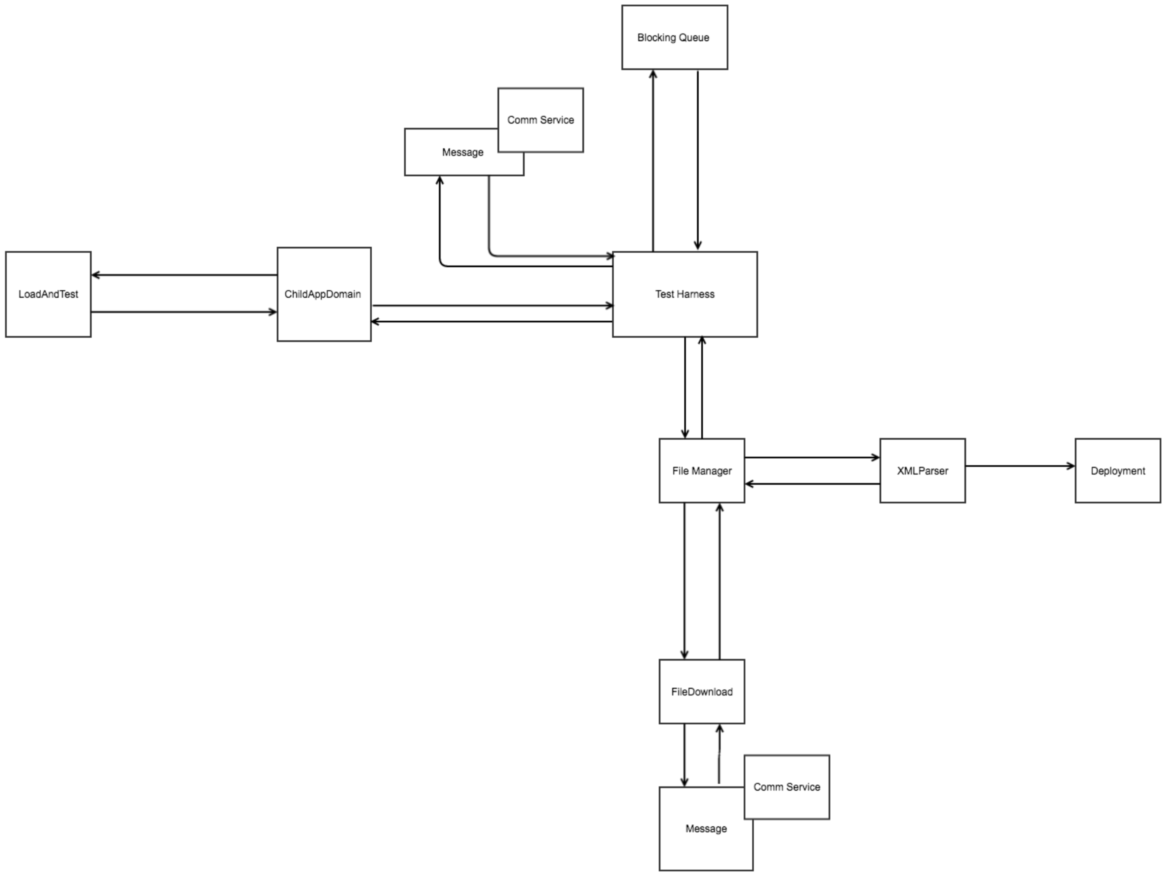


Figure 7 Test Harness Package Diagram

TestHarness

1. This is the core package which manages all the packages and gets all the work done.
2. It receives all the Test Requests and pushes them to the Shared Blocking Queue.
3. A Child thread deques a Test Request and send it to File Manager package.
4. It received test results from the ChildAppDomian package after processing the TestRequest.
5. A distinct log file which contains the test results is created and sent to the repository via WCF channel.
6. The test results are sent to the corresponding client.

Blocking Queue

1. It is a shared Blocking Queue which provides a mechanism to enqueue all the messages coming from multiple clients.
2. The enqueue operation adds the message to the queue when the test harness calls the Blocking Queue package.
3. Every time a message is dequeued from the BlockingQueue and processing of the request starts.
4. This process is repeated as long as there are messages in the BlockingQueue. It uses efficient locking mechanism as multiple threads will access it.
5. If the Test Harness tries to dequeue a message from an empty BlockingQueue, it will be blocked till a message is enqueued.
6. This builds an efficient waiting mechanism

File Manager

1. The File Manager package interacts with XMLParser package and parses the message to obtain the required file information.
2. It gets the list of all the files required for testing.
3. All the required Data Structures which holds all the information of the files required for testing are built.
4. A temporary folder which is unique for a Test Request is created.

5. The local cache is searched for the files. If the files are found in the local cache, they are copied to the temporary directory. Else a File Download request is sent to the FileDownload package.

FileDownload

1. It sends a message which contains the data structures of all files required for the Test Harnessfor testing.
2. Using WCF, the files are downloaded from the repository into the temporary folder created by this package.
3. In case the dlls are not found in the Repository Server, Test Harness is notified about it with a proper message.

ChildAppDomain

1. TestHarnesss sends the details of the temporary folder created (It contains all the files required for testing) to the ChildAppDomain package.
2. The ChildAppDomain package creates a new AppDomain for every TestRequest to provide isolation.
3. In case any exception in executing a TestRequest, it won't affect the Test Harness as every TestRequest is processed in separate ChildAppDomains.
4. This package interacts with LoadAndExecute Package where the TestDriver code is executed.

LoadAndTest

1. The package is responsible for loading the dynamic link libraries for each of the test drivers and their test codes.
2. Every file is scanned to know which dll implements ITest interface (A common interface which is to be implemented by every test driver).
3. It identifies the TestDriver code, runs it and results are given to the ChildAppDomian which calls it.

Message

1. Every communication between any two servers should comply with a clear message structure which is understandable by the opposite server.
2. This package takes care of building the message (In XML format probably) which is sent to the other server via a WCF channel.

CommService

1. The CommService package contains commservice class which implements ICommunication interface.
2. All the services has a CommService package which does similar things so that a common contract is present between the servers for communication.
3. This package is essential for communication between any two servers[14]

Deployment

1. This package is responsible for all Deployment activities.
2. This package deploys the certified code on the customer machines automatically once the testing is successful

Activity

The following activity diagram demonstrates the various activities that occur in Test Harness

[14] The Servers here include Test Harness Server, Repository Server and Client Servers

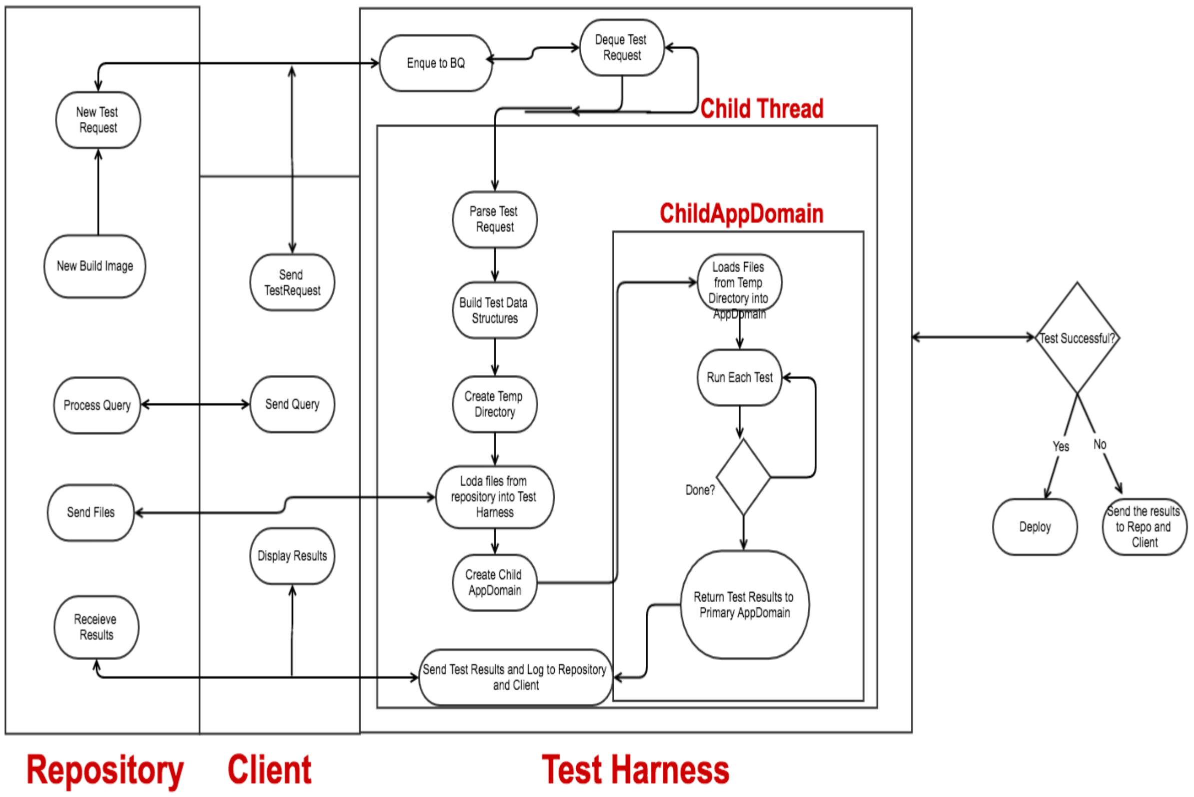


Figure 8 Test Harness Activity Diagram

1. Test Harness receives a Test Requests from multiple concurrent Clients or Repository.
    - If a client wants to test a module before checkin, he will build a test request (probably in the form of XML) which contains the details of Code to be tested, Test driver details, Developer's identity which can be parsable.
    - When a new module is checked in, Build Server builds an execution image which contains the module which is to be tested and all the corresponding dependency files which are obtained from the dependency graph of that module. The Build image is sent to Test Harness for testing.
2. All the Test Requests from Clients/ Repository are enqueued in a shared blocking queue.

3. A set of threads always await to pull a request from the blocking Queue. It is called Blocking Queue because it is always blocked as long as it receives a Test Request.
4. A Test Request is dequed from the Blocking Queue by one of the child thread of Test Harness.
5. The child thread parses the Test Request it receives using an XML Parser.
6. All the Test Data structures required to send a request to the repository are built. The Data structures hold the information of required files which are required for testing.
7. A Temporary directory with an unique identity(A combination of Test Developer name, timestamp and the corresponding version number of the file) is created in one of the Test Harness directories.
8. Test Harness checks in the local cache if it contains the files required for testing. If they are present, they are coped to the temporary directory. If not, Test Harness makes request to Test Harness to get the required files.
9. Repository processes the request and uploads all the files to the temporary created directory of Test Harness.
10. A new Child AppDomain[15] is created which is isolated from the main AppDomain and processes the current request.
11. All the files from the temporary directory are loaded into the child AppDomain.
12. All the dll files are scanned to obtain the files which contains testcases (All the test cases implement a common interface; This fundamental idea is used to obtain the files which contain the Test Cases).
13. All the tests are run one by one till all the tests of a Test Request are processed.
14. The Child AppDomain sends the Test Results to the primary domain after the completion of execution[16]. The Child AppDomain is unloaded after the completion of execution.

[15] A new AppDomain is created for every Test Request to ensure the isolation of a Test Request. If a Test Request if processed in separated AppDomain, only it is affected if any exception occurs when processing it. The whole system would not blow up if this approach is used.

[16] Main AppDomain and Child AppDomain implement MarshalByRef to enable communication between them.

15. The main AppDomain builds a log file which is unique (A combination of Test Developer name, timestamp and the corresponding version number of the file) and uploads it to the Repository.
16. The Test Results are sent to the corresponding client for this reference.
17. The Child thread is blocked again to receive a Test Request from the Blocking Queue.
18. All the Message passing and File upload/download service between all the Servers (Test Harness, Client and Repository) are done via Windows Communication Foundation.
19. Finally, if all the tests are passed, the latest code are deployed on the customer machines so that the changes are reflected on their machines.

In this way, Concurrent clients and repository are catered as efficiently as possible using multi-threading.

Critical Issues

GUI Design may be complex:

GUI Design may be complex due to number of messages that must be processed.

Solution:
We use tabbed display with tabs for connecting and for each type of message sent. Each message tab has a message creator method that the tab calls.

Graceful shutdown of multiple processing threads

The threads which are spawned by Test Harness must shut down gracefully for the efficient functioning of Test Harness.

Solution:
We will have each thread re-enqueue quit message before exiting. So, all threads will shut down. We will have to keep a list of active threads to wait on after sending quit message.

Large Testing Loads

It’s crucial how we manage huge glut of Testing Requests

Solution:
We can add additional Test Harness servers, as each Test Request is independent of other. We will need to keep a list of all running servers and their endpoints that can be read by all clients and repository so clients can find a newly added server.

Testing the Tester

Automated testing of asynchronous systems like Test Harness can be difficult, since the outputs are not always generated in the same order.

Solution:
Develop a set of test messages that are rich enough to exercise all the System’s functionality. Log responses at every step of processing in a list of responses. With a validated system or by analyzing the TH Code, record the responses and store for later comparisons. When comparing just check to see if a response is in the list and, at the end, every list response has been checked.

Security

As Test Harness is used by various type of users, security is also a key issue. A developer may not be permitted to view the Test Results of another developer. So, setting up roles and permissions is quite necessary.

Solution:
Separate GUIs for developers, Managers and QA’s need to be provided and a restricted access is given to Repository Server. Developers and managers working on certain modules are given a Folder access in the Repository Server. They can’t access other projects repositories.

Performance

The performance is a critical issue which needs to be considered when designing the Test Harness. As the number of packages to be tested increases, the performance of Test Harness is hampered. Especially, QA Analysts sometimes want to check end to end functionality of the project. This will make Test Harness rather busy.

Solution:

To address this issue, we have embraced multiple techniques like using Blocking Queue and multi-threading are used. If the Test Harness tries to dequeue a message from an empty BlockingQueue, it will be blocked till a message is enqueued. This builds an efficient waiting mechanism. Also, each Test Request is processed on separate child thread so that multiple Test Requests are processed concurrently on multiple threads.

Caching

There might be cases when a module has to be tested continuously for some testing purpose. In that case, the same request has to be the repository unnecessarily.

Solution:

Local cache is maintained in Test Harness. If a file is downloaded from repository, it is stored in the local cache. If the cache is full, efficient purge algorithms to knock out the least recently files are employed. In case a file is required by the client continuously, it is searched from the cache before making a request to the repository.

Client

Executive Summary

Client is any team member who is part of software development and who uses SCF for developing software. Clients can be of various roles. They may be Developers. Managers, Architects, QA's etc. All the clients will collaborate their work using the services of the SCF.

Clients are part of the connected network of all the team members of the project. They check-in code into the repository so that it will be available for the team. They submit their time sheets and can extract documents from the Collaboration Server for various activities. They can communicate with each other with the help of Virtual Display Client installed on their machines. They can chat, video conference and exchange information with other client using various sophisticated mechanism.

Organizing Principles

1. There are people with different types of roles who use Clients. They might Developers, Managers, QA's, Architects etc.
2. As a client, can be any person who is one of the part of SCF, it should be highly customizable in order to accommodate an new type of client in its architecture.
3. There should be some core activities of each client like having a Virtual Display Client, and communicating with the Repository, Collaboration Server and Test Harness.
4. Apart from that is should allow to configure ad-ons in its architecture to support the corresponding clients. For example, it should allow managers to query on it and It should allow QA's to fetch the metrics of the project from the repository.
5. Clients are designed in such a way that they are well embedded in the SCF hierarchy. They should be able to communicate with the peer clients. The communication includes chatting, setting up meetings, Video conferencing etc.

6. They should be able to publish and subscribe to some other client events. For example, a developer client should be able to receive notifications from the Manager client whenever he chooses to do so.
7. A client is designed in such a way that is receives index files every day from the repository. The new index file should be automatically replaced by the old index file.
8. Client software should be easily installed on bulk number of machines. The software should support all the platforms (Windows, Mac and Linux etc.).
9. Any update in the client software should be easily deployed all the client machines instantly and without any issues.
10. The installation of client should be clearly abstracted from the internal technicalities of SCF. Probably, the client can be installed by just clicking on the exe file. The exe file differs on the type of the client so that various types of clients are installed depending upon the requirement.

Responsibilities

1. Each client should provide a Graphical User Interface through which it sends the input request and receives results asynchronously.
2. Developer client should be able to check-in code, checkout code, extract documents form Collaboration server, upload documents to Collaboration server etc. It should display the milestones of the developer based on filters (like daily milestones, weekly, quarterly etc.).
3. Manager client should be able to query the repository, get the log files from the repository, to set up meetings with the peer clients.
4. QA clients should be able to get the metrics of the project and perform various types of quality assurance tests on the source code of the project.
5. All the clients should have restricted[17] access to the collaboration Server so that they can download/upload documents from/to the collaboration server.

[17] Access should be restricted because every role can't access some documents. For example, developers should not have access to the business level documents as they are confidential.

Structure

For the ease of extendibility and maintainability, Client is organized into following packages.

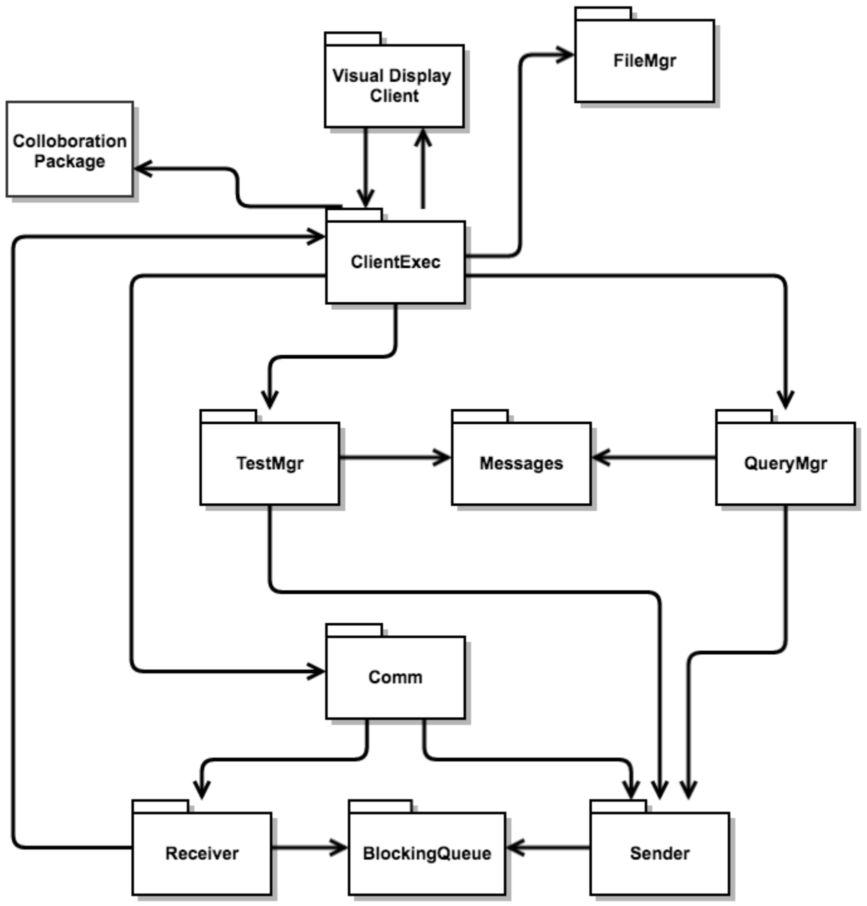


Figure 9 Clinet Package Diagram

ClientExec

1. The Core package of the Client which gets the work done from the other packages.
2. It will interact with TestMgr package to build a TestRequest which contains all the dll files information.

3. It interacts with QueryMgr package to build a query request which is sent to the repository.
4. It interacts with Collaboration package to get documents from the Collaboration Server and upload to it.
5. It interacts with VisualDisplayClient package to get the required information such as the Milestones of the developer, project related visual tools like pie charts, bar graphs etc.
6. It interacts with Communication package to send and receive files from the repository and TestHarness.

FileMgr

1. This package is responsible for allowing code for Check-in into the repository.
2. This package is responsible for extracting code from the repository.
3. The package is used by the client to browse the repository for files without extracting them. All the dependencies related to that module are displayed on the client GUI elegantly for the developer to understand the module.
4. This package is responsible for extracting the log files from the repository.

ColloborationPackage

1. This package is responsible for communicating with the Collaboration Server.
2. All the documents such as time sheets and Design level documents created by the Client are uploaded into the Collaboration Server by this package.
3. All the Project related documents like Project status, schedules, work packages are downloaded from the collaboration server by this package.

VisualDisplayClient

1. This package provides a GUI for the client.
2. It interacts with the ColloborationPackage to display all the documents required. The documents include Time sheets which are filled by the Client and uploaded back to the ColloborationServer, project related documents etc.
3. It shows the developer his milestones, the status of the project etc on the GUI.

4. This package is responsible for the client to view the code from the repository.
5. All the GUI related activities are performed by this package.

Message

1. This is a package which is utilized by TestMgr and QueryMgr to frame the message. The message body included ToUrl, FromUrl and the message body. The message body depends if the request is of type query or TestRequest.

TestMgr

1. This package is responsible for constructing the required TestRequest.xml which has information about all the test drivers and the code to be tested dlls.
2. It interacts with Message package to build the message in a predefined format.

QueryMgr

1. This package is helps the Client to query on the repository.
2. The type of query depends on the client. The queries may be the status if the module, the metrics of a module, the status of the volatile packages etc.

3. For a developer, this package helps to query the status of a test case. For a manager, this package helps to understand the status of the baseline and the status of the progress of the development. For a
4. This package is uses the Message package to bundle the query in a format and send it to the repository.

Comm

1. All the communication between Client and various servers is taken care by this package.

2. It implements an interface which is a common contract which defines Service and operation contracts that helps the client to interact with other services.
3. The contract includes both file streaming and message passing to the remote servers.

Receiver and sender

1. Both the packages are used for receiving and sending messages.
2. All the messages received by the client are stored in the BlockingQueue.
3. Each message is dequed by any of the threads to process it.

Blocking Queue

Blocking Queue functionality is similar to the one used in Test Harness.

Activities

The Activity diagram shown below defines the series of actions that happen for various events in the client.

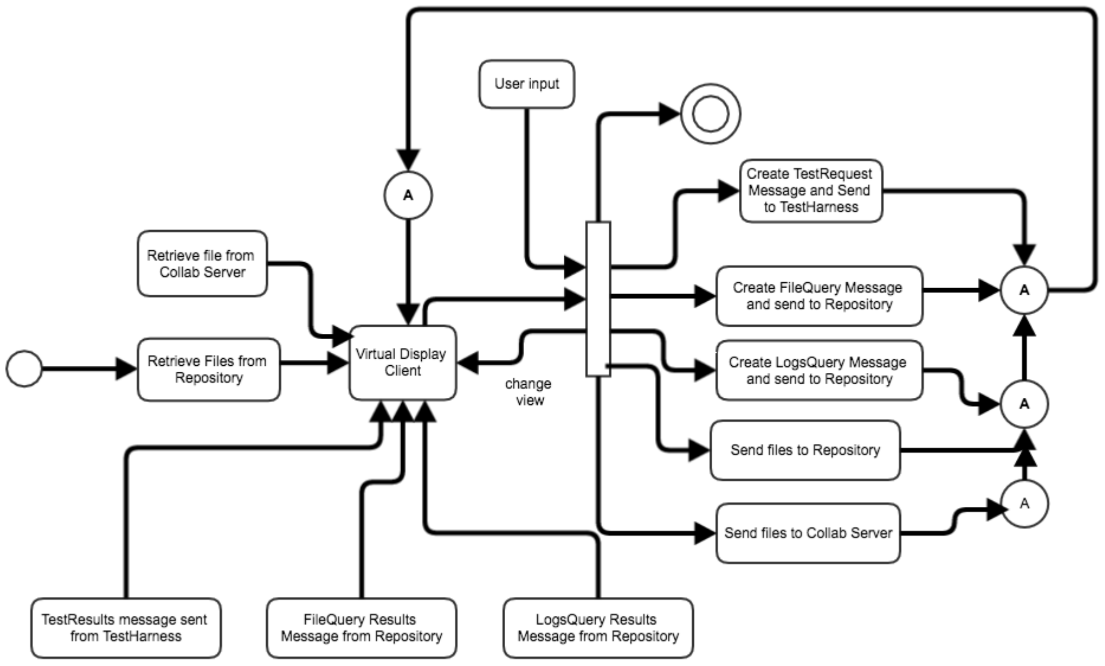

User input
A
Retrieve file from Collab Server
Retrieve Files from Repository
Virtual Display Client
change view
Create TestRequest Message and Send to TestHarness
Create FileQuery Message and send to Repository
Create LogsQuery Message and send to Repository
Send files to Repository
Send files to Collab Server
TestResults message sent from TestHarness
FileQuery Results Message from Repository
LogsQuery Results Message from Repository

Figure 10 Client Activity Diagram

1. The rectangular bar shoes the type of events that diverge from the various actions taken by the Client from the Virtual Display Client.
2. Various type of requests made by Client are:
   1. **Create TestRequest Message and Send to TestHarness**: When the developer wants to perform testing on a module, he wraps all the source code which is to be tested and the TestDrivers which contains the testcases using message package and TestMgr package and sends it to TestHarness. Before sending the TestRequest to TestHarness, developer uploads all the required files to the repository.
   2. **Create FileQuery Message and send it to Repository**: When a developer or manager wants to query the Repository for a file, the local cache is searched for the file. If the file is not present in the cache, the local indexing is searched for the information of the file and a request is made to the repository in the form of an XML with the help of Message package and FileMgr package.

  3. **Create LogQuery Message and send it to Repository**: When a developer or manager wants to query the Repository for a result of testcase or any query on log files, he sends a request in the form of an XML with the help of Message package and FileMgr package and sends it to the repository.
  4. **Send Files to Repository**: When a developer wants to check-in a file into the repository.
  5. **Send Files to Collab Server**: When a Client wants to receive documents such as Work packages, Design documents, work schedules and all the project related documents from the Collaboration Server, he makes a query to it requesting the same.

3. The Client receive message/files in the following context.
    - It receives the test results for every TestRequest it has sent to the TestHarness.
    - It retrieves file/files as a response for the FileQuery message it has sent.
    - It retrieves file/files as a response for the LogQuery message it has sent requesting for the log files which contains the Query string.
    - It receives files as a Check-out process from the repository.
    - It receives all the project related documents, work packages, schedules. Design documents from the Collaboration Server.
4. All the results are displayed on the Visual Display Client interface to make it understandable for the clients.
5. All the Messages/Requests are sent to the corresponding servers (Test Harnessand Repository) via Asynchronous WCF communication. As all the messages are sent asynchronously , the view of the Display is changed as soon as the request is made.

Uses and Users

Client is a common name for many types of users. The users may be Developers, Managers, QA's etc. Client acts according to the type of user.

Developer

1. He uses client to Check-in code so that his work is merged to the baseline and available for the team.
2. He uses client to test his module if it can be successfully integrated with its dependencies by sending a TestRequest to the TestHarness.
3. He uses client to browse files from the repository.
4. He uses client to interact with the team using chats, webcams and all visual display tools remotely to work along with the team.
5. He uses client to download work packages, understand his milestones and the status of the project.

Impact on Design:

1. Visual Display Client is customized according to the developer's requirements. The display elegantly displays all the activities he does and all the results he receives.

Managers

1. Managers use client to query the repository for log results and to understand the progress of the project by looking at the baseline of the repository.
2. He uses client to know about all the volatile packages in the repository.
3. He uses client to notify all the developers regarding an event or meeting.

Impact on Design:

1. Virtual Display Client is customized to easily query the repository, notify him about the status of every integration test that happens in the Test Harness and set up meetings with the team.
2. The display is enabled for him to notify all his subordinates about any meeting, change in the team activities etc.

QA's

1. QA's use client to continuously interact with the repository to perform all sorts of quality tests on the project code such as Performance test, load test, stress test etc.
2. They use Client to understand the metrics of the project.

Impact on Design:

1. Virtual Display Client is customized to update the metrics of the project continuously so that the QA can take any corresponding action.

Critical Issues

Strict Policies

Client is a generic term which is used to all the team members of the project. Nevertheless, all the team members are not delegated to do the same. Every role of client has predefined activities and they are forbidden to do any things they are not intended to do so. How are they restricted to work on stipulated lines?

Solution

Strict policies are enforced on what a client can do and want he cannot. All the clients will abide by the policies. When the policy is changed by the administrator, it is reflected for all the clients who are under that policy.

Local Caching

The client requests files from Repository and Collaboration Server for various purposes. When they request for same file multiple times, they should unnecessarily wait for same file.

Solution:

Local cache is maintained on each client. If a file is downloaded from any server, it is stored in the local cache. If the cache is full, efficient purge algorithms to knock out the least recently files are employed. In case a file is required by the client continuously, it is searched from the cache before making a request to the server.

Collaboration Server

Executive Summary

The Collaboration Server is the core server which is responsible for the project management. It helps the managers to manage the resources, schedule meetings and monitor the progress of the project. It is analogous to the Repository for the very change that it holds all the project management stuff whereas the Repository holds all the files related to the Source Code. It is closely coupled to Virtual Display Server and is of great help during the meeting times. The Virtual Display Server can access all the stuff from Collaboration Server and presents it interactively on the Display.

Organizing Principles

Configurable Tools

The Collaboration Server is highly dependent on the type of project, methodologies employed in the project, style of wok in the project. To make it easy to set up a Collaboration Server for a project, it is imperative that it should be highly configurable so that it can serve the purposes of the project.

Security

The Collaboration Server is a common server which is used by all the members of the SCF. But every document or every piece of information may not be accessible to everyone. Restricted access is given to the Collaboration Server so that developers won't have access to the business documents, work packages scheduling information etc.

Continuous Refresh

The Collaboration Server should be continuously updated to have the latest information of the project status. If a developer updates the status of the mile stone, it should be immediately reflected in the overall project status.

Notifications Manager

The Collaboration Server should maintain distinct mailing lists for all the activities that happen in the SCF. For example, if a developer updates the mile stone of the project, all the people who are in the mailing list have to be sent notifications about the activity. The mailing list may include the Architect, Manager, Team members who work on the same module, Testing team etc. similarly for every activity, mails are to be sent to the corresponding members.

Responsibilities

1. Tight Security authentication is very much required for a Collaboration Server. It contains restricted regions which are accessible to only a set of SCF members. As the Collaboration Server contains all the business documents, budget information, Work scheduling, critical information of the product which may be patented in future, it is to be given restricted access.
2. It may have a version control system to provide automatic version numbers to all the documents submitted to it. This helps to retrieve all the preliminary documents in case of requirement.
3. If the same Collaboration Server is used across multiple teams, it is very important to isolate information of one team from the other.
4. Developers should be able to extract the High-level design documents from the Collaboration Server.
5. Developers should be able to submit the Design level documents for review to the Collaboration Server.

6. Architects uploads all the High-level documents to the Collaboration Server and make them available to the team.
7. It should store all the information[18] of the employees and their hierarchy[19] in the team which is queryable and easily retrievable.
8. It should store all the calendars of the employees and updates them if they are invited for a meeting.
9. It should hold all the work package descriptions that define the flow and sequencing of all work activities for the project, monitor the status of work in-progress and completed, and measure the resources expended on each task. Each work package is assigned to a single Responsible Individual (RI) has a job description and tangible milestones to measure completion.
10. It should have a notification mechanism which notifies the corresponding mailing lists about any activity.

Structure

For the ease of maintainability and extendibility in future, the Collaboration Server is divided into following packages

[18] The information include all the personal data, their income, the benefits they are given by the company , their milestones in the project etc.

[19] Typical Hierarchy in a Software development company is
Architect and Project Manager->Team Lead->Developers

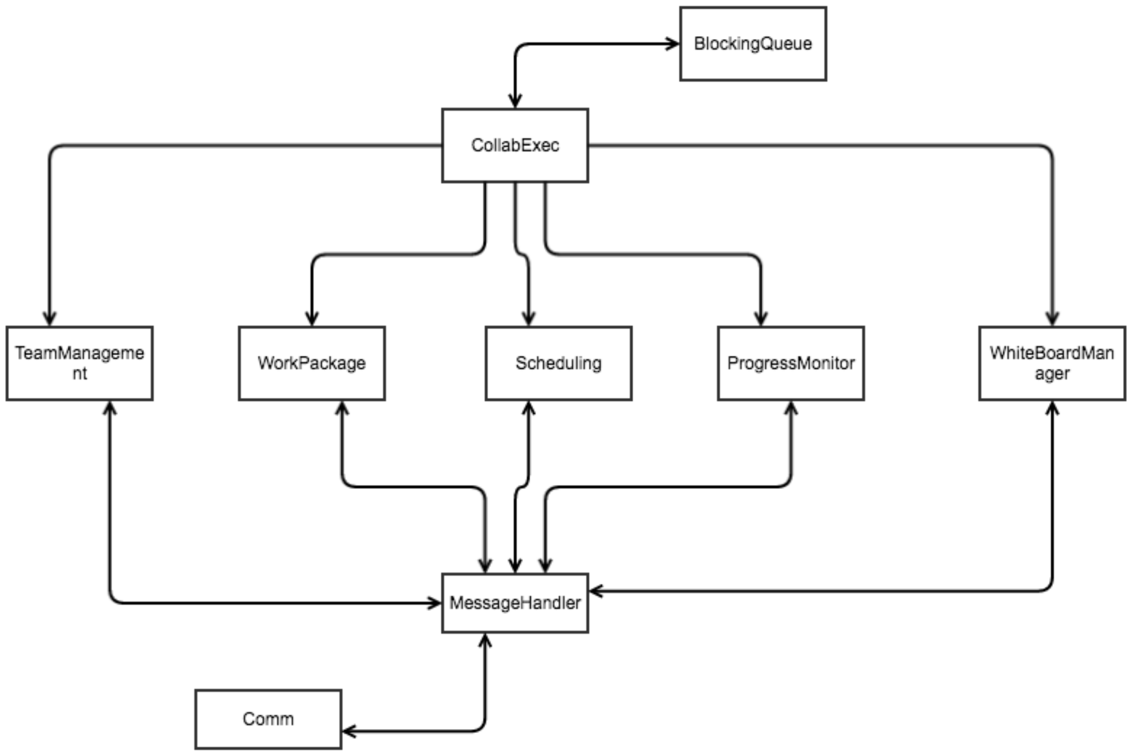


Figure 11 Collaboration Server Package Diagram

CollabExec

1. This is the core package which processes all the requests coming from any Server.
2. It is the entry point which processes all the requests from the multiple Servers.
3. It queues all the messages into the BlockingQueue. Each request is dequeued and processed by one of the threads spawned from the main thread.

TeamManagement

1. It processes requests which involves the information about the Team hierarchy (All the information of a Team members) and their corresponding mailing lists.

2. It processes the upload/download requests coming from multiple clients.
3. This package is used to send notifications to a part of team/Whole team or to the whole project members.
4. All the Team activities are processed by this package.
5. It gets the information of all the team members by performing a set of queries on the database.

WorkPackage

1. It is used to assign work packages to the team members and blocking their calendars accordingly after assigning them a work package.
2. It processes the size of work packages, number of work packages etc.by considering the number of resources and the amount of work to be done.

Scheduling

1. It manages all the work schedules of the team members. It is used to schedule work to the team members based on his calendar.
2. It is used to schedule meetings for a set of team members of to the whole team.
3. When the meetings are scheduled, it notifies the person who is scheduling the meetings about the free time of the team members.
4. It updates the calendar of the team member when a meeting has been scheduled and prevents others from scheduling any event in that time.

ProgressMonitor

1. The status of the project is updated when every developer updates his accomplishments.
2. Time sheets are processed which are received from multiple clients and the corresponding work progress of the developer is updated.
3. In case when a mile stone time line is exceeded than expected time, this package is used to send the notification/mail to the corresponding clients to whom the work package is assigned.

WhiteBoard

1. It processes the request from the White board of the VDS for a document or work schedule or any project related document.

For all the above packages, the response is sent via MessageHandler and Communication package to the corresponding destinations.

MessageHandler

1. This package is used to build messages which are to be sent to the corresponding destination probably in the form an XML.
2. The message body is constructed with the details of the source and destination so that it can be processed by the destination server.

Comm

1. This package is used for communication between Collaboration Server and other Servers.
2. The main functionality of sending and receiving messages/files is implemented by a class which implements a common interface.
3. The same interface is implemented by all the Communication packages so that message exchange and file exchange are remotely possible.

Activities

The following Activity Diagram demonstrates various activities that happen in the Collaboration Server for various types of requests.

1. CollabExec receives message from various Clients and queues them to the Shared Blocking Queue.
2. One of the thread dequeues message from the Blocking Queue and processes it. It directs the message body to the corresponding package.
3. If the message is a request regarding Team managing, required information is fetched from the Database.
4. If the message if a request regarding work package details, the database is searched for work package details of the developer making the request. The work package details are retrieved.

5. If the message is a request regarding Scheduling meeting, a meeting is scheduled and the calendar of the developer is blocked. A notification is sent to him that he has been invited for a meeting.
6. If the message is a request regarding the progress of the project, the query is processed and the required result is obtained.
7. If a white board sends a message requesting for a document/Schedule, it is processed and the required document is uploaded instantly to the VDS.
8. For all the requests, the message is wrapped in an XML body which contains the source and destination IP and it is sent via a communication channel.

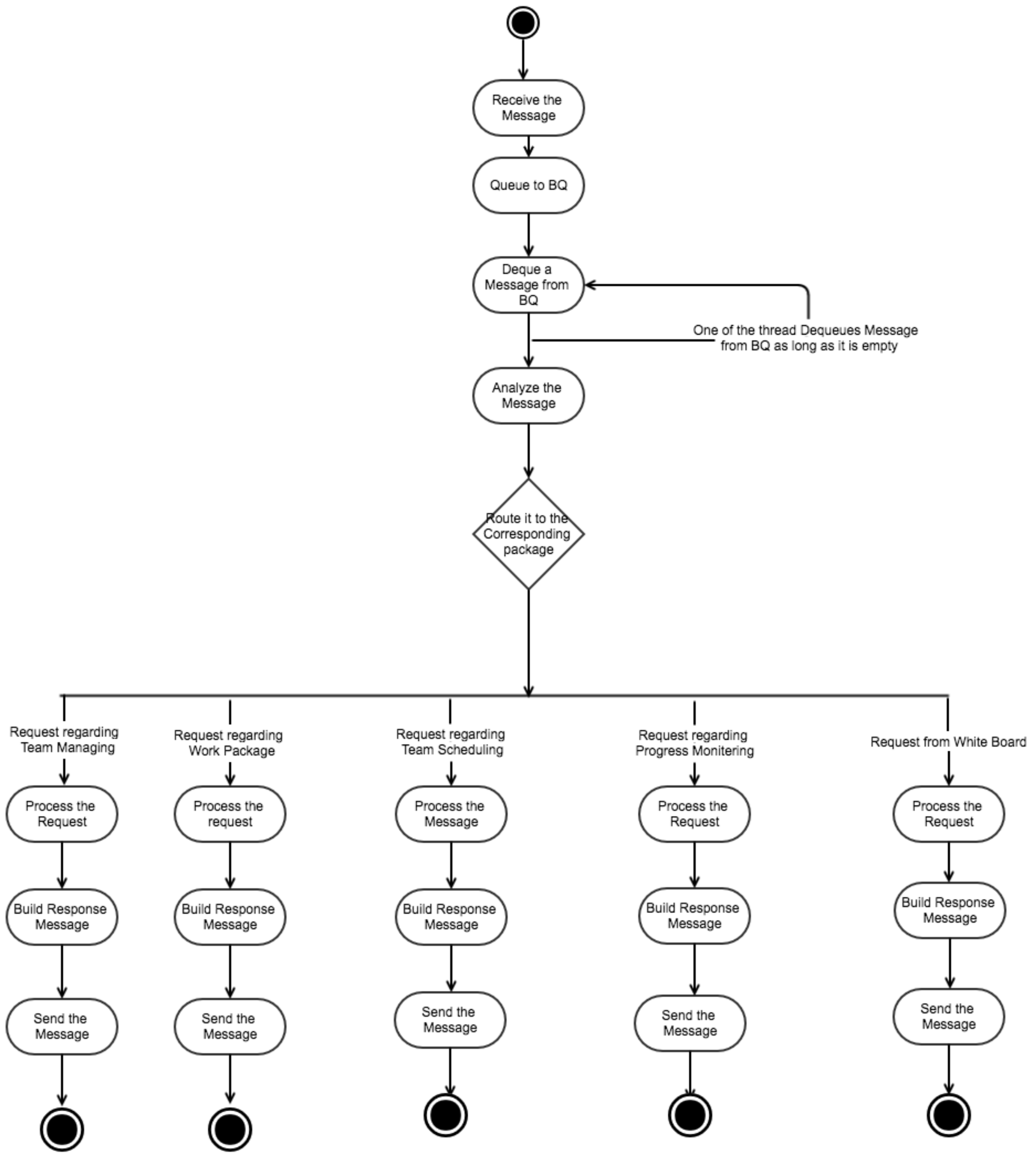


Figure 12 Collaboration Server Activity Diagram

Uses and Users

Architect

1. He uses Collaboration Server to download requirement documents which are uploaded by the document.
2. He uses Collaboration Server to upload Functional requirements documents and gets its approved by the Customer.
3. He uses Collaboration Server to upload High Level Design Documents and make them available to all the developers who are assigned to work on the project.
4. He uses it as a meeting platform to communicate with the Customers, Managers as well as Developers.
5. He uses Collaboration Server to understand the metrics of the project by pulling stats out of the Repository.
6. He can save all the UML diagrams drawn on the Virtual Display Server as a part of meeting.

Developer

1. He uses Collaboration Server to get all the information about the tasks assigned to him, his work packages information, his mile stones, his meeting schedules.
2. He uploads all his time sheets every day to the Collaboration Server so that his development activity can be updated.
3. He can extract the High-level design documents from the Repository and starts developing Design level documents for it.
4. He can understand the progress of the project from the data of the Collaboration Server.
5. He can see the bugs raised by Test Harness after the completion of an Integration Test.
6. He can communicate with other project members with the help of Chat Servers and Video Conferencing etc.

Managers

1. He uses Collaboration Server to schedule meetings with the Developers the status of the project.
2. He uses Collaboration Server to set up meetings with the customers to discuss about the business requirements.
3. He uses Collaboration Server to analyze the current status of the project i.e. Is the project in line with the schedules, should the project need more resources? etc.
4. He hires new resources for the project by looking at the job descriptions of the employees and picks them who suits the skill sets required.

Critical Issues

Notify overlap of events

When a manager schedules meetings, if the calendar of a developer is blocked by some other event, it creates a chaos.

Solution:
Efficient algorithms should be implemented as a part of business logic so that they will prevent the manager from scheduling meeting when the calendar of the developer is already blocked by any other event.

Data loss and Cloud backup

As all the documents, work schedules, employee details are crucial, if the information is lost, it creates chaos.

Solution:
Data should be replicated across virtual servers in multiple locations. Also, this helps when a severe load is created on a server. With the help of load balancing techniques, the requests are routed to relatively free servers. Also, a cloud backup is necessary which will take of accidental loss of data on all the servers.

Offline Availability

Sometimes due to heavy traffic on the Server, the server may go down. This prevents the developers from accessing any previous documents and work schedules also.

Solution:

Offline Availability of Collaboration Server is to me made possible. In this case, even if the server goes down, the last updated session of the developer will allow him all the content that was available in the that session.

Visual Display System

Executive Summary

The Virtual Display Systems are interactive GUI's which are closely integrated with Collaboration server and assists various users of SCF in various activities. The main concept of Virtual Display Systems is to enable the team members communicate both verbally and visually with SCF in which VDS is embedded. Virtual Display Systems are of two types: Virtual Display Clients and Virtual Display Servers.

Virtual Display Clients are installed on all client machines to make them available for the members of Software Collaboration Foundation. The Virtual Display Client will assist the client in various activities like interacting with Repository, Test Harness, Collaboration Server and the peer clients as a part of Software development.

Virtual Display Server is a single server which is common for all the Clients and all the other Servers of the Software Collaboration Foundation. It is used when a meeting has to be set up between multiple developers working remotely from multiple locations. It provides various Visual tools such as Webcams, chat windows, IDE (to run code), white boards etc. to make the meetings interactive and feasible. VDS is highly configurable to suit the requirements of the team.
The Virtual Display Server, Collaboration Server and Repository are collectively called Virtual Repository Test Harness System(VRTS).

Organizing Principle

1. There are multiple VDCs[20] and s single VDS[21].
2. All the VDC's are interconnected as well as connected to the VDS.
3. The Display System is called virtual because it can be altered according to the user requirements. A Virtual Display will have multiple windows which does a specific task. For example, a client can have multiple displays in which one will have the dependency graph of the repository, the other will have the test results from the Repository and the other may have documents from the Collaboration Server.
4. The VDC will help the client to interact with VDS during meetings.
5. In order to support collaborative interactive drawing, text messaging, and VoIP communication, latency will be an important issue for VDS. Therefore, you should develop a concept for two-tiered storage, e.g., fast short-term storage within VDS, and long-term storage in VRTS servers.

Uses and Users

Virtual Display Server

1. It is an interactive collaborative display server which constantly interacts with Collaboration Server.

2. During the meetings, an architect draws images, writes notes etc. to demonstrate the requirements, activities, Source code of the project. The display system should be able to store all the data that is written/drawn on the whiteboard. Collaborative drawing and document pasting requires each Virtual Display to send a stream of messages to all other connected virtual displays.

3. It should be enabled with a digital white board. The white board should be able to understand all the freehand diagrams and store them in a specified UML formats. For example, if an architect draws a package diagram, the Virtual Display Server should intelligently identify it as package diagram and

[20] VDC- Virtual Display Client
VDS- Virtual Display Server
21

store it making necessary amendments to the diagram and convert it into full fledge package diagram.

4. It should display all the contents of the Collaboration Server so that any image or document can be dragged from the Collaboration Server on to the White board to explain about it or to make changes to it.

5. It should have a configurable[22] IDE embedded in its architecture and connected to the repository. During meetings, any part of code can be pulled out of the repository, tweaked and executed to solve a technical issue or analyze a situation. If a customer raises a bug, he can demonstrate the bug when the development team and the customers are available in the same meeting.
6. It should show a hierarchy of Systems, Programs, and Packages, connected in dependency order, annotated with names and version numbers, and providing access to detailed information through specification and design documents, and source code views.

7. Special purpose charts and visuals, based on user-defined templates can be created, placed anywhere on the drawing surface, edited, saved, and retrieved at any time. These charts show information about a current software development project in ways that are briefly described below, and which you will explore as part of this project.

8. It provides a simple and intuitive way of navigating through the source code, test results, charts, and diagrams and display them as HTML pages on the Display.

9. Webcam images of collaborators can be placed in resizable windows anywhere on the display surface, and linked through Voice over IP.

[22] The IDE should be configurable because the IDE is chosen depending on coding language of the project. It should be able to switch between the IDE’s if the project uses multiple languages.

10. It should have an embedded Visio/Gliffy software. The software will help the demonstrator to draw UML diagram, flow charts, demo GUIs and explain them.

11. It should have a 'Raise hand' option enable for every client who are part of the meeting. If a client raises hand, the host of the meeting gets a notification

12. The Virtual Display Server should be able to set up meeting which involves people from multiple locations. A meeting invitation is sent to all team members who are supposed to attend the meeting. The meeting host will receive the response of the team members to whom the invitation is sent. While scheduling, he gets to know if the calendar of any invitee is blocked with another meeting.

13. It should display the progress in various visually appealing ways. The completion status of the project should be in different color when compared to the uncompleted part. They should be clickable to reveal the internal details of the completed/uncompleted portions.

14. It should probably understand voice commands and act accordingly. All the communication should be documented accordingly in case we want to retrieve them back using voice recognition. Deep learning algorithms are implemented to intelligently identify the voice and store the conversation in the document.

Virtual Display Client

1. It is installed on every client machine and is coupled with the client.

2. The VDC[23] provides interface for all the clients who are part of Software Collaboration Foundation.

3. The VDC of a client is customized according to the type of project and the role of the client

[23] VDC- Virtual

4. The VDC will allow the client to view documents which are present in the collaboration Server as well as his local machine.
5. It contains different webcam views which show all the team members who are present along with him in the meeting.

6. It shows the various visual tools like bar graphs, pie diagrams, tables and various charts that describe the progress of his work in the project.

7. It also shows the progress of the project which is similar to the one displayed in the Virtual Display Server.

8. It contains the chat windows, video conferencing panels which are used for interacting with other clients.

9. It should display the time line of work packages, colored to show the completed packages and the work to be done ahead.

10. It should display the test results he received from the Test Harness clearly. The passed TestCases are given green color, the failed once are given red color and the ones which could not be executed are given yellow color. It should give an option to the client to filter the results based on day, week or quarter. So, that a Client can view all the TestRequests made by him and their corresponding results during a day, week or quarter.

11. Support ad-hoc queries from the Repository and Collaboration Servers by selecting tables, and attributes from each table. The display may possibly allow entering conditions on other attributes that limit the records returned and displayed.

12. It should provide means to optionally save any of these ad-hoc queries as views that could be re-run with a single tap on the control panel at any time.

13. It should allow Clients to schedule a set of TestRequests to be sent to the Test Harness at any scheduled time. This will allow Client to perform Scheduled Tests.

Views

The samples views of various Virtual Display Systems are shown below. They are very dependent on the project. Also, there are multiple windows for each server. These are just some of them.

Virtual Display Server

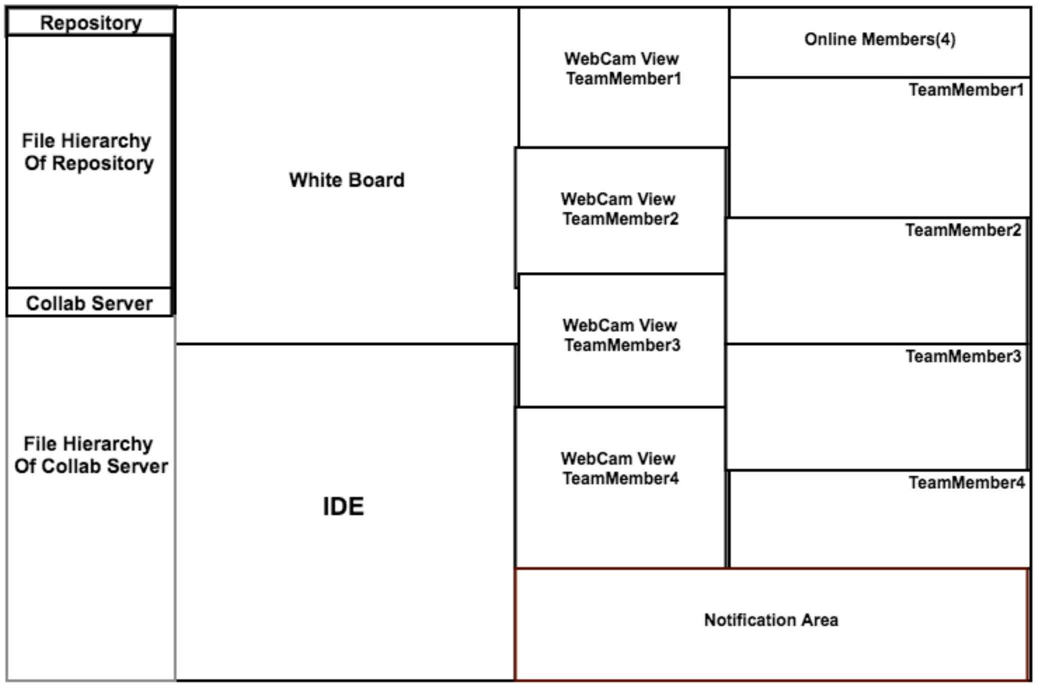


Figure 13 View of Virtual Display Server

Virtual Display Client

Developer

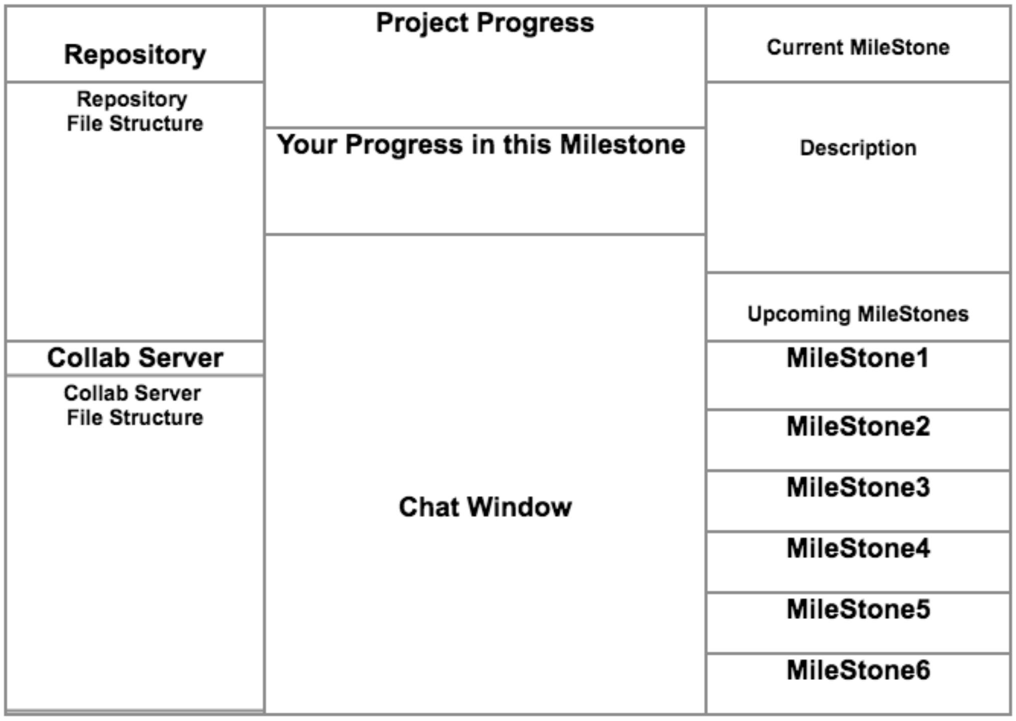


Figure 14 View of Developer VDC

Manager

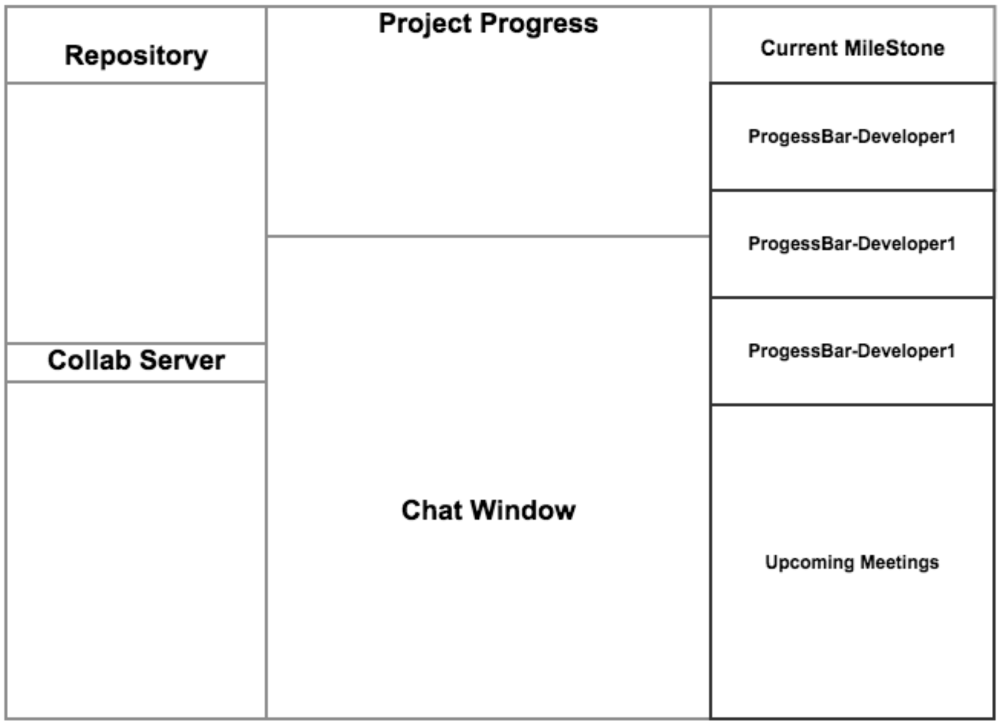


Figure 15 View of Manager VDC

Critical Issues

Collaborating multiple Teams

As each team has its own Collaboration Server, Test Harness Server and Repository Server, how will different teams collaborate code or documents?

Solution:

Virtual Displays of multiple teams can be connected into ad-hoc networks within a location and between locations to support the efficient exchange of information between teams.

Changes in main Repository and Collaboration Server

During meetings, if the host makes changes in the code or a document, they will be reflected in both the Repository and Collaboration Server.
What if the host don’t want to retain the changes?
Solution:
Every project is provided with two sets of servers (Main Repository, Secondary Repository, Main Collab Sever and Secondary Collab Server). If any changes are made by the host during meetings, they will be checked into the Secondary servers and the primary server remains unchanged. It is up to the interest of the host if he wants to reflect the same in Primary Server.

Communication System

Executive Summary

As SCF is a federation of servers which interact with each other for the collaboration activities, it is imperative that there should be an efficient Communication system. Two remote machines can communicate with each other with the help of communication systems efficiently. The communication system is structured to have both sending and receiving channels. Using the communication system, the messages are sent from one service end point to the other. Usually a service end point is hosted on an application or server. Both Service and Clients communicate with each other using communication channels and end points.

The Sender consists of

- Sending Message Blocking Queue
- Proxy List

The Receiver consists of

- A Web service hosted on it
- Receiving Blocking Queue

Organizing Principles

The Communication Services are bound to follow some organizing principles.

Explicit Boundaries

All the boundaries must be explicit. No attempt is made to hide communication.

Autonomous Services

The Services can be Deployed, managed and versioned independently. So that they can work on the class platforms also.

Services share Contracts and Schemes

- The Contracts define behavior and schemas define data.
- Client needs a reference to the Contract, not to the Service implementation.

Compatibility is Policy-based

Policy supports separation of behavior from access constraints.

Key Architectural Ideas

Contract

All the services expose certain contracts. The contract is a platform-neutral and standard way of describing what the service does. There are various types of contracts.

Service Contract

They describe which operations the client can perform on the service.

Data contract

They define which data types are passed to and from the service.

Message contract

They allow the service to interact directly with messages.

Binding

1. Bindings are objects that are used to specify the communication details that are required to connect to the endpoint of a Communication service.
2. Each endpoint in a WCF service requires a binding to be well-specified.

End Point

[24]All communication with a Communication service occurs through the *endpoints* of the service. Endpoints provide clients access to the functionality offered by a Communication service.
Each endpoint consists of four properties:

- An address that indicates where the endpoint can be found.
- A binding that specifies how a client can communicate with the endpoint.
- A contract that identifies the operations available.
- A set of behaviors that specify local implementation details of the endpoint.

Activities

[24] https://msdn.microsoft.com/en-us/library/ms733107(v=vs.110).aspx

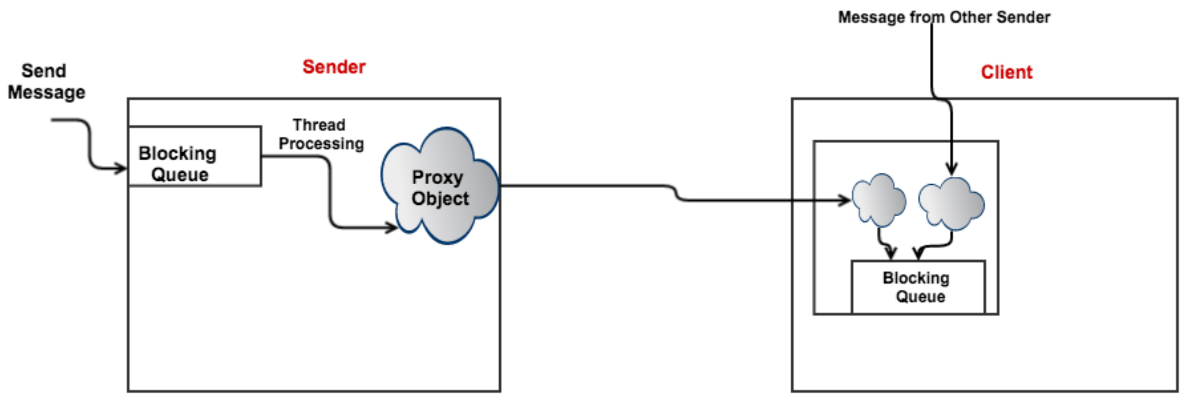


Figure 16 Communication System Activity Diagram

1. A message which is sent to the receiver is enqued in the Sender Blocking Queue.
2. A Child thread will dequeue the message from the sender Blocking Queue and will add the Sender address and receiver address to the message body and creates a proxy object.
3. The sender maintains a list of proxy servers which hosts the web services on various servers.
4. The received details which are stored in the message body is searched in the proxy list.
5. If the proxy is not available, then it is created in the proxy dictionary.
6. If the proxy is available, the sender can now post the message to the receiver through the communication channel by a remote call.
7. The web service hosted on the receiving server will receive the message sent from the sender.
8. The message is de-serialized by the web service at the receiver end and is converted into a Class object.
9. To prevent the loss of any object received at the receiver, the object is sent to the Blocking Queue as soon as it is received.
10. A Child thread waits on the Blocking Queue to receive a message and process it.

Class Diagram

The following Class diagram illustrates the technical details of Communication channels.

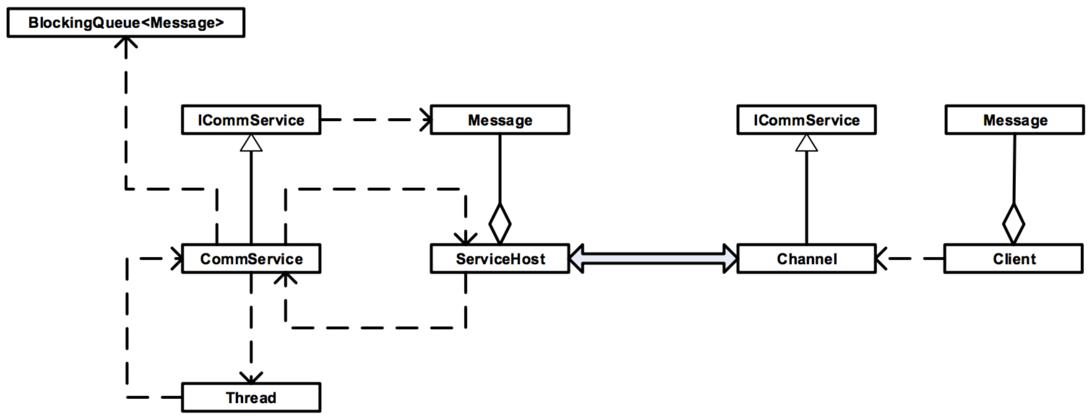


Figure 17 Communication System Class Diagram

Uses and Users

Every user uses Communication to communicate with the other servers. Every federated system (TestHarness, Repository, Client, Collaboration Server) has a dedicated package called Communication for this purpose. It is associated with a Message package which is used to construct the message body. The Communication package will have a class (CommService) which has all the methods required for communication like PostMessage, GetMessage, UploadFile,DownloadFile etc. All the CommServices classes in all the packages will implement a common interface (ICommService) which is a common contract for all the federated servers.

Critical Issues

Security

As many servers communicate among themselves, there are high chances of some foreign server try to hack the repository server or Collaboration Server.

Solution:
All communication channels are authenticated using strict firewalls before accessing the contents of the server. To enhance this, efficient hashing algorithms are used to encrypt data and decrypt them on the other end.

Bottle Necks at the Servers

There are chances that a server might be clogged with a glut of messages and it is unable to process them.

Solution:
To avoid Bottle necks, Blocking Queues are implemented at all the Servers. When a message is received, it is pushed into the blocking Queue instantly. Multiple threads are spawned from the main thread which will process the messages one by one.

Messages

Executive Summary

Messages are prime vehicles of communication between all the federated servers in the SCF. The federated servers communicate with each other using Messages to provide elegant software development mechanism. A message which is being sent to the receiver should comply with some policies so that it can be easily extracted at the other end. The messages usually have Source address, Destination address, Message body and some other data specific to that message. The messages are serialized before sending it remotely via a network. All the advanced languages provide nice abstraction from developer worrying about the serialization. The Message received at the receiver end is deserialized and processed.

As SCF is a federation of multiple servers, there should be common interface om which messages are sent and received. All the communication service classes implement ICommService to accomplish this. Similarly, all the message bodies should follow a specific format to make them understandable by all the servers. All the messages are probably in XML format which has predefined tags so that they can be parsed accordingly.

Organizing Principles

The message construction should abide by some principles in order to make them an integral part of SCF.

Structure

All the message bodies should follow specific template which contains predefined tags. This is intended because every server has same message parser. So, the message which is sent from one server to the other should follow a template which is flexible.

Address Information

Every message which is being sent should have Source address, Destination address embedded in it. The Destination address helps the communication channel to hit the destination server. The Source address helps the destination to understand the origin of the message.

Uses and users

Every Server should have Message package which accomplishes the building of the Message. The Message package is used by Communication channel to communicate with the other servers.

Client

1. Client uses Messages to send TestRequest to Test Harness for Testing.
2. They receive Test Results in the form of messages from TestHarness.
3. They send requests to Collaboration Server for documents in the form of Message.
4. They receive results from Collaboration Server in the form of Message.
5. They use Message to request a checkout from Repository.

TestHarness

1. Test Harness uses Message to send Test Results to the requesting clients.
2. It uses Message to send the log file to Repository after the testing.
3. It uses message to request the files. The message body contains the list of all the file names which it requires.

Collaboration server

1. It uses message to send notifications about meeting schedules.
2. It uses message to send the results of query to the client.

Repository

1. It uses message to send query results to the requesting client.

Structure

Various Types of Messages in the SCT are

Check-In from Client

```
<?xml version="1.0" encoding="utf-8"?>
<Message>
<senderAddress>http://178.168.1.1</senderAddress>
<senderPort>8051</senderPort>
<recipientAddress>http://189.1782.10.2</recipientAddress>
<recipientPort>8061</recipientPort>
<reciepientEndPoint>/Repository/CheckIn<reciepientEndPoint>
<Payload>CheckIn</Payload>
<description>Check-in comments</description>
<MessageBody>

<files>File1.cs</files>
<files>File2.cs</files>
<files>File3.cs</files>
</MessageBody >
</Message>
```

From the Client end, all the above files are upload when Checking-in along with the above message.

Check-Out by Client

```
<?xml version="1.0" encoding="utf-8"?>
<Message>
<senderAddress>http://178.168.1.1</senderAddress>
<senderPort>8051</senderPort>
<recipientAddress>http://182.167.10.3</recipientAddress>
<recipientPort>8062</recipientPort>
<reciepientEndPoint>/Repository/CheckOut<reciepientEndPoint>
<Payload>Check-Out</Payload>
<MessageBody>
<fileNames>TestHarness/BlockingQueue.cs</fileNames>
<fileNames>TestHarness/Messenger.cs</fileNames>
</MessageBody>
```

```
</Message>
```

TestRequest from Client

```
<?xml version="1.0" encoding="utf-8" ?>
<Message>
<testRequest>
  <author>Saisumanth Gopisetty</author>
  <test name="First Test">
    <testDriver>TestDriver1.dll</testDriver>
    <library>CodeToTest1.dll</library>
  </test>
 <test name="Second Test">
    <testDriver>TestDriver2.dll</testDriver>
    <library>CodeToTest2.dll</library>
  </test>
</testRequest>
<Message>
```

TestResult from TestHarness

```
<Message>
<testResultsMsg>
<testKey>Saisumanth
Gopisetty_12_6_2016_22_15_58.9021003_ThreadID11</testKey>
  <timeStamp>12/6/2016 10:16:11 PM</timeStamp>
  <testResults>
    <testResult>
      <testName>test1</testName>
      <result>passed</result>
      <log>demo test that always passes</log>
    </testResult>
    <testResult>
      <testName>test2</testName>
      <result>failed</result>
      <log>file not loaded</log>
    </testResult>
    <testResult>
      <testName>test3</testName>
      <result>failed</result>
      <log>demo test that always fails</log>
    </testResult>
  </testResults>
</testResultsMsg>
<Message>
```

Document request from Developer

```
<Message>
<senderAddress>http://192.168.1.1</senderAddress>
<senderAddress>http://178.168.1.1</senderAddress>
<senderPort>8051</senderPort>
<recipientAddress>http://182.167.10.3</recipientAddress>
<recipientPort>8062</recipientPort>
<reciepientEndPoint>/CollaborationServer/DocRequest<reciepientEndPoint>
<PayloadType>Collaboration</PayloadType>
<description>Collaboration of Documents</description>
<fileName>Project5</fileName>
<payload>
<fileName>HLD1.doc</fileName>
<fileName>HLD2.doc</fileName>
<fileName>HLD3.doc</fileName>
</payload>
</Message>
```

Request from Build Server to TestHarness

```
<senderAddress>http://192.168.1.1</senderAddress>
<senderAddress>http://178.168.1.1</senderAddress>
<senderPort>8051</senderPort>
```

```
<recipientAddress>http://182.167.10.3</recipientAddress>
<recipientPort>8062</recipientPort>
<reciepientEndPoint>/TestHarness/Build<reciepientEndPoint>
<PayloadType>Build</PayloadType>
<description>Build File</description>
<fileName>Project5</fileName>
<payload>
<BuildName>BuildFile.proj</BuildName>
</payload>
</Message>
```

These are sample messages which are used by various servers during communication. Apart from these, there are many Messages which are used as a part of development in SCF.

Critical Issues

Improper Message

As the messages are written by Developer's, sometimes there may be errors in the tags used which will cause exceptions while parsing the messages.

Solution:
Instead of Developer's writing the XML manually, they are provided with an UI which will help them to build the XML internally. In this way, the syntax errors can be resolved.

Storage Management Subsystem

Executive Summary

Software Collaboration Federation is a great solution for collaborative software development. We have dedicated servers for Testing (Test Harness), Managing Source code(Repository), Display Server (Virtual Display Server) and Project Management (Collaboration Server). All the servers communicate among themselves for collaborative software development. But many servers and clients communicate among themselves, it is difficult to manage the communication among them, notify the occurrence of events, Route Data etc. A great solution for this is to have a Central server which manages communication and data routing. SMS provides an abstraction to all the servers and clients who use Software Collaboration System from worrying about the Communication, Notifications and Data routing.

SMS is a central server which manages all the communication between various components of SCF. All the components of Software Collaboration Federation have a SMS subsystem through which that component communicates with other components of the SCF.

SMS is a hierarchical composition of Data and Event managers. They reside on each of the components of the SFC. The central SMS server will have Data and Event coordinators who control the operation of all the Data and event managers.

I did not make SMS as a part of the SCF in all the above discussions. But if SMS is a part of SCF, the structure of SCF would be like this.

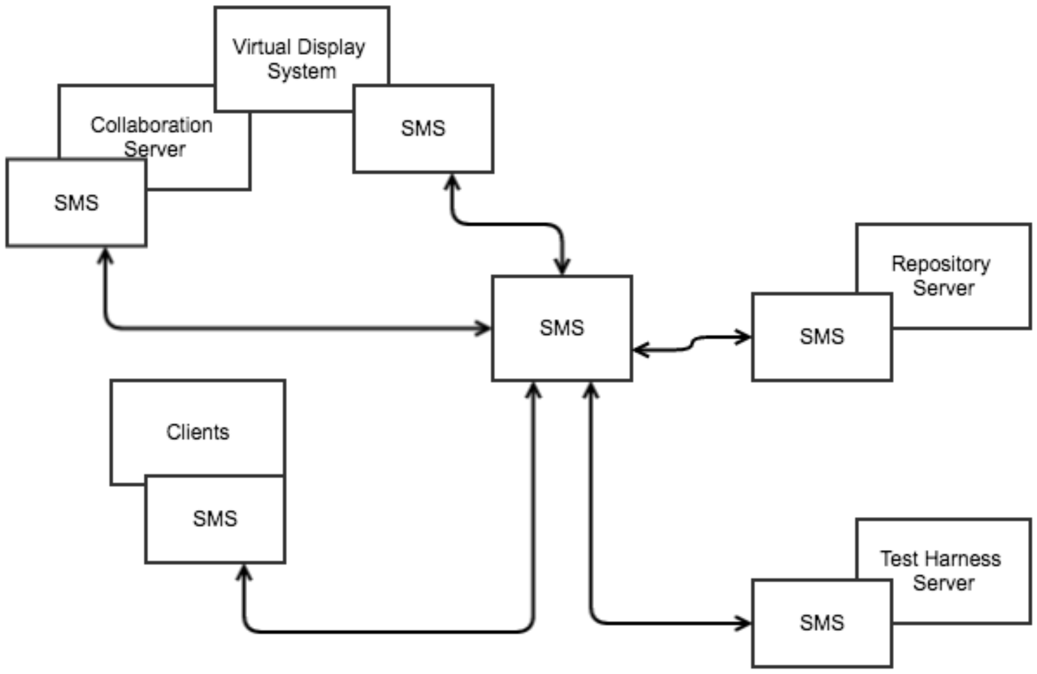


Figure 18 SCF Structure with SMS

All the servers and clients communicate via SMS in this case.

Components of SCF

SMS consists of two components which will help them to accomplish its tasks.

Event Managers

As many systems collaboratively work together in the process of software development, many events occur as a part of it. For instance, sophisticated notifications are to be sent to the corresponding servers when an activity happens. For example, Test Harness is to be notified of the new inkling build from Build Server, Developers are to be notified about the schedules of meetings etc.
This works on the Pub-Sub model. When a new subsystem comes to the SCF, it must register for a set of events with the SMS. This will make the subsystem a part of the event subscribers list. When an event triggers, if the subsystem is a part of the subscriber’s list, it gets a notification.

Data Managers

Various types of Data is exchanges between the components of the SCF as a part of Software Development. The various types of Data include Messages (which might be in the form of XMLs), files etc. All the data is routed via SMS so that we ensure that data is exchanged without any mismatch. I have discussed in Messaging how the sender will look through a Dictionary of proxy list to find the end of the receiver. In this case SMS will take care of all the routing and maintains the proxy list.

Structure

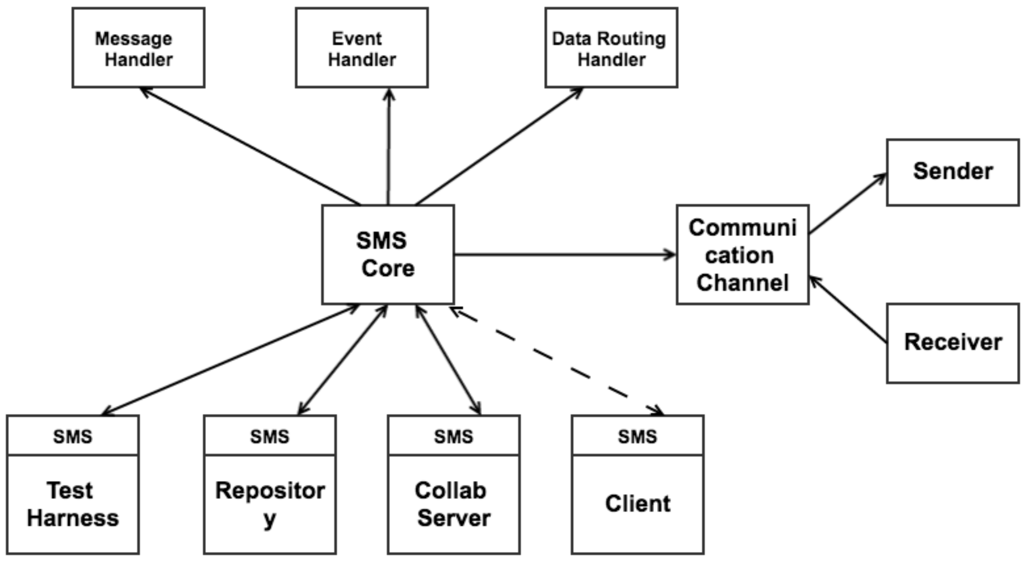


Figure 19 SMS Package Diagram

The dotted line here indicates there can be many clients connected to the SMS.

SMS Core

This is the Core Handler of SMS. This is the point of contact of all the Servers. All the SMS components on all the subsystems interact with this service to communicate with other servers. This is the entry point for all the requests from multiple subsystems of SCF. Depending upon the type of request, the request is routed to the corresponding service like Message handler, Event handler or Data Routing handler.

Message Handler

The Message Handler will parse the message it received. The message is obligated to have Destination address enclosed with it. It looks in the proxy list for the Destination URL and sends the message to it. If the Destination address in not present is not present in the Proxy list, it is added.

Event Handler

Event Handler is responsible for handling all the events that occur in various subsystems of SCF. When an event occurs in any subsystem the SMS of subsystem sends a notification to the SMS core. The Event Handler of the SMS will look through the subscription list of the events and they are sent event notifications accordingly.

Data Routing Handler

Data Routing Handler is responsible for all the exchange of data among the various subsystems of the SCF. The data request sent by a subsystem of SCF is processed and the request is sent to the destination server. The destination server sends the file to the SMS and it is routed to the requesting server.

Communication Channel

This is responsible for exchange of messages and files among the various servers of SCF. This is discussed in detail in the Communication system.

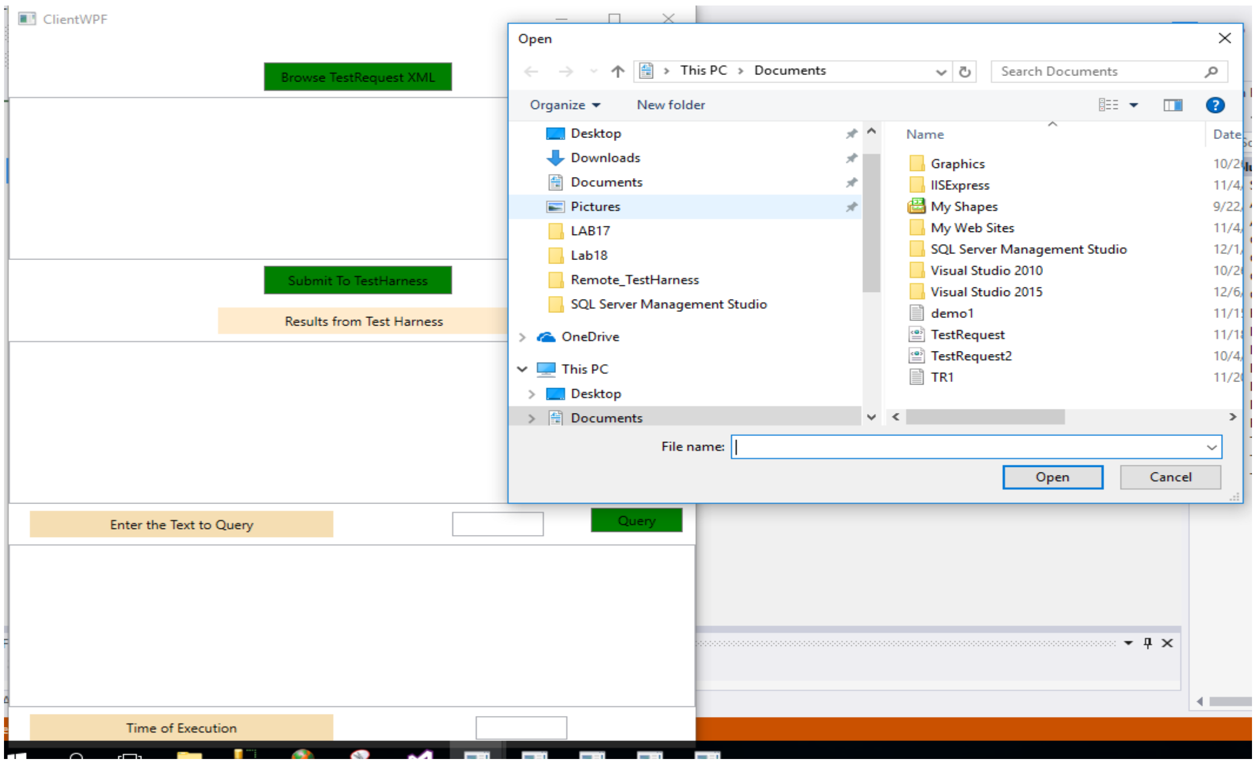
Prototypes

Test Harness

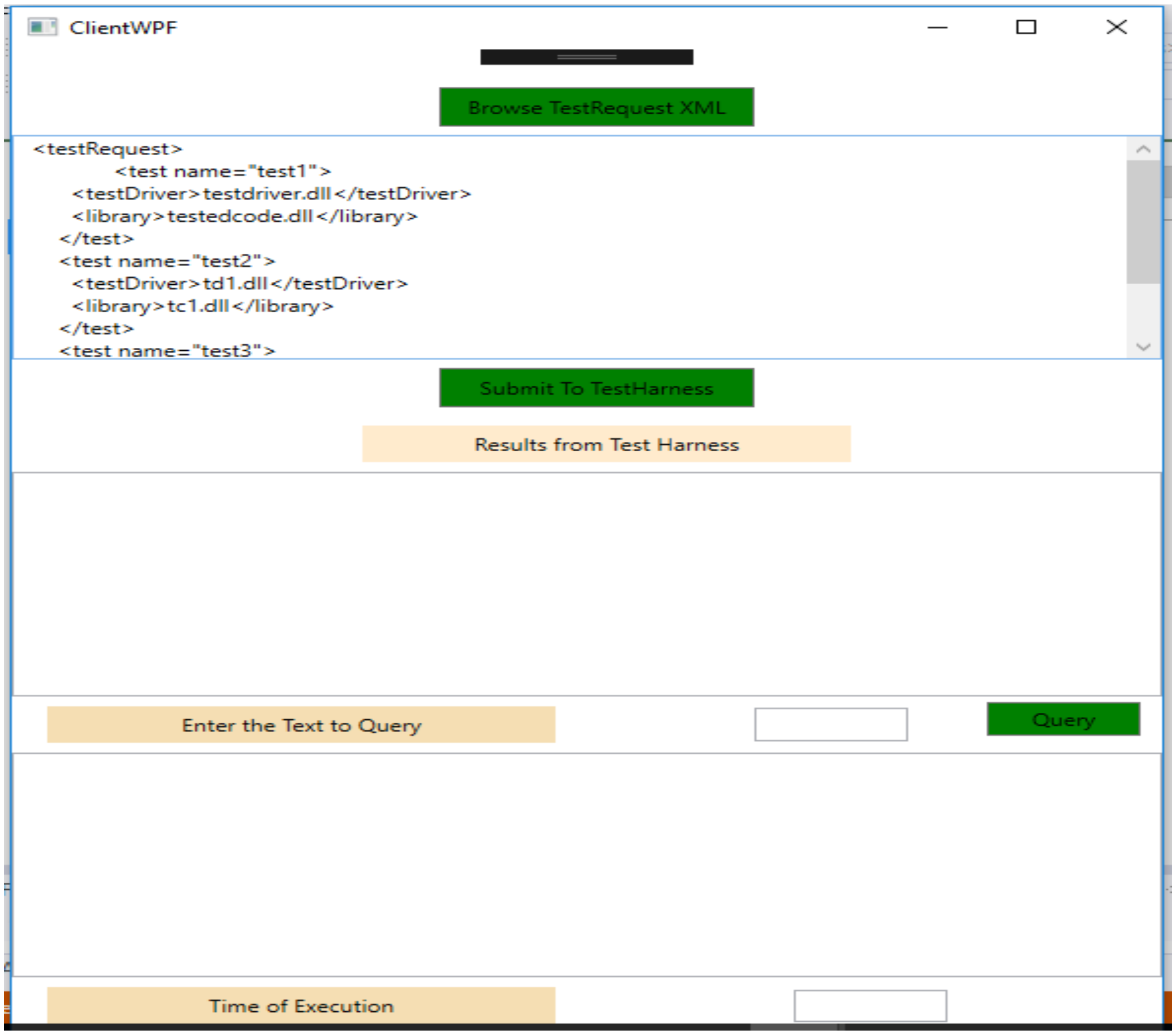
This prototype demonstrates the key functionality of Test Harness. In this prototype, a Developer can submit a Test Request from the GUI. The Test Harness fetches all the files required to perform Test cases from the Repository. After the Testing, the results are sent to the corresponding client and the log file is stored in the Repository.

The following set of screenshots will show the process:

1. Developer selects the Test Request which have information about the Test Driver information and Source code information.

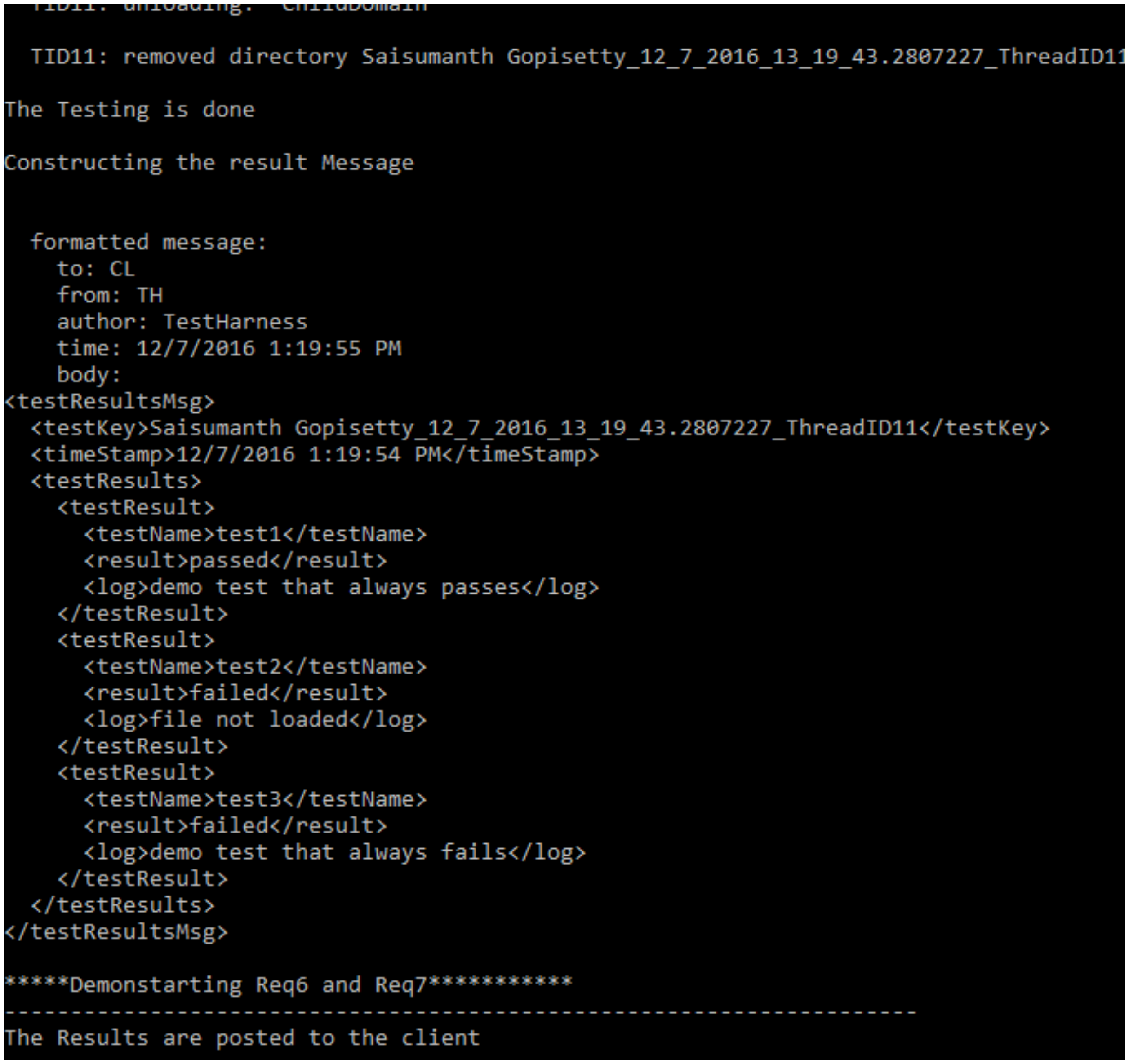


2. The Test Request is submitted to the Test Harness.



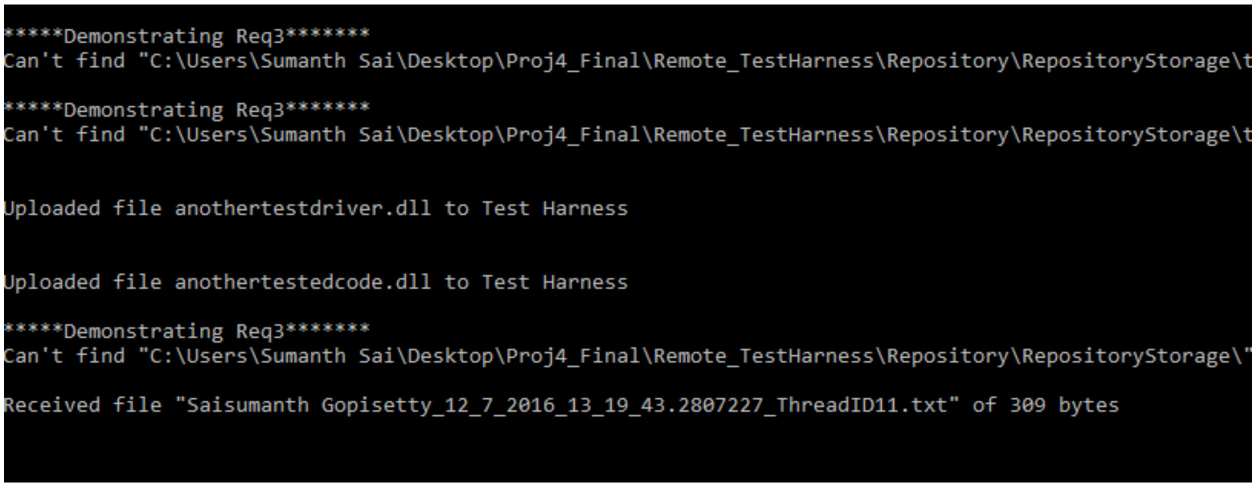
```
<testRequest>
        <test name="test1">
    <testDriver>testdriver.dll</testDriver>
    <library>testedcode.dll</library>
   </test>
   <test name="test2">
    <testDriver>td1.dll</testDriver>
    <library>tc1.dll</library>
   </test>
   <test name="test3">
```

4. Test Harness has processed the Request.



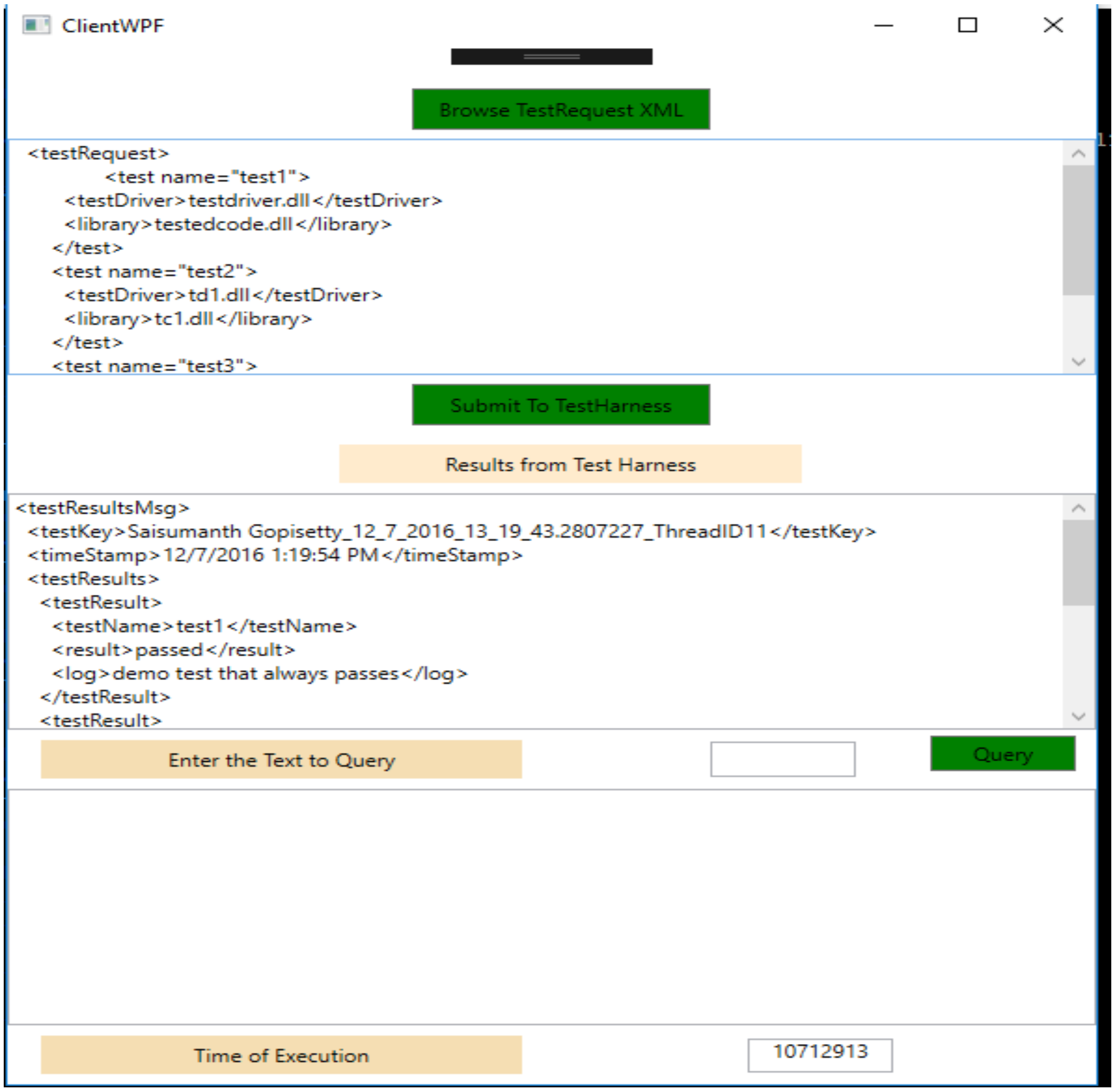
```
  TID11: removed directory Saisumanth Gopisetty_12_7_2016_13_19_43.2807227_ThreadID1

The Testing is done

Constructing the result Message


  formatted message:
    to: CL
    from: TH
    author: TestHarness
    time: 12/7/2016 1:19:55 PM
    body:
<testResultsMsg>
  <testKey>Saisumanth Gopisetty_12_7_2016_13_19_43.2807227_ThreadID11</testKey>
  <timeStamp>12/7/2016 1:19:54 PM</timeStamp>
  <testResults>
    <testResult>
      <testName>test1</testName>
      <result>passed</result>
      <log>demo test that always passes</log>
    </testResult>
    <testResult>
      <testName>test2</testName>
      <result>failed</result>
      <log>file not loaded</log>
    </testResult>
    <testResult>
      <testName>test3</testName>
      <result>failed</result>
      <log>demo test that always fails</log>
    </testResult>
  </testResults>
</testResultsMsg>

*****Demonstarting Req6 and Req7***********
----------------------------------------------------------------------
The Results are posted to the client
```

5. Log file is sent to the Client.



```
*****Demonstrating Req3*******
Can't find "C:\Users\Sumanth Sai\Desktop\Proj4_Final\Remote_TestHarness\Repository\RepositoryStorage\t

*****Demonstrating Req3*******
Can't find "C:\Users\Sumanth Sai\Desktop\Proj4_Final\Remote_TestHarness\Repository\RepositoryStorage\t


Uploaded file anothertestdriver.dll to Test Harness


Uploaded file anothertestedcode.dll to Test Harness

*****Demonstrating Req3*******
Can't find "C:\Users\Sumanth Sai\Desktop\Proj4_Final\Remote_TestHarness\Repository\RepositoryStorage\"

Received file "Saisumanth Gopisetty_12_7_2016_13_19_43.2807227_ThreadID11.txt" of 309 bytes
```

6. Results are sent to Test Harness.

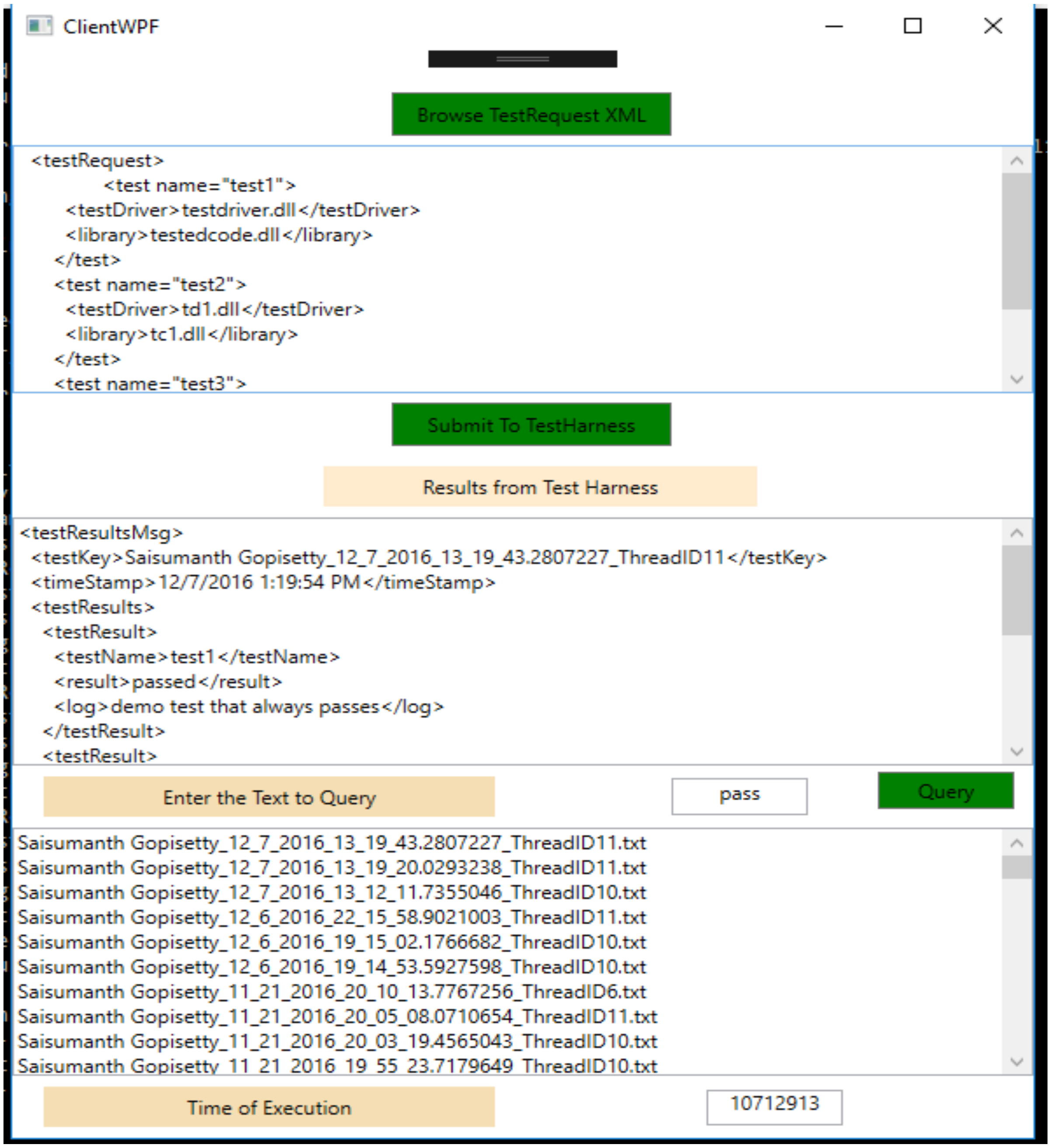


```
<testRequest>
        <test name="test1">
    <testDriver>testdriver.dll</testDriver>
    <library>testedcode.dll</library>
  </test>
  <test name="test2">
    <testDriver>td1.dll</testDriver>
    <library>tc1.dll</library>
  </test>
  <test name="test3">
```

```
<testResultsMsg>
  <testKey>Saisumanth Gopisetty_12_7_2016_13_19_43.2807227_ThreadID11</testKey>
  <timeStamp>12/7/2016 1:19:54 PM</timeStamp>
  <testResults>
    <testResult>
      <testName>test1</testName>
      <result>passed</result>
      <log>demo test that always passes</log>
    </testResult>
    <testResult>
```

Time of Execution 10712913

7. Client can query on the Repository.



```
<testRequest>
        <test name="test1">
    <testDriver>testdriver.dll</testDriver>
    <library>testedcode.dll</library>
  </test>
  <test name="test2">
    <testDriver>td1.dll</testDriver>
    <library>tc1.dll</library>
  </test>
  <test name="test3">
```

```
<testResultsMsg>
  <testKey>Saisumanth Gopisetty_12_7_2016_13_19_43.2807227_ThreadID11</testKey>
  <timeStamp>12/7/2016 1:19:54 PM</timeStamp>
  <testResults>
    <testResult>
      <testName>test1</testName>
      <result>passed</result>
      <log>demo test that always passes</log>
    </testResult>
    <testResult>
```

pass

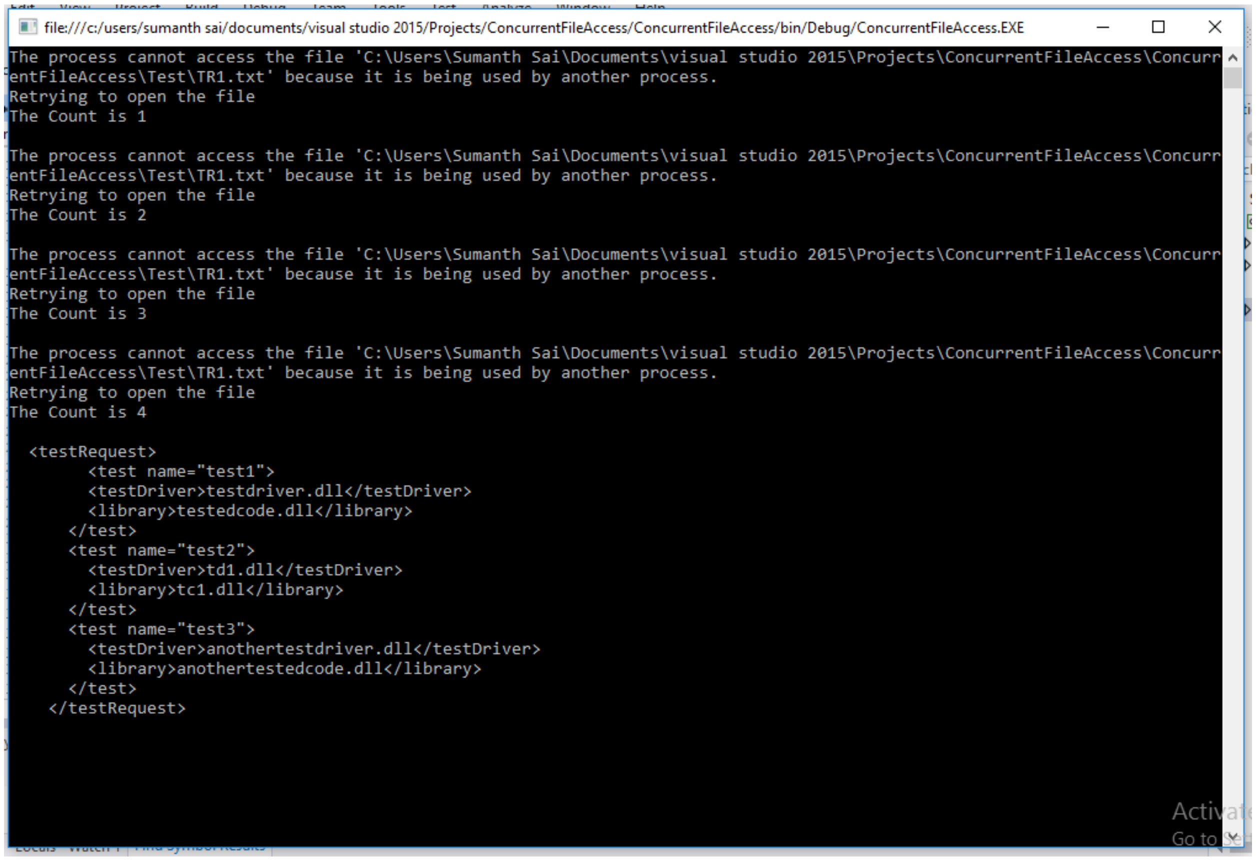
```
Saisumanth Gopisetty_12_7_2016_13_19_43.2807227_ThreadID11.txt
Saisumanth Gopisetty_12_7_2016_13_19_20.0293238_ThreadID11.txt
Saisumanth Gopisetty_12_7_2016_13_12_11.7355046_ThreadID10.txt
Saisumanth Gopisetty_12_6_2016_22_15_58.9021003_ThreadID11.txt
Saisumanth Gopisetty_12_6_2016_19_15_02.1766682_ThreadID10.txt
Saisumanth Gopisetty_12_6_2016_19_14_53.5927598_ThreadID10.txt
Saisumanth Gopisetty_11_21_2016_20_10_13.7767256_ThreadID6.txt
Saisumanth Gopisetty_11_21_2016_20_05_08.0710654_ThreadID11.txt
Saisumanth Gopisetty_11_21_2016_20_03_19.4565043_ThreadID10.txt
Saisumanth Gopisetty_11_21_2016_19_55_23.7179649_ThreadID10.txt
```

10712913

Similarly, Build Server will send the latest build for Testing to Test Harness.

Concurrent File Access

This a solution to one of the critical issues of Repository and Collaboration Server. For a file access, if a client requests for it and if the file is accessed by another resource. The exception occurs and the request of client is wasted. To avoid this in the exception block, a certain number of retries are made with the intension that the file will be released by the resource.


file:///c:/users/sumanth sai/documents/visual studio 2015/Projects/ConcurrentFileAccess/ConcurrentFileAccess/bin/Debug/ConcurrentFileAccess.EXE

```
The process cannot access the file 'C:\Users\Sumanth Sai\Documents\visual studio 2015\Projects\ConcurrentFileAccess\Concurr
entFileAccess\Test\TR1.txt' because it is being used by another process.
Retrying to open the file
The Count is 1

The process cannot access the file 'C:\Users\Sumanth Sai\Documents\visual studio 2015\Projects\ConcurrentFileAccess\Concurr
entFileAccess\Test\TR1.txt' because it is being used by another process.
Retrying to open the file
The Count is 2

The process cannot access the file 'C:\Users\Sumanth Sai\Documents\visual studio 2015\Projects\ConcurrentFileAccess\Concurr
entFileAccess\Test\TR1.txt' because it is being used by another process.
Retrying to open the file
The Count is 3

The process cannot access the file 'C:\Users\Sumanth Sai\Documents\visual studio 2015\Projects\ConcurrentFileAccess\Concurr
entFileAccess\Test\TR1.txt' because it is being used by another process.
Retrying to open the file
The Count is 4

  <testRequest>
        <test name="test1">
        <testDriver>testdriver.dll</testDriver>
        <library>testedcode.dll</library>
      </test>
      <test name="test2">
        <testDriver>td1.dll</testDriver>
        <library>tc1.dll</library>
      </test>
      <test name="test3">
        <testDriver>anothertestdriver.dll</testDriver>
        <library>anothertestedcode.dll</library>
      </test>
    </testRequest>
```

In the above case, I set the Number of retries to 10. I was able to access the file after 4 retries. The retry is made after the made to sleep for some time(10s).

Conclusion

Through this document, a proposal is made how software development process can be effectively managed. The Document discusses extensively on high level design principles of the Software Collaboration Foundation. Any software development team who wants to set up SCF for their project can take implementation ideas from this document. The document only discusses high level ideas of the system and it's up to the team which technologies they want to use while building this system.

The document discusses about Organizing principles, Uses and Users, Architectural ideas, Structures, Activities and critical issues of each federated server.
While writing this document, it is kept in mind that the development of SCF can be done in reasonable amount of time with reasonable amount of budget. If any person who wants to implement the ideas are not clear about any part of implementation or they want to propose a better solution for that, they are welcomed to contact me.

References

http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project5-F2016.htm
https://msdn.microsoft.com/en-us/library/ms733033(v=vs.110).aspx
https://msdn.microsoft.com/en-us/library/ms733107(v=vs.110).aspx
https://www.safaribooksonline.com/library/view/programming-wcf-services/0596526997/ch01s04.html

Many references from taken from Dr Fawcett’s website apart from this. Many documents provided by him were used while building the architectural ideas of this document.