SOFTWARE COLLABORATION FEDERATION

CSE 681 PROJECT #5

Arpit Utpalkuamr Shah
SUID: 263387322
Instructor: Jim Fawcett
Date: 7th December, 2016

Table of Contents

1. Executive Summary

Software with thousand or hundredths of packages are being developed now-a- days. Therefore, development process of the software need more numbers of people to work on it with coordination. Software collaboration federation is collection of more than one client and server which is used to formulate coordination between single or multiple software development teams.

This architectural document essentially represents the internal interactions between elements of the Test Harness and the Storage Management Subsystem and; external interactions with other services and components of the SCF. Its main purpose is to support Continuous Test and Integration (CTAI) between large software development teams.

This is a very huge and complicated system comprising of servers like Collaboration server, Test Harness Server, Repository server, Virtual Display Server and multiple client server interacting with this system. We have discussed each and every server with its detailed behavior, its uses and its interactions with other server in terms of messaging communication. It also uses Storage management system(SMS) to organize the data and addresses optimization of efficiency and pace at which the data is accessed.

The users of this system will be Developers, QAs, Project Managers, Customers. Developers and QA will use this system for technical stuff like check in, downloading baseline, testing etc. Project Manager and customer will use this system on higher level to keep eye on development process and to improve the process to make their teams more efficient. We have described user and uses for each server in this document in detail.

We have faced critical issues like versioning, caching, performance, Communication latency, locking, file access, communication errors and etc. We have discussed each and every critical issue in this architectural document along with their potential solution.

2. Introduction

Software industry is growing very rapidly as well as the size of the team working on it. Software with thousand or hundredths of packages are being normal thing now-a- days. Therefore, it is impossible to work or develop one full package or module by one developer. Therefore, project managers or leaders started dividing a big development module into small chunks of work items, so that it can be worked simultaneously by different development teams rather than one by one. Because of this solution, problem of collaborations arises.

Other way around, software industry is scattered all across the globe where the market provides strategic benefit to the company and hence for an organization it is very difficult to operate on a single project from a single geographic location, and that's where the software collaboration Federation (SCF) shows its significance by providing perfect way for developing teams and managers/leaders to connect and synchronize their work on a project. Additionally, the SCF also provides remote accessing of the code, which includes maintaining the source code at remote locations on servers and testing that source code. This makes the software development process much more flexible and robust at the same time. So, Software Collaboration Federation (SCF) is a collection servers and associated software which is designed to support activities of a software development team.

It is used for Creating and publishing plans for development. This includes writing concept documents, creating and editing work packages, scheduling work packages, and allocating resources to work packages, preparing new source code packages and Acquiring existing source code packages for reuse. Since it contains test harness, it can be used to perform unit tests, integration tests, regression tests, performance tests, stress tests, and acceptance tests. It is also used to deploy software executables and documentation.

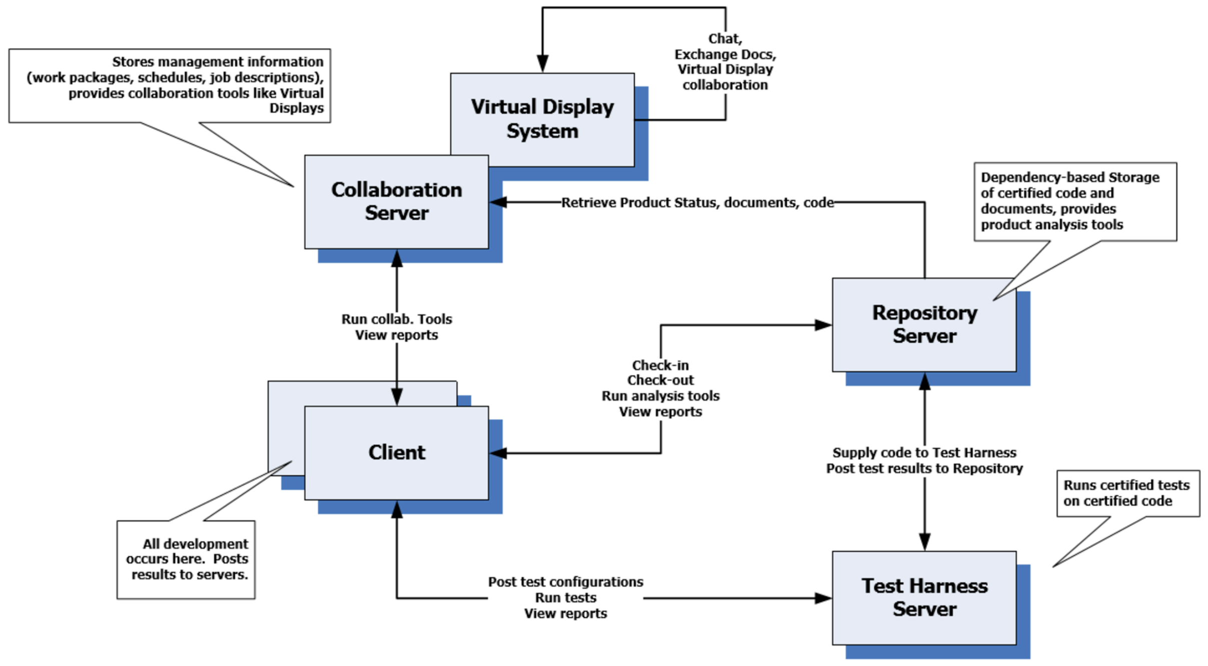
As it can be inferred from discussion above the WCF will have four major parts:

- Collaboration Server
- Test Harness Server
- Visual Display System
- Repository Server
- Client Server

All of these are discussed in good detail in this document.

3. Overall Architecture of the Software Collaboration Federation

Below is the high level overview diagram which depicts association of clients and different servers which makes up the whole architecture of Software Collaboration Federation(SCF). It also demonstrate the high level use of this server in the diagram.


Stores management information (work packages, schedules, job descriptions), provides collaboration tools like Virtual Displays
Chat, Exchange Docs, Virtual Display collaboration
Virtual Display System
Collaboration Server
Retrieve Product Status, documents, code
Dependency-based Storage of certified code and documents, provides product analysis tools
Repository Server
Run collab. Tools View reports
Check-in Check-out Run analysis tools View reports
Client
Supply code to Test Harness Post test results to Repository
Runs certified tests on certified code
All development occurs here. Posts results to servers.
Post test configurations Run tests View reports
Test Harness Server

3.1. Concepts

3.1.1. Support for collaboration while developing software

- Software Collaboration Federation(SCF) is developed to achieve a chief aim which is to get collaboration between different development teams working on same project or different package.
- Collaboration sever plays an important role which helps different developers to associate and coordinate with each other while development process. It stores data related to work packages and enables other collaborating services (like VDS services, SMS services etc.).

3.1.2. Support for Continuous Integration

- Software development is process of continuous integration now-a-days. No one is making full software packages and modules at one go. It will be divided in small work packages and solution (developed code) of that work packages are being integrated continuously with the whole system. It will continuous integration with testing, so that if some non-working code tried to get integrated with system package as solution of assigned work package, it will get tested by test harness first. If test gets failed, that in code will be integrated or checked in and marked as FAILED status.

3.1.3. Support for disclosing current state of baseline

- As we have discussed, WCF is designed for Continuous Integration while developing software. to provide this functionalities, it will be necessary to keep track of the information of current state of baseline. It has all dependency information about namespaces, packages, files with their working status. This all information will be saved in metadata files with other important information about developer who developed this code, documents etc.

3.2. User and Uses

3.2.1. Developers & Maintenance Team

Developers are the frequent user of this system. They use test harness as part of check in process and defect analysis. They use collaboration server to get their work packages organized which will work as medium which shows the actual progress of each task.

Maintenance Team will use this system frequently. Whenever QAs will log a bug by adding Work Packages, Maintenance team will resolve that issues, and checked in the changes with the help of Manger.

Mostly, this user may use it for Integration testing and Unit testing.

Unit Testing

When any module or piece of code has been developed, requirement of testing that code arises. Therefore, developers will create test driver for that new modules/codes and run that that test driver until they get expected result. Advantage of this testing is, we can use that test driver in future when the requirement of testing of the same module/code arise again.

Integration Testing

It is a testing, which will be easy to do if you have tool like test harness which will do those tasks for you. Whenever developers add any functionality or modules, they should check all the old modules, which are already there at the time integration. Therefore, it is very rigorous work to check all the module again and again manually whenever new functionality is added. Hence, Test Harness will be very useful for this.

Impact on Design

- We should make well designed client GUI, so that work for this user will be easier. Mostly, every activity is initialized by client which is done by sending message to other server. So, will provide automatic message creation by using deferent templates from SMS, which will reduce the load on client and it will be more efficient.
- Impact of this users, also motivate to improve the performance. Generally, developers are used to run small test comparing to QA and they cannot wait too long to get the result since they have to do code changes according to the result of that test if it gets failed. Therefore, implementation of threading might help to overcome this.

3.2.2. Quality Assurance

Quality Assurer are users who focused on maintaining the quality of the software. They have to perform different type of test to check each scenario which may occurs. QA will write different types of test suit (test driver) to perform different types of testing like, Performance testing, Stress testing, Integration testing, Quality testing. If they find any issue while testing, they will log an issue as Work package in Collaboration server.

Here I have discussed different type of testing which can be done by QA using this system:

Regression Testing

These tests are executed on the whole baseline to ensure that it is not broken due to any recent changes. These kind of tests are executed at regular interval. The interval can vary according to the development requirements and project management. But generally, these regression tests are executed daily. The quality assurance engineer can modify this test to test package more effectively.

Bug Detection Test

If any bug is reported for specific feature of particular version of the software. The quality assurance team can design test-suite to check whether that bug is solved or not. This test is ran after attempt to solve bug. This way, the system can be useful to supply bug-free software.

Stress Testing

The testing team can create stress test with heavy load to ensure that the software handles such heavy loaded situations and can recover from them. These kind of tests can be created by creating test vector generator to supply large amount of input. These tests can ensure the software’s performance.

Performance Testing

Before each release, testing team will run pre-developed Performance test suit to get the performance rating of the system. Performance testing is in general, a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload.

Impact on Design

- QAs will do very heavy testing on the baseline to get confident about product quality. So, test harness should be an efficient performer in this system. To tackle this situation, we will implement multiple test harness server and one Load Balancing server. Load balancing server will help client to find Test Harness server with least load.
- Apart from that, QA just have to give information about either test code or test driver. We will find the value of other dependent using metadata file of that test code or test driver. It makes easy for them to configure any test and run it.
- This user need very good UI since they are frequent user of this application. If UI is not very good, then they will start finding alternatives.

3.2.3. Manager

Managers will need the summary level information about project progress and status of each running task/work packages.

Impact on Design

We will create Agent on Collaboration server which will accumulate all the data about work packages and it also retrieve data from repository about test request and results. It will make one summary report of all the data and send that file to the Project manager at scheduled time.

Manager also can make changes in the work packages, like, adding work packages, removing it, filling estimated time for completion.

3.2.4. Software Architect

Software Architect are the designer of the system who decide architecture of the system and create some rules which has to be followed by each developer while working. It explains the whole architecture of the system to the team which helps team to be focused and binds the team to work on system.

He/she will be the one who mostly create or modify work packages and documents which helps the assignee of the task/work package.

Impact on Design

We should create easy and interactive UI to create work package for Software Architect. It can help him/her to improve work efficiency. We should save this all information in NoSql database since this is very massive volumes of new, rapidly changing data. Storing it in NoSql database will increase the retrieval speed which will help this user to work effectively.

3.2.5. Uses

- **Continuous Integration**
  SCF provides one platform for software development teams where developers can continuously integrate the files with new version names. All the team member can work on their work items without disturbing other member's developments. It is designed with one owner policy so that it won't have confliction issues between their files handling processes.

- **Easy project management**
  SCF implements a great collaboration system between bigger teams and it stores important data about work packages and its status. Managers or leader can activate agents who automatically integrate all the important information regarding project management, creates one file, organize all the information in it and send it to the leader/Project Manager daily at predefined time as notification.

- **Testing automation**
  It supports automated testing on each integration. Whenever any files gets checked in. system automatically start testing of that file and the file who dependent on that. It will send notification to the Manager and the person who have did this modification in that file.
  It also provides different types of custom testing to the users, like unit tests, integration tests, regression tests, performance tests, stress tests, and acceptance tests. User just have to create the test driver for the test he/she wants to execute. If he/she already has one, it can be executed directly.

- **Interaction using Virtual Display(VD)**
  It supports setting VD which is on collaboration server. Any of the User can set up VDS without any control. Each user will have tab to set up VDS. It provides option to start

meeting; to include the members; to chat; to open any file, documents, diagrams. It uses collaboration server's Virtual Display System to provide all this functionality.

3.3. Activities

The main functionality of SCF is to provide managed and organized working environment where all the developing individuals works with association and coordination of each other.
It has some servers like clients, repository, test harness, collaboration which communicate with each other and set up good collaboration so that each working client(users) can participate in

development process with full concentration without being concerned about organization or integration of the works done by whole team.

Since SCF is very big system which includes many other servers and subsystems, it performs numbers of activities. From all of the server it consists, Client is GUI renderer which mostly the initiator of all the activities. Therefore, user can use client and start any task or fire any activity or event.

Here, I have demonstrated high level activity diagram for SCF:

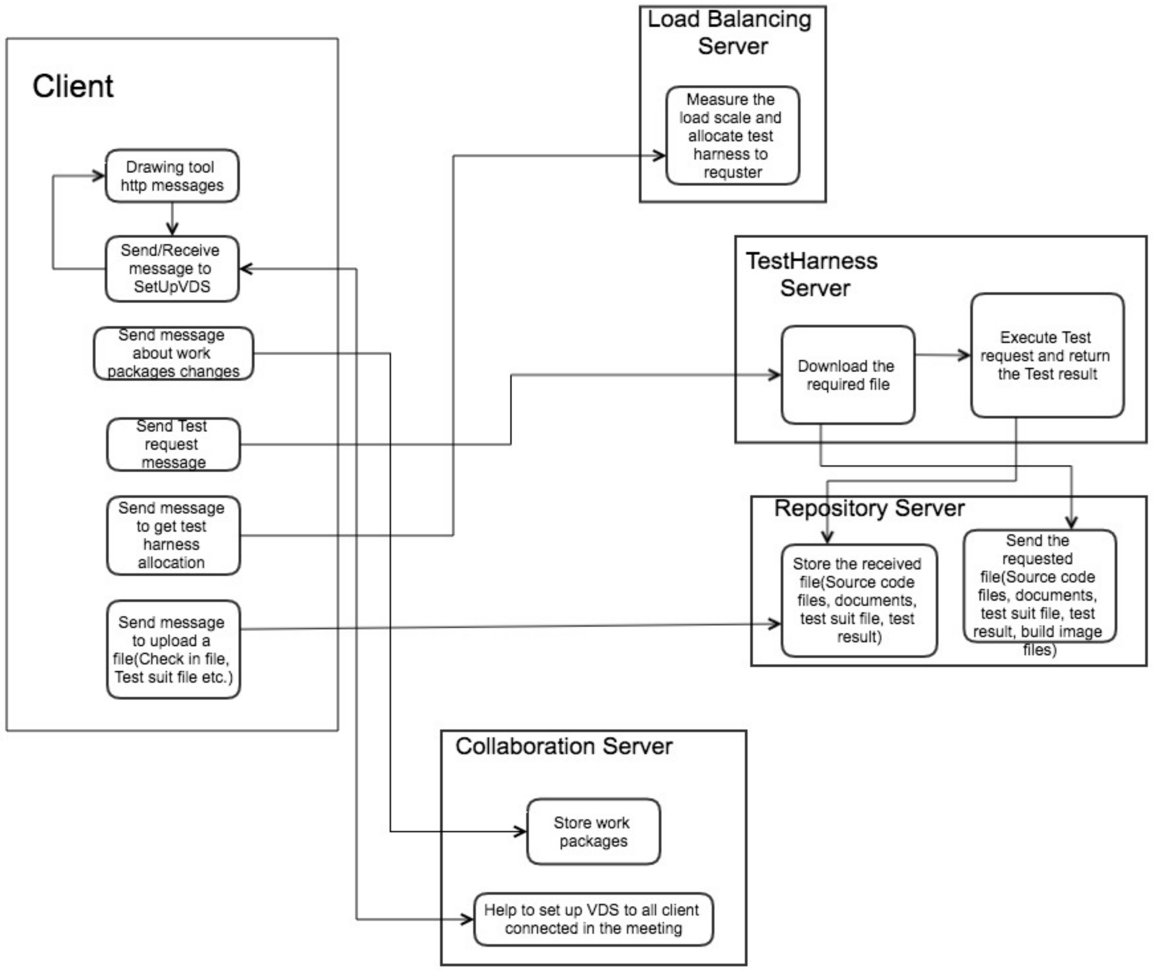

Client
Drawing tool http messages
Send/Receive message to SetUpVDS
Send message about work packages changes
Send Test request message
Send message to get test harness allocation
Send message to upload a file(Check in file, Test suit file etc.)
Load Balancing Server
Measure the load scale and allocate test harness to requster
TestHarness Server
Download the required file
Execute Test request and return the Test result
Repository Server
Store the received file(Source code files, documents, test suit file, test result)
Send the requested file(Source code files, documents, test suit file, test result, build image files)
Collaboration Server
Store work packages
Help to set up VDS to all client connected in the meeting

Figure 1 High level Activity diagram for SCF

Below, we are going to described all the activities briefly from above diagram:

- When user logged in using its credentials, the GUI will be rendered.
- If client can send set up VDS messages and use all the functionalities of VDS like setting meeting, webcam, chat.

- If client want to use drawing tool, it will connect directly to the other client's virtual display. It will not user Virtual Display System but, two clients will directly have connected using http message channel and use that drawing tool. According to design, only two clients can communicate using http channel. Therefore, any two clients can communicate concurrently on Virtual Display while using drawing tool.
    - Example like, if client A and client B is using drawing tool on Virtual Display and sharing their idea. No one other can join their meeting.
      But, at the same time, client C and client D can communicate using http channel and use drawing tool on Virtual Display.
- Developers can check in source code file with help of Managers. QA can check in test suits.
- User can send GetTH message and get Test Harness server with least load. It will communicate with it and send the test request to get it executed and get the test result. That test result will be stored on Repository and send to requester client.
- User can add, update and remove the work packages on collaboration server by sending message to collaboration server.

3.4. Partitions

SCF can be divided into different machine server and some subsystem. These partitions are listed below:

1. Client Server
2. Collaboration Server
3. Test Harness Server
4. Repository Sever

5. Virtual Display System
6. Software Management Subsystem
7. Load Balancing Server

Below is the structure diagram of client server which demonstrate connection between each package:

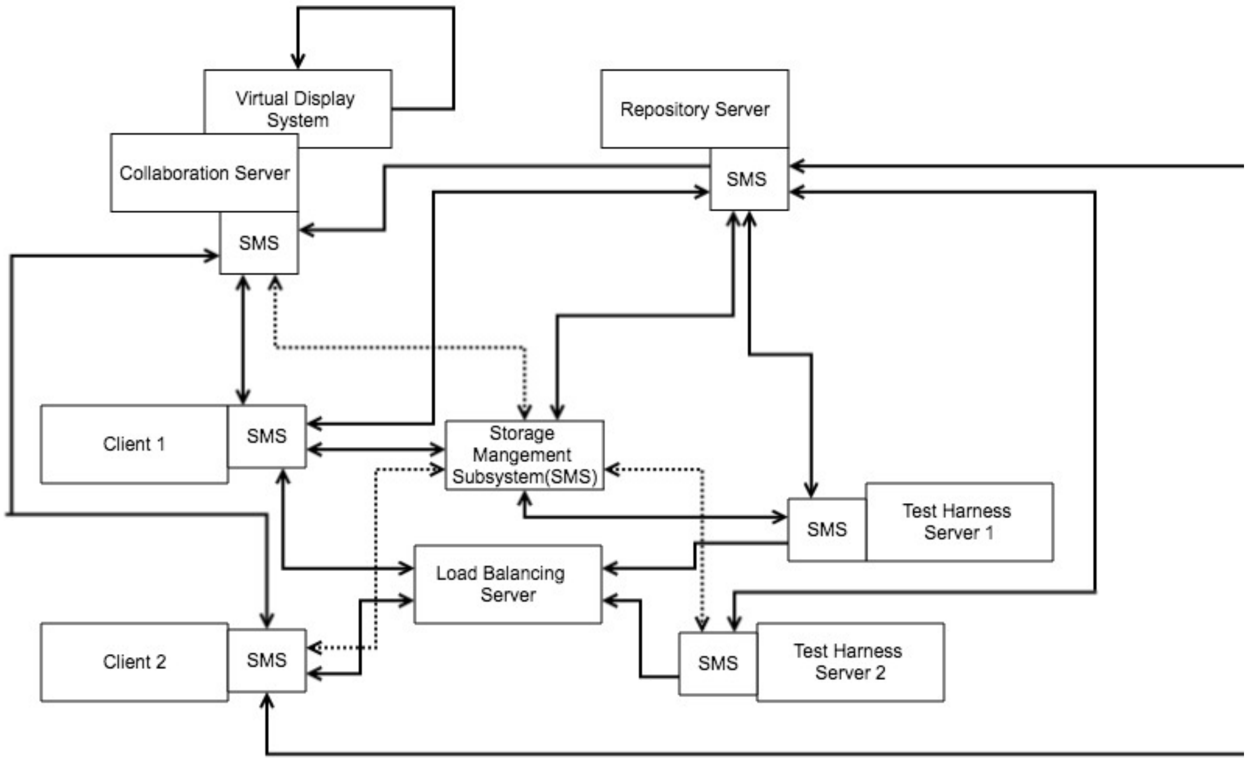


Figure 2 Package diagram for SCF

Here, we are going to describe each server from the structure diagram (dotted line at confliction of two line to remove the confusion):

3.4.1. Client Servers

This is the point from which our system starts. It helps to render easy and interactive GUI with which users can easily interact. It renders GUIs according to the user type and its allotted privileges. Since it is the server through which user can interact with whole system, client server has to communicate with every server (depicted in diagram posted above.) using SMS. It is used to give instruction message to other server to get work done by collaboration of all server.

Responsibilities:

Client server will have following responsibility which it will provide by using client GUI:

- check-in and check-out of code and documents
- creation and scheduling tests
- Querying repository for test results
- Querying different server for the log files
- Managing Virtual Display(VD)- It sends all setup and layout information to collaboration server like where to place the windows and what contents to display using SMS. The SMS at Collaboration server will store all the information related to layout and helps collaboration server to deal with VDS.
- Sending Test request- the main functionality is to send test request and get the result of that test request. But, for sending test request to test harness, client have to interact with load balancing server to get test harness server which is least busy. After that, it will communicate with that test harness directly by establishing the connection. You can see in the diagram, I didn't show any connection between clients and test harness, because load balancing server helps client to find least busy server to get their test request executed efficiently.

Interactions with other server:

It communicates with all the other server which includes collaboration server, Test Harness server, Load Balancing server, Repository server.

3.4.2. Collaboration Server

As its name suggest, it is used for collaboration or coordination between different parts of SCF. Therefore, it is the main part of software collaboration federation. It supports project management by storing work packages, work schedules, job description and by providing other collaborative tools. Work packages are small chucks of big work load to make it easy and organized. It would be easy to complete all this small work packages, merge them and package instead of creating one big work load in one go as single work package. Work schedules contains estimated time to complete that work package and current status of that task of work package. It also stores important dates about meeting, release deadlines and client demonstration date. It makes easy for project manager to keep eye on development process and enhance the productivity of the teams.

It also supports automated notification system using SMS event managers which creates backhand agent which collects all the data about progress of project development work packages, testing result of each packages and create one brief statistics informative file and send it to project manager as a notification on scheduled time. It makes project manager’s life much easier and helps to manage the project.

It closely works with virtual display system which is focused on managing display of collaborative work related to data. Data managers of SMS will get all the information about content to be displayed, place to put different windows. Therefore, collaboration server set up the VDS using the data SMS have. This VDS will help all the clients to set up their Virtual Display.

Responsibilities:
It is responsible to store storing work packages, work schedules, job description. It will give the information when requested. It also helps to create summary report of work packages status and baseline status and send it to Project Manager at scheduled time.

It also helps to set up main Virtual Display system which helps all the clients (who are in the meeting) to set up their Virtual Display.

Interaction with other servers:
It interacts with client server and repository server.

3.4.3. Test Harness server

Test harness server is the main server which supports the purpose of SCF which is continuation testing and integration(CTAI).

Responsibilities:
Its main task is to execute the test and provide the result to the requester. It gets the XML messages about test execution request. It deserialized the message and get information about test execution.
It closely works with repository server’s build manager to get required files images for test execution. After test execution completes, it sends the test result

message to repository to get it stored for future use and sends it to client to for display purpose.

Interaction with other servers:

It communicates with Repository server to get required image file necessary for test execution and to store the test result file.

It communicates with client to get test request, and also to send test result after test execution completes.

It also communicates with Load balancing Server to let it measure the load scale test harness server already have.

3.4.4. Repository Server

It is the centralized information storage factory for whole system. It stores all the checked in source codes and test suits with all the versions. Repository never delete anything but it organizes and manage each and every module and its files with its dependency information. Since it is the main storage system, repository used more extensively the SMS and its services.

It builds metadata for each source code which contains developer's information, dependency information, documents information etc. It stores that information using data managers of SMS.

Repository plays an important role to help teams and their way of doing business. It actually manages and organize all the important technical data about project according to its predefined pluggable policies for each operation.

Responsibilities:

Its major responsibility is to store the files and information; and send them when requested.

Interaction with other servers:

It interacts with test harness server to send build image file and to get test result file.

It interacts with collaboration server since collaboration server requests for current baseline summary information.

3.4.5. Virtual Display System

It provides help in setting up a sophisticated interface for publishing management information and collaboration activities that involve viewing source code, documents, diagrams, sketches, and webcams to enhance personal and team interactions. This is a chief vehicle for collaboration between remote teams to communicate.
It closely works with collaboration server to get data to display it on requested space using SMS, and send that data to all the participants of the meeting, so that they can display it on their Virtual Display(VD).

Responsibilities:
When VDS get set up information from SMS, it should create SetUpVDS message and send that information to all the clients who are participants of the meeting. Therefore, each will be look like real time screen sharing.

Interacts with other servers:
As we have discussed, VDS communicate with only collaboration server to get and send the data about Virtual Display(VD) set up.

3.4.6. Software Management Subsystem(SMS)

Software management subsystem(SMS). It is not an any individual entity or proper server which works for itself. But, it is composition of one or more nodes on each client and other servers, which is providing different utility based SCF services, where nodes are formed of NoSql DB to store and manage important information, communication services, and Templates for messages handling.

Responsibilities:
It is not any single unit with only aimed responsibility. It exists on every server, so it has different responsibility for each server. But, as a common, It is used to store message templates which is used in message creation to send that message. It also helps to get data back from that filled message template.

Interacts with other servers
It interacts with every server available in the SCF.

3.4.6.1. Central Software Management Subsystem(SMS)

- Central SMS is used as just data storage. We are storing the common data which can be share by the server in that central SMS.
- As of now in the central SMS we are storing, list of available servers in the SCF. Therefore, if any system wants to establish the connection with any server, it will look in this central SMS that whether this server is available or not.

Responsibilities:
Before connecting to the any server, SMS at the server side will fetch the information about the available servers. So, if server is not in the list, it won't try to send the message or try to establish the connection. It will directly send notification to the sender that sever is not available

Interacts with other servers
It interacts with every server's SMS in the SCF.

3.4.7. Load Balancing Server

It is an important addition to the SCF which improves performance. Whenever, it gets message from client to send the test request, it will measure the Load Scale Point for each test harness server (currently I have demonstrated two test harness server in my partitions diagram) and helps client to establish the connection with test harness server with least load, so that test request can get executed and test result will be available early.

Responsibilities:
It is responsible for maintaining the Load Scale Count for each server. According to that, it has to send address of the Test Harness, which has a least load, to the requester client.
Interacts with other servers:
It communicates with all the client to help them to find Test Harness Server with least load.
It also communicates with Test Harness Server to maintain their load status.

3.5. Critical Issues

3.5.1. Version Control over baseline

Since we are focusing on continuous integration, each time any of the changes done in any files, we will have to create new version of that file and set it into current baseline. But, creating new version with same file name and uploading that version will replace it with old file and we lost support to the user who is using old baseline

Solution

To manage the versioning, we have to create new directory and add that file with new version over there. Then, we also should keep create new metadata files for its parent package/namespace which will be depends on the new file but still point to its old file. This will give us full control over versioning like, if new version gets proved incorrect and lead to other problems in development, then we can easily roll back the changes if we have information available about old versions.

3.5.2. Security

Security will be the major concern since we are using communication module to transfer messages and we kept this channels open to get better efficiency over communication which can be done without any pass and check.

Solution

We will place login parameter to access this system. We will use this parameter first time to establish the connection. After the authorization of that connection, we will keep that connection open, so that it can get better efficiency over communication which can be done without any check and pass. By doing this, we are getting security with efficiency.

3.5.3. Error Handling

Error handling will be very painful for this big system. We have to consider each and every scenario so that nothing missed out and turns out into big fatal exception. But, if we fill this system with full check and pass then, the performance will be decreased.

Solution

Optimum and organized design can solve this problem. Let me give on example, if we design the test request creation in a way that user can only select a file from Repository, then we don't have to put check on test harness that whether file is available in repository or not since user can only send test request to test harness for the file if it is available in repository. This will save lots of time by eliminating case in which if file is not on repository, then test harness have to send a message to the client about unavailability of file. Therefore, this type of design for error handling will not interrupt information flow.

3.5.4. VDS latency in transmission while VD using Drawing tool

Whenever VDS is used to set up real time drawing tool on Virtual Display (VD) at clients, it faces some latency in transmission of drawn object on VD, which makes it unusable. This latency happened because it is getting message through WCF communication channel which uses blocking queue to handle sending and receiving of the messages, Additionally, WCF communication is also very slow which is not suitable for this type of application.

Solution

We can create one dedicated channel while client starts drawing tool on his/her Virtual Display. Then, we will use that dedicated channel only to send little small message for each dot(.) user is making to draw objects. We will use that dedicated channel for communication between those two VD instead of using that virtual display system and collaboration server for information sharing between Virtual Displays. We will send the information like place dimension, colure information and draw the same dot on same dimension on another end. According to this solution, only two Virtual Display can communicate if they are using drawing tool.

4. Storage Management Subsystem

Storage management subsystem is a core service of the software collaboration subsystem and it will provide critical services like storing data in managed fashion and communication between modules of the system. To provide these services, the SMS will have data manager, event manager and NoSQL database.

The main responsibilities of the SMS are as below:

- Define and store routing lists for notifications and messages of the system.
- Store layout information about the virtual display system
- Store the Dependency list for baseline and path to the metadata for each, which will make accessing them easy.
- Keep track of check-ins of the code in the repository.
- Store the Work Package dependency list and path to the work package data files, which helps to access them easily without any searching.
- Store and use the templates of messages to communicate efficiently between the modules of the system.

4.1. Concepts

SMS service provides important components which helps to manage all the activities related to Storage Management Subsystem:

1. NoSql database
2. Data manager
3. Event Managers

We can consider this data manager and event manager as a key idea who plays important role to make it serve as whole SMS system.

4.1.1. NoSql database

SMS has an important and full featured NoSQL database system having key/value database, it will allow performing manipulation on this data and will support remote access to this data. The main purpose of creating the NoSQL database is the limitations of conventional SQL database which has unacceptable performance issues while working with huge data sizes and it could be only solved by proposing a database which has no defined structure and could respond the incoming queries and data retrieval requests with same response time without any constrains of the size of the database, and hence the NoSQL database is used as the major and only storage component of the system.

4.1.2. Data Managers

Data managers is an important part of SMS service. It implements a generic key/value in-memory database with specified parameters under the metadata; it does different types of manipulations on this data such as addition and deletion of the key/value pair, editing of values of the parameters. Additionally, it provides functionality of storing and retrieving the in-memory data to and from an XML file.

4.1.3. Event Managers

It will use data managers to access message templates and to keep track on a logged event. When any configured or pre-defined loges added in the log file, event manager will get notified about that and they will do the pre-defined activity configured with that event. Example like, sending a message, invoking any agent to do some work.

The above figure shows the higher level figure of the storage management subsystem, and as shown in the above figure, file will be stored on local memory of the server. The NoSQL database will just store the references to that files and will retrieve the files and send them to requestor using the communication module.

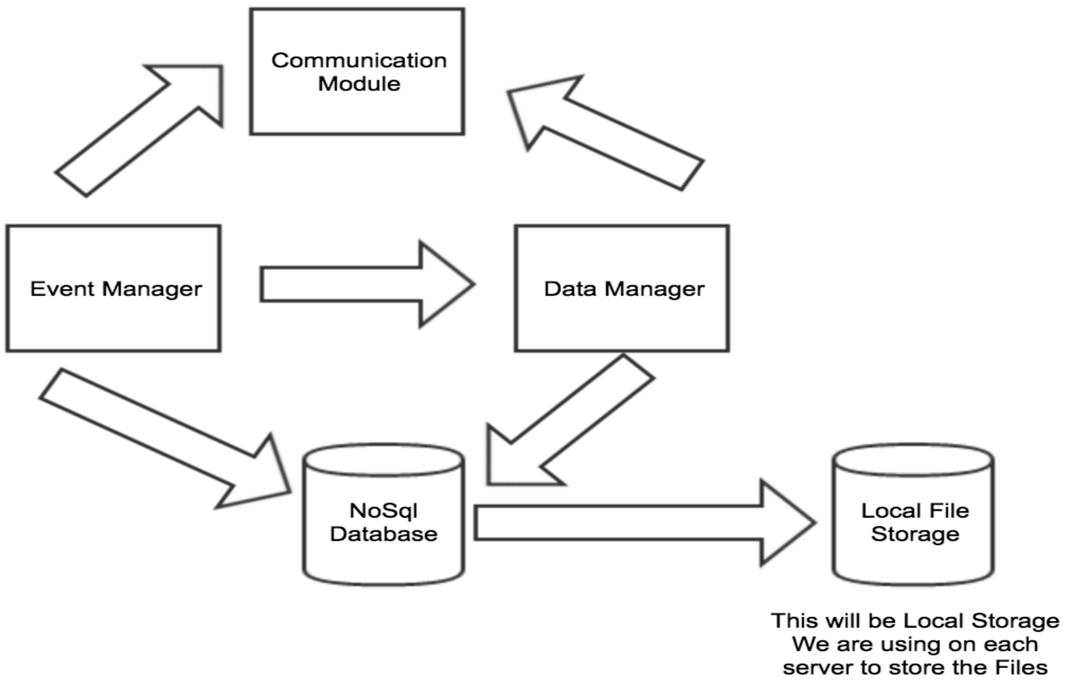


Figure 3 High level description of SMS managers and database

4.2. Structure of SMS

The below figure shows the generalized package diagram of the storage management subsystem. We have used generalized words for different servers on which they implemented.

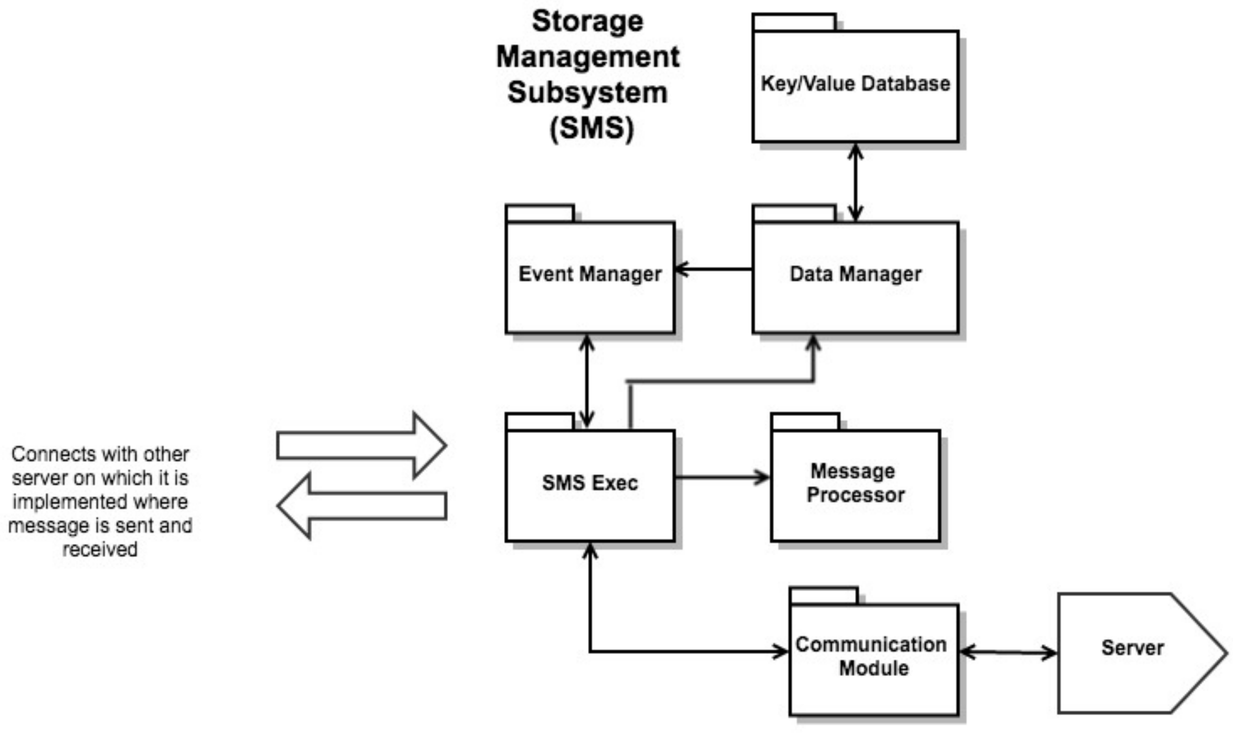

Storage Management Subsystem (SMS)
Key/Value Database
Event Manager
Data Manager
Connects with other server on which it is implemented where message is sent and received
SMS Exec
Message Processor
Communication Module
Server

Figure 4 Generalized package diagram for SMS

4.2.1. SMS Executive

This is the central member of the whole system.

Responsibilities:

it is responsible for coordinating with other packages in the system to provide services to the requesting servers and clients. It mostly communicates with the server on which SMS in implemented.

Interacts with other packages/servers:

This will use all the other packages available in the system such as data managers, event managers and comm modules.

It will also communicate with some package of the server on which it is used. Example like, if it is used in Repository server, then it will communicate with Repository Exec.

4.2.2. Data managers

Data manager is very important part of the SMS which allows performing data manipulation operation, data retrieval operation.

Responsibilities:
The responsibilities of this data managers is to implement a generic key/value in-memory database with specified parameters under the metadata; it perform different types of manipulations on this data such as addition and removal of the key/value pair, editing of values of the parameters. Additionally, this data manager provides functionality of storing and retrieving data to and from an XML file. It also supports different types of queries such as getting values or the children of a specified key, getting set of keys which matches specified patterns and also getting keys for a defined time stamp.

It also responsible for fetching message templates while SMS exec get some data to create message. SMS exec will create message with the help of message processor and fetched message template.

Interacts with other packages/servers:
It communicates with Event manager since event manager get notified because of event logs data manager has. It also communicates with SMS exec, NoSql database.

4.2.3. Event Managers

Event managers helps in generating notifications and sending them to the server using email services or WCF communication.

Responsibilities:
Event managers responsible to invoke configured agents or to do some process when some predefined event happens. It uses data manager to access a message templates and also to keep track on the logged event, so that it can be invoked to do some work for some pre-defined events.

Interacts with other packages/servers:
It interacts with the data manager, SMS exec.

4.2.4. Message Processor

Message Processer is used by the SMS Exec for creating notifications for the event managers and for creating messages for overall other communications, it will use message template in NoSql database for its operations and this will help it in creating messages at faster pace.

Responsibilities:

It is responsible for creating message from data using Message templates and message processer. It also responsible for getting data back from messages by processing those messages with the help of appropriate Message templates.

Interacts with other packages/servers:

It communicates with just SMS exec.

4.2.5. NoSQL key/value database

It would be used by the message builder to access the predefined messages in the storage, it would contain xml with blank tags and the data is filled in it by the event manager or the SMS Executive. Example like, if user Repository gets TestResultSearchQuery message, then it will find all the inoframtion and fill the blank with the result and send the same message, which we tell Result TestResultSearchQuery message.

Additionally, it can be used to store different kind of data on every server.Example like, If we use it Repository server then, we will use it store Dependency list information. I If we are implementing it on Collaboration Server, then we can use it to store the WorkPackage Dependency List.

4.2.6. Communication module

It is an important part of this subsystem as it will be used by the system to pass the messages from one place to another.

Responsibilities:

It will be responsible to send and receive message. When we implement SMS on any server, we can use this messaging service of SMS for that server, so that we do not have to implement separate communication module foe that servers.

Interacts with other packages/servers:

It will interact with SMS exec to get messages to send and give received messages back to SMS exec.

4.3. Messaging templates in SMS

We will be using the XML formatted messages to communicate between the servers and clients as the XML file is uniquely structured and could be modified according to needs. There are different types of message structures available in the SMS to be used as the messages and each server and client have their appropriate message set to form a correct type of message.
Generally, this server will have following type of messages. We have described each message XML below:

- **CheckInRequest Message**:

```
  <Message>
<Type> CheckInRequest </Type>
<PName>(Package Name)</PName>
<Email>(Uploader Email)</Email>
<Date>(Date Of upload)</Date>
 <CheckInOldPackage>(On which package/file you want to do checkin)</ CheckInOldPackage
>
  </Message>
```

- **FileQuery:**

```
  <Message>
<Type> FileQuery </Type>
<PName>(Package Name)</PName>
<IncludedDependnecy>(Is dependent files needed with this?true/false)</
IncludedDependnecy >
<Email>(Requester Email)</Email>
<Version>(Version No of the requested package)</Version>
</Message>
```

- **DependencyQuery:**

```
<Message>
<Type> DependencyQuery </Type>
<PName>(Package Name)</PName>
```

<ResultDependency>(Will be filled when result message will be sent with list of dependent package this package have)</ ResultDependency >
<Email>(Requester Email)</Email>
<Version>(Version No of the requested file)</Version>
</Message>

- **TestRequest**

```
<Message>
<Type> TestRequest </Type>
 <Email>(Requester Email)</Email>
<TestRequest>
<author>Arpit Shah</author>
<aut          hortype>Developer</authortype>
<test name=(name of the test)>
<testDriver>(name of the test driver)</testDriver>
<library>(name of the test code)</library>
</test>
<test name=(name of the test)>
<testDriver>(name of the test driver)</testDriver>
<library>(name of the test code)</library>
</test> </testRequest>
```

- **LogQuery:**

```
<Message>
  <Type> LogQuery </Type>
  <Email>(Requester Email)</Email>
</Message>
```

- **SetUpVDS:**

```
<Message>  <Type> SetUpVDS </Type>
<Email>(Requester
```

Email)</Email> <EEmail> <Emp1></Emp1> <Emp2></Emp2> . . </EEmail >
<LayOutInfo>(Will have layout information about all the windows open on VD)< /LayOutInfo >
<OpenFile><FileName>(file name)</ FileName ><VersionNumber>(version name)</ VersionNumber ></ OpenFile >//(If someone has opend file put the filename with version number)
<ListOfToolEnabled>()</ ListOfToolEnabled >//to keep track if drawing tool is enabled, we will ask client to have just one more client to communicate with
<WebCamView>(address of all the client who is in the meeting with webcam)< /WebCamView >//using that unique address we can diffrenttiate between different Virtual Display
</Message>

- **CheckIndecision**

<Message>
<Type>CheckInDecision</Type>
<Decision>True</Decision>
<CheckInObjectPath>(Path on old object on Repository)</ CheckInObjectPath>
<DevelopersInfo>(Developer Email)</ DevelopersInfo >
<Date>(Date Of upload)</Date>
<DocumentPath>(Path of the Document)</DocumentPath>
<RelatedWorkPackage>(Work Package Id)</ RelatedWorkPackage >
</Message>

- **GetTH**

<Message>
<Type> GetTH </Type>
<TestHarnessAddress>(Result Test Harness Address with least load)</ TestHarnessAddress >
</Message>

- **TestResultQuery**

<Message>
<Type> TestResultQuery</Type>
<Name>(Test Result file name)</Name>

<Email>(Requester Email)</Email>
</Message>

- **TestResultSerachQuery**

  <Message>
<Type> TestResultSerachQuery</Type>
< Keywords >(Keywords to search the file)</Keywords >
<ResultListOfFile>(ResultList Of file sent as a string separated by comma(;))</ ResultListOfFile
>
</Message>

- **WorkPackageChange**

  <Message>
<Type> WorkPackageChange</Type>
<WorkPackageXML>(It includes all the information about work package)</ WorkPackageXML
>
</Message>

- **WorkPackageInfo**

  <Message>
<Type> WorkPackageInfo </Type>
<Id>(Work package Id)</Id>
<WorkPackageXML>(It includes all the information about work package as a Result)</ WorkPackageXML >
</Message>

4.4. Communication

Communication is an important part of the whole system and it is also a big part of SMS as the data and event managers use this services to communicate between different modules. All servers in this Software collaboration system needs to interact with each other and this will be done by the communication module. There could be different types of messages but here we are dealing with mainly messages with XML format. Each client and server will have their own dedicated address for communicating with each other and these address would be unique. The communication in this SCF is built using the Windows Communication Framework (WCF). WCF enables to send data as asynchronous messages from one service endpoint to another, and asynchronous messaging system is required as it is a very heavily featured and loaded system which will have tons of data processing to do, and hence this messaging system is made of asynchronous.

4.4.1. Data Contract & Service Contract

To make the communication possible between the components ServiceContract, OperationContract and DataContract are defined.

The data contract for this system could look something like below:

```
public class Message {
[DataMember]
public string fromUrl { get; set; }
 [DataMember]
public string toUrl { get; set; }
 [DataMember]
  public string message { get; set; }
 [DataMember]
public byte[] fileinformation { get; set; } }
```

This data contract is explained as below:

- First two data members are toURL and fromURL these are very essential as they will decide the direction of the data. And they are defined by incoming message as it would contain the tag for the address.
- Message data member will contain the XML message in form of string and for this data member a well-defined message contract is required as discussed in the messaging in the SMS.
- The last data member is in form of byte and it will contain the accompanied files like code files, log files, test files and test results. They are sent in byte by byte in small data chunks.
- We are using thread safe blocking queue to send and receive the messages, so this will be efficient implementation of WCF.

Here I have demonstrated small diagram about how WCF working with Sending and receiving queue of SMS.

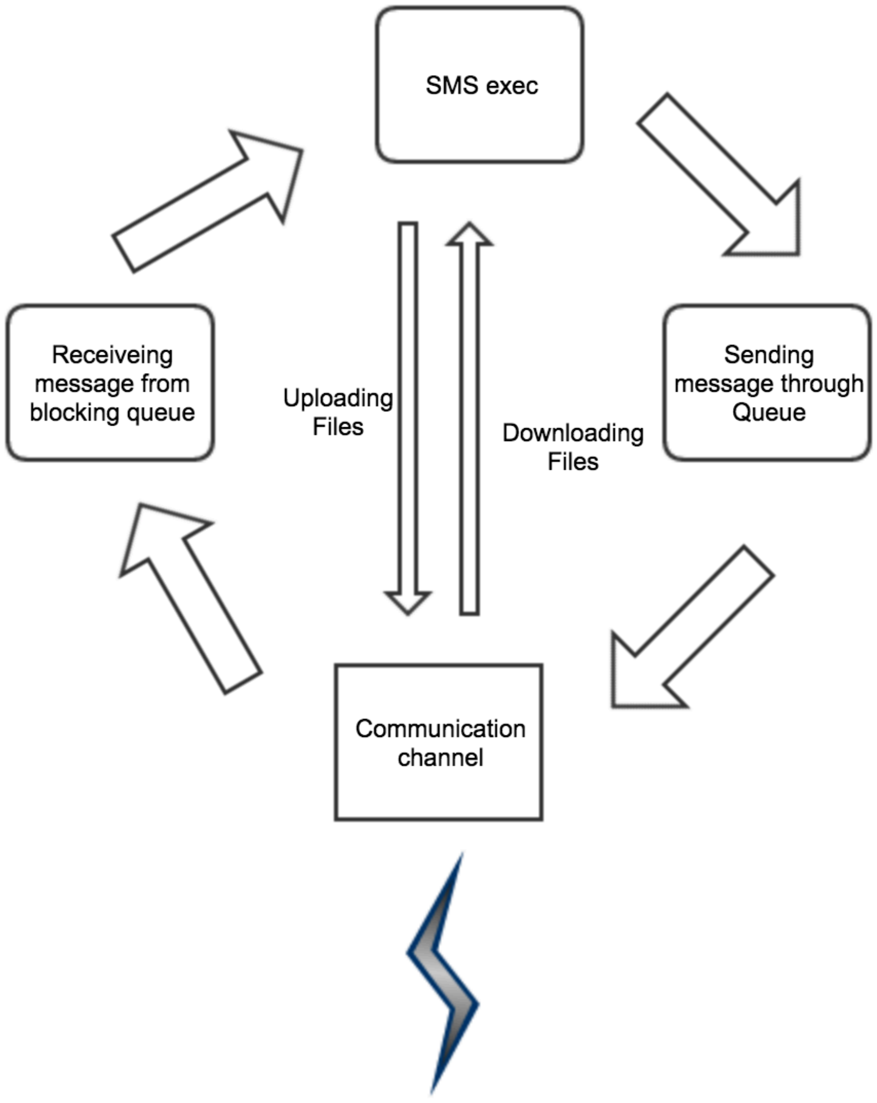

SMS exec
Receiveing message from blocking queue
Uploading Files
Downloading Files
Sending message through Queue
Communication channel

Figure 5 WCF communication in SMS

- It will use sending and receiving queue for simple messages communication.
- For files communication, it will directly upload or download the file using communication channel. It will not use queue for that.

Here is the service contract it will use do all this operation:

```
[ServiceContract(Namespace = "Project4")]
public interface ICommService
{
```

```
                                        [OperationContract(IsOneWay        =        true)]
    void sendMessage(Message msg);

                                        [OperationContract(IsOneWay        =        true)]
    void upLoadFile(Message msg,string receiverDirectoryPath);

                                                                     [OperationContract]
    byte[] downloadFile(Message msg);

  }
```

SendMessage operation: will send the message in the using blocking queue.
For receiving message, we will have inner implementation while implementing Service Contract.
UplaodFile operation: It will directly upload the file at the given location without using any queues for communication.
Download Operation: It will download the file using file byte without using any queues for communication.

4.4.2. Critical Issues

4.4.2.1. *Locking*

- When more than one clients are trying to manipulate the same data and get the original data then they might get wrong data as a result. Hence we should implement some system to lock the access of a data while it is being used by one client.

  Solution: We can use some bit like flag with all the data bytes, when a data byte is being accessed for any reason the flag bit would be set, and no other request would be able to access that byte.

4.4.2.2. Communication Errors

- We rely on some kind of communication channel to transmit messages between client and server. There could be some inherent issues which could make the message erroneous.

  Solution: We can use check with the messages to check it at the receiver side. If the message has errors we can have a system which can request the sender to send that information

4.4.2.3. *Loosing message if the server is not up*

- When the server is not yet initialized but the client starts shooting messages at the server then these messages would be lost and error could occur.

  Solution: We can implement Dead queues to solve this problem. We have senders which have dictionary and that contains a URL and a proxy. Instead of using value as the proxy, we could create a pair, one of which is proxy and other is a queue which will contain all the messages which were not able to be sent. Hence this dead queue will dispatch all the messages when the server is available and a successful connection is made.

4.5. Critical Issues of Storage Management Subsystem

4.5.1. Service Integration

- SMS is a subsystem of the Software collaboration federation and so, integrating it with efficiency is very essential. If the SMS is not integrated properly then the system can malfunction, which is not acceptable when we are creating system of industry grade.

  Solution: Throughout the document we have discussed various working principle and discussed how the system is going to work in detail, all these concepts should be followed rigorously and implemented in order to make the services integrated with SCF seamlessly.

4.5.2. Security

- In this SMS if no security checks are implemented then, the repository server data would be vulnerable to the unwanted access, this is dangerous if the repository is used for storing some sensitive information about the on-going project.

  Solution: Secured login is provided at the client side and this will filter the unauthorized access from the employees but for making the system hundred percent secure, the files should be paired with the login and password and these tags should be used to sort the data in the repository and only that data should be allowed access upon and hence a user cannot mess with the protected files of other users, which is an essential need in the secure system.

4.5.3. Inconsistency in message structure

- "Messaging templates in SMS" section discusses the structure of messages which are passed in the system but the structure of these messages are not coherent at the level it should be to qualify as fully optimized and consistent at industrial standards.

  Solution: Messaging system could be made much more efficient than it currently is by using the tags which are more coherent to each other across all the message type, this would make the parsing of the system very easy and make reading the message efficient.

5. Client

- All the users can access all available features of SCF using client. So, it is client's responsibility to make all features of the system available to the users. It will be the face of our system with which users will talk with.
- Therefore, it needs to have effective GUI to serve different kinds of users. If the client doesn't have effective GUI, it can be tiresome for the users to interact with the system. The client server has to communicate with all other servers in order to serve the users' requests.
- Effective representation of the received results from various servers will be also very important. The communication errors and errors from servers must be handled properly and displayed in GUI. Therefore, we will update results on respective tab of GUI from which

request is sent and we notify the user about the change using notification service on main page. All of this detailed description is covered on this section with demo GUI presentation to give better idea and understanding.

5.1. Concepts

- Client is the package which is used to interact with this whole system. It is the extension of all the services provided by other servers. Example like, Repository provides check in service in SCF. But client will provide the actual GUI tab for this specific process which will allow user to do this check in process on repository. So, client will provide extension of every service which is actually provided by some other server.
- Since client is providing extension of every service, it communicates with every server to invoke the actual service with the input parameter it gets from GUI.

5.2. User and Uses

5.2.1. Developers

Developers can use this server to perform a process like check in the code files, to get dependency of the files, to download the source code, to test their changes, to get test result, to use virtual display for interaction, to add or modify the work packages. All of this process will be perform on some different server. Client server will just work as medium which helps to initialize this process on different serve by sending a messages.

5.2.2. QAs

QAs can use this server to perform to upload test suit for different kind of testing. They will majorly use this server to start heavy loaded test process by sending test request to test harness. QAs will use virtual display for interaction between the team members.

5.2.3. Mangers

Managers will use this server to get actual progress of the work packages. It will also use it to accept check in request coming from different developers. Manager will use virtual display for interaction between the team members, start team meeting etc.

For all this three users,

Impact on Design:

GUI which is implemented on Client server should be light. GUI loaded with heavy process will create issues while using it. We should always run process on different thread instead of main GUI thread. It will make it light and responsive.

Apart from that, client GUI should be interactive. It shouldn't be hard to use and understand for the users.

5.2.4. Uses

This server provides GUI which have different service and functionalities which is enabled as per user. Each user has some user type attribute. According to this user type, he/she has some privileges on software. Therefore, tab will be displayed after logging in as per that user privileges. Example like, if Project Manager log in to this system, there will be no meaning to show him the functionality tabs which are related to core testing of the software.
Here, I have described all the uses of the client server:

- **Notification**
  Client GUI will have many tabs which is allocated to some predefined work, so it is impossible to see all the tabs at a time. But, data information and status are keep changing because of synchronous process running on each tab. So, Client GUI will have notification tab which will give notification on every change. User will get real time notification of each process running on each tab of GUI.

- **Send check in Request**
  It will allow users to upload file with change or new file itself to the repository which will consider as check in request of that file. User have to upload code files along with test packages and documentation. User also have to select file from repository for which you are uploading the new version. After that, the request will get generated and sent to owner (Project Manager/Leader). Owner is the only person who can make any changes in the code according to the single owner policy.

- **Download Code**
  It will provide the facility to download the code from repository. User will have option to select the code version and download it. So Every user can download the copy of that actual source code on their system and work on it, but only single user can change it in the actual copy of the file which is in repository.

- **Dependency check**
  Whenever any developer is working any file on local copy, it is necessary to take care about the dependency of other packages on that file. So that, it won't break any

previous functionality after check in. So, the system can be used to check dependency of the package in current baseline.

- **Testing**
  The system is focused on continuous testing integration(CTAI). It provides very good facilities related to test automation. It provides functionality to perform each type of testing on the any package of current baseline. This will be used by developers and QA to run test execution on the system.

- **Download Log files**
  This will be very important feature of the system. It helps to get information about server activity when any server is facing any problem or its not properly working. User can download the log file of any server and debug the problem server is facing during execution.

- **Get test result**
  Whenever test execution is completed, requester will get notification about it with test request result. User can get test result files by just clicking on that notification. User can also get test result files for the old test request by querying the repository.

- **Interaction using virtual display**
  This is a collaborative activity for the user. Therefore, user(client) can use their own virtual display(VD) and set it up using Virtual Display Server(VDS). They can use VD for chats, real time screen sharing, opening documents, code files, chats.

- **Check In**
  Since the system follows one owner policy, only one owner can check in the code in main baseline. That means, only single owner has authority to do changes in any source code files or any documents related to baseline. Developers will just send check in request with their changes to the owner/manager/leader, the final decision is on them to accept the changes or not. They will do the code review of the changes and if is appropriate than they will accept it as a check in code.
  If they find some wrong thing, they deny the check in request with some correction comments, which will be notified the developers who has sent check in request.

- **Assigning the work packages**
  User can access any of the work packages available in the collaboration server and assign it to their self. User also have to keep it updated while working on that package

so that it reflects perfect progress and status of the task.

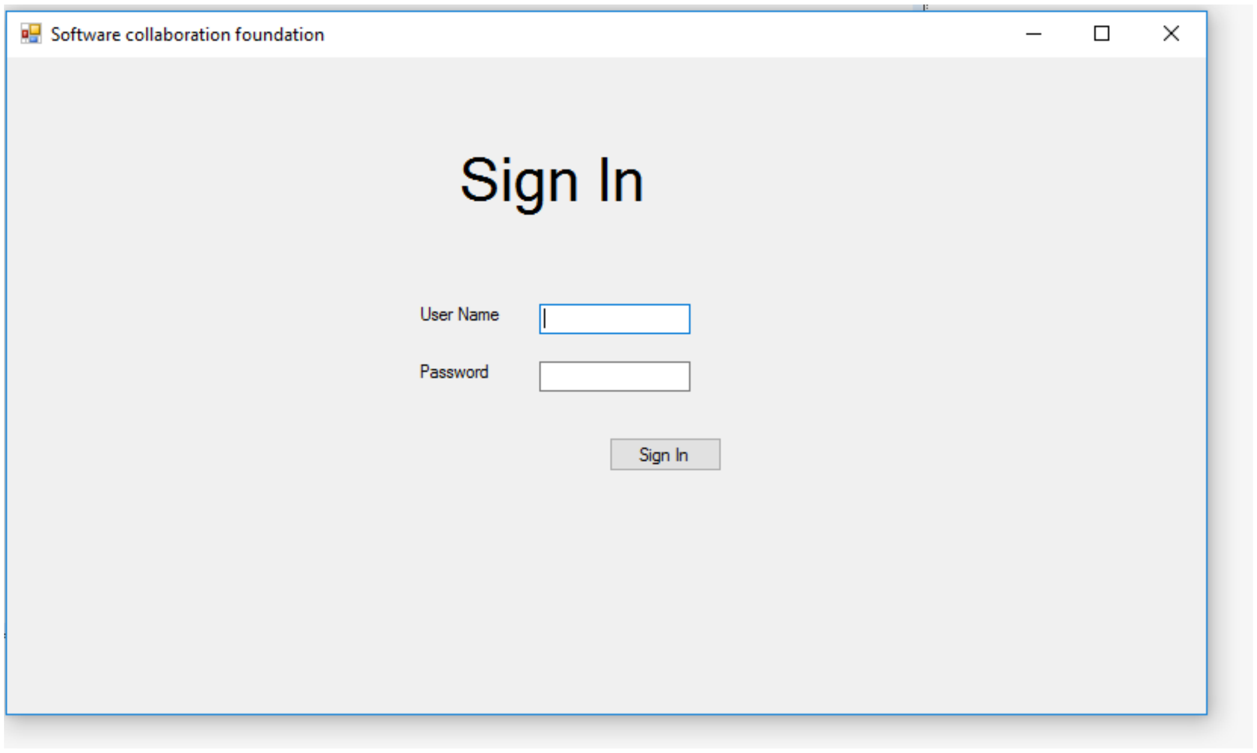
5.3. Views

Below I have attached screenshot of the GUI I have created using WCF, and description of each of them.

5.3.1. Log In screen

This is the log in screen of the system, in which user have to enter his/her user name and password as credentials. Client server will get the information about the user identity and provided privileges with the help of other server and render next UI on basis of that user identity.

Below is the screenshot of the Log Screen:

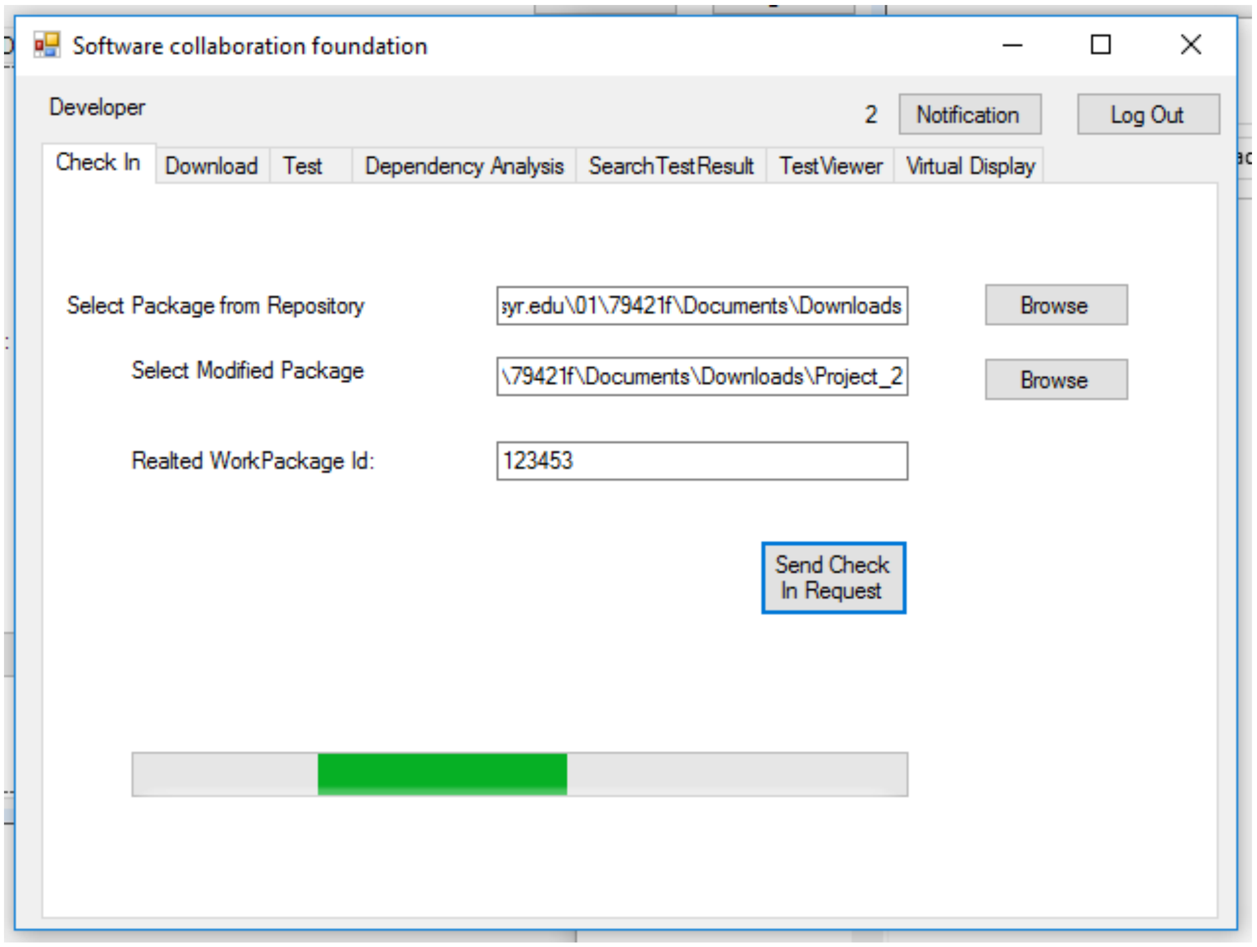


I have defined three types of user for this system, which are Developers, QAs and Project Managers.

5.3.2. Developer's GUI

Developer will have following functionalities:

1. **Check in Request**
   In this, developers have to select old file/package from repository, new file from local storage, realted work ID under which this modification in the code is done. After that, developer have to click on button Send Check In request.



Select Package from Repository: syr.edu\01\79421f\Documents\Downloads

Select Modified Package: \79421f\Documents\Downloads\Project_2

Realted WorkPackage Id: 123453

Figure 6 Check In request window

2. Download

Developers can download the code from repository. User has a two option:

1. To download the latest code

   In this user just have to select radio button and click on the download button. The latest baseline will be downloaded.

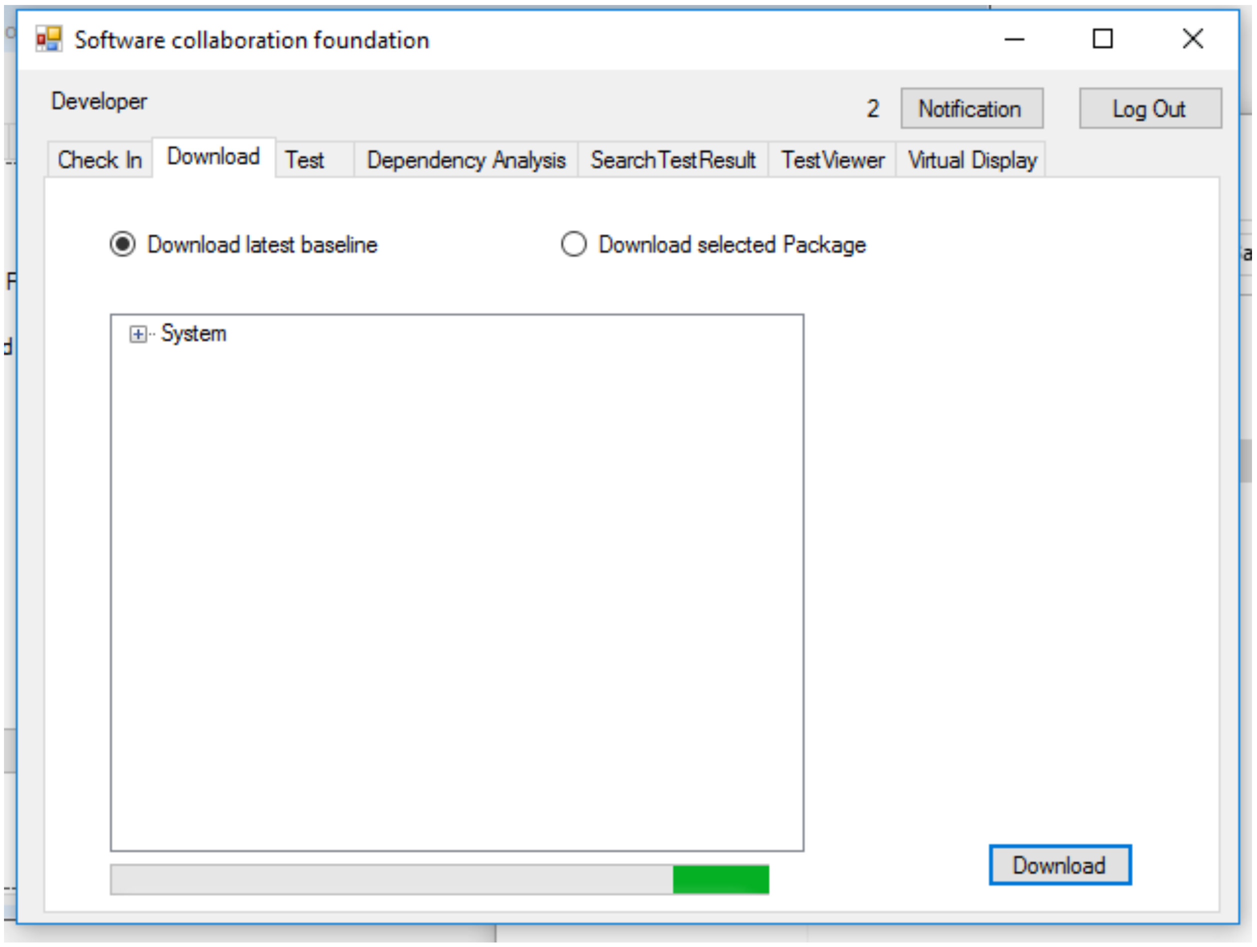


Figure 7 Download the latest baseline window

2. To download the selected code of selected version
   For this, user have to select the file he/she wants by exploring baseline using tree view and then they can download it.

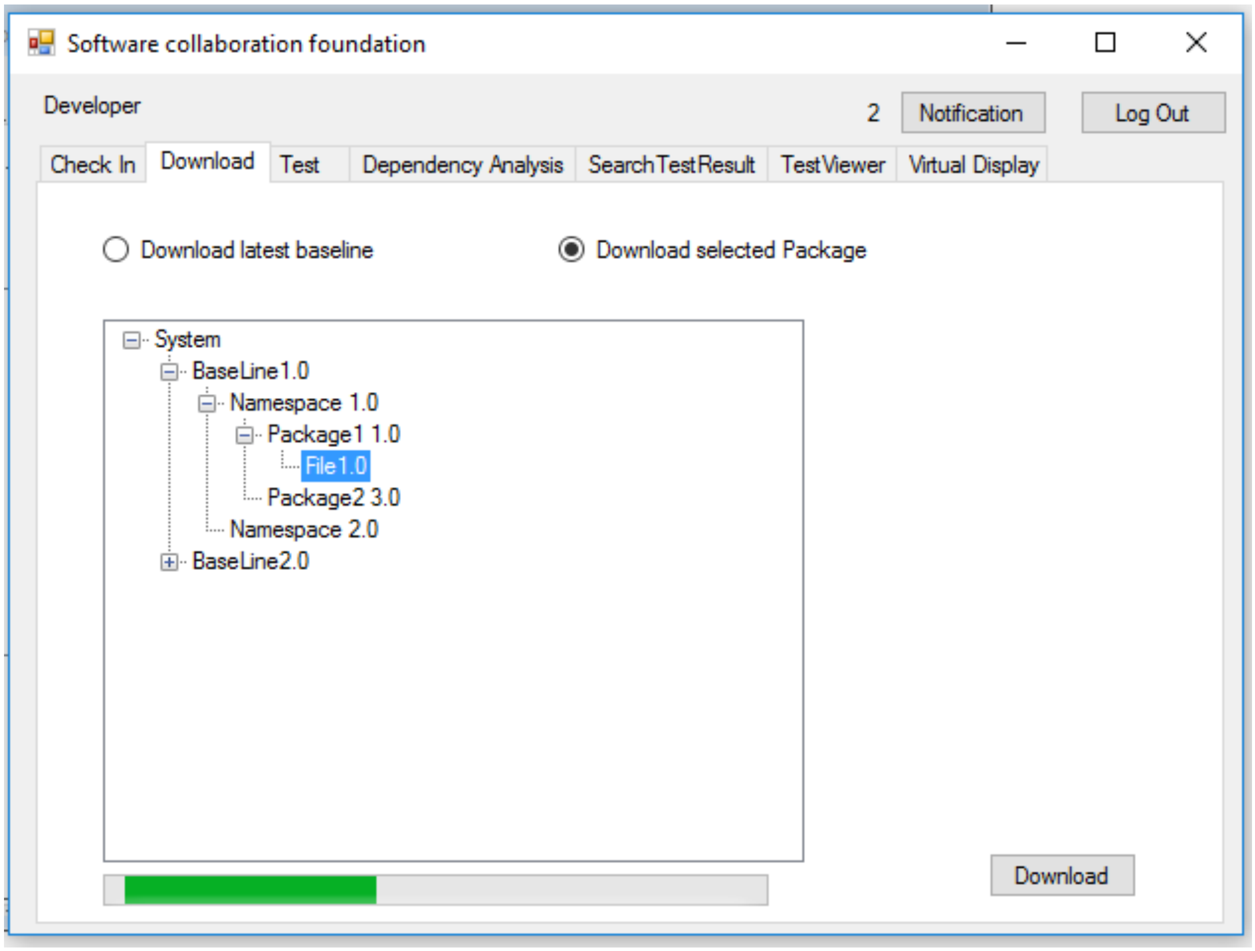
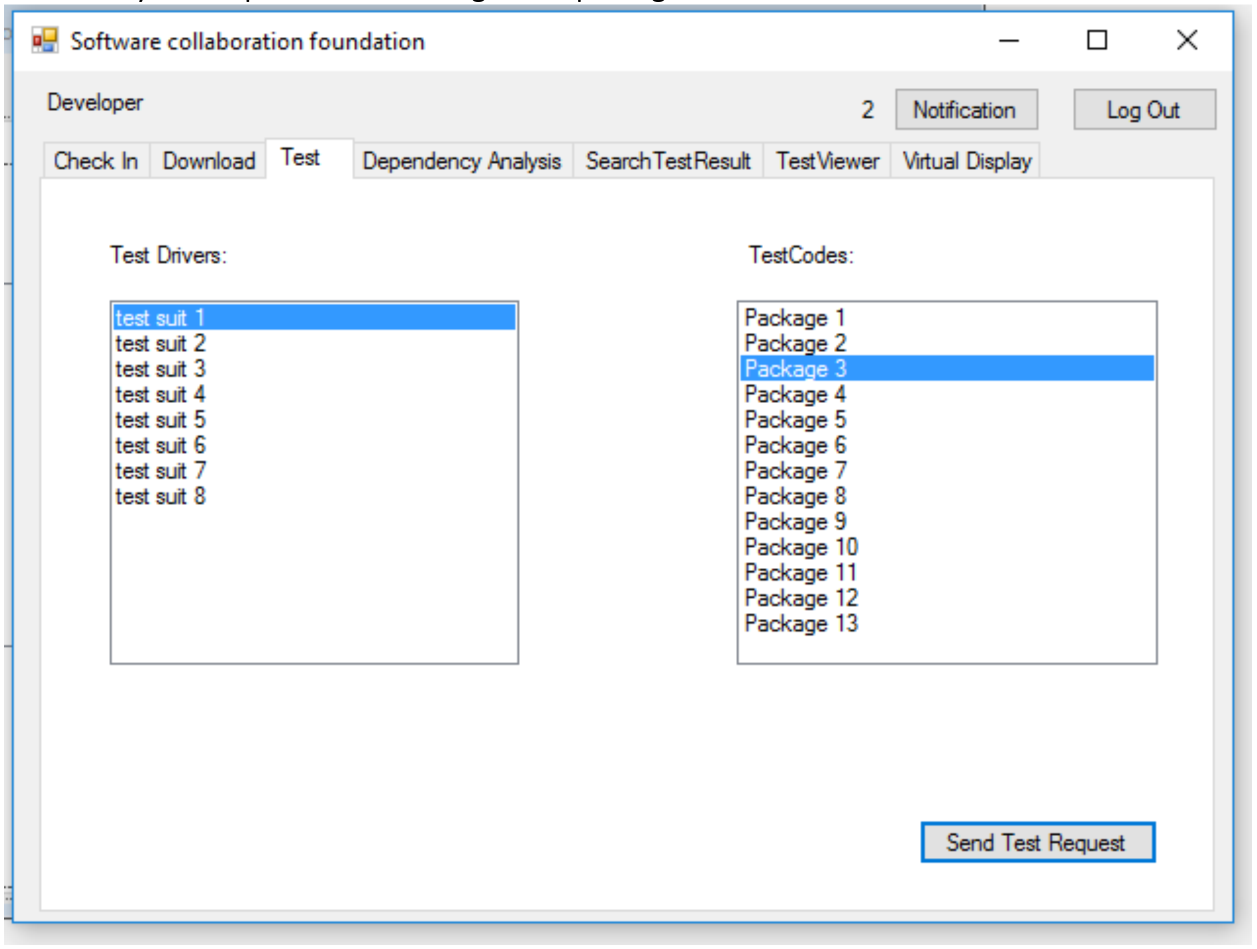


Figure 8 Download the selected code from baseline

3. **Test**
   User have option to select multiple test driver and test code. If user just select test driver, then system will automatically find the dependent files and run test execution on it.

If user just select the test code, then primary test execution only run on that test code. Primary test execution is the test execution done by using the test driver which was checked by developer while checking in the package.



Test Drivers:

test suit 1
test suit 2
test suit 3
test suit 4
test suit 5
test suit 6
test suit 7
test suit 8

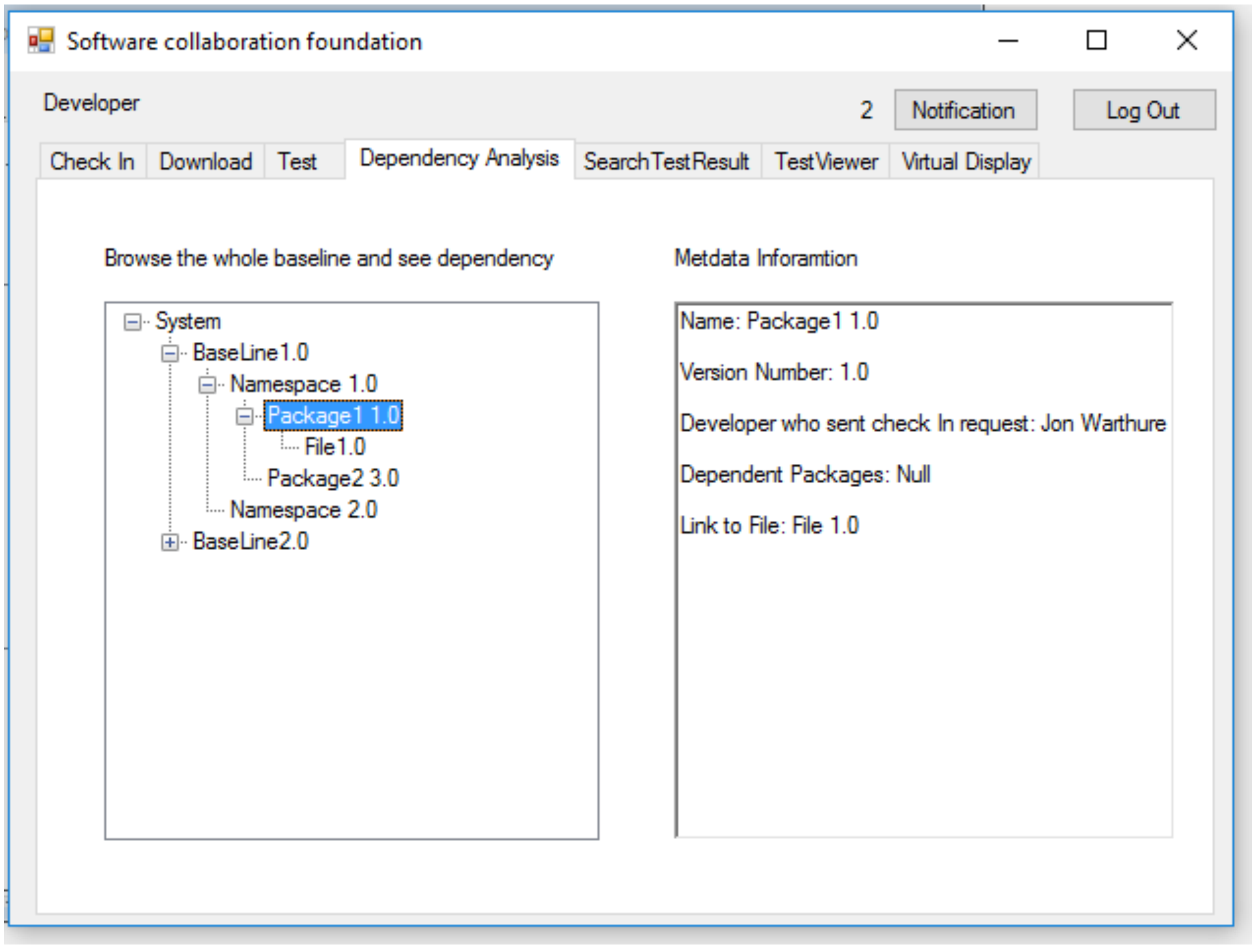
TestCodes:

Package 1
Package 2
Package 3
Package 4
Package 5
Package 6
Package 7
Package 8
Package 9
Package 10
Package 11
Package 12
Package 13

Figure 9 Window for test execution

4. **Dependency Analysis**
   This window is specially allotted for dependency analysis. User can do dependency analysis of the baseline using this window.

It also renders information from that files metadata file while you click on package, namespace or file while browsing the tree view.



Name: Package1 1.0

Version Number: 1.0

Developer who sent check In request: Jon Warthure

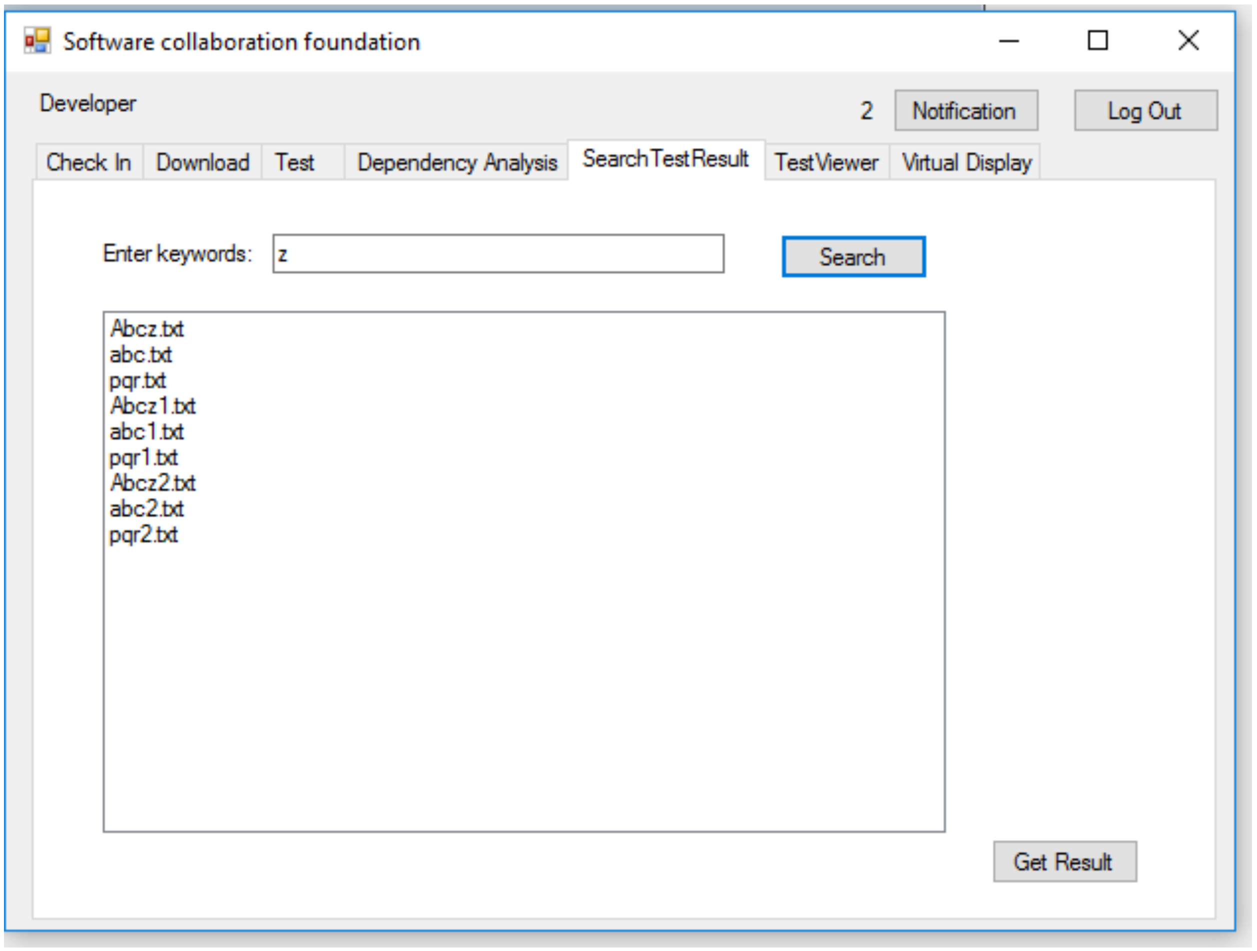
Dependent Packages: Null

Link to File: File 1.0

Figure 10 Window to analyze the dependency of the baseline objects

5. Search Test Result

This window is used to search the Test result. User just have to enter nay key words in the text box and press search. It will come up with list of test result file from Repository which has that keywords in its name or content.



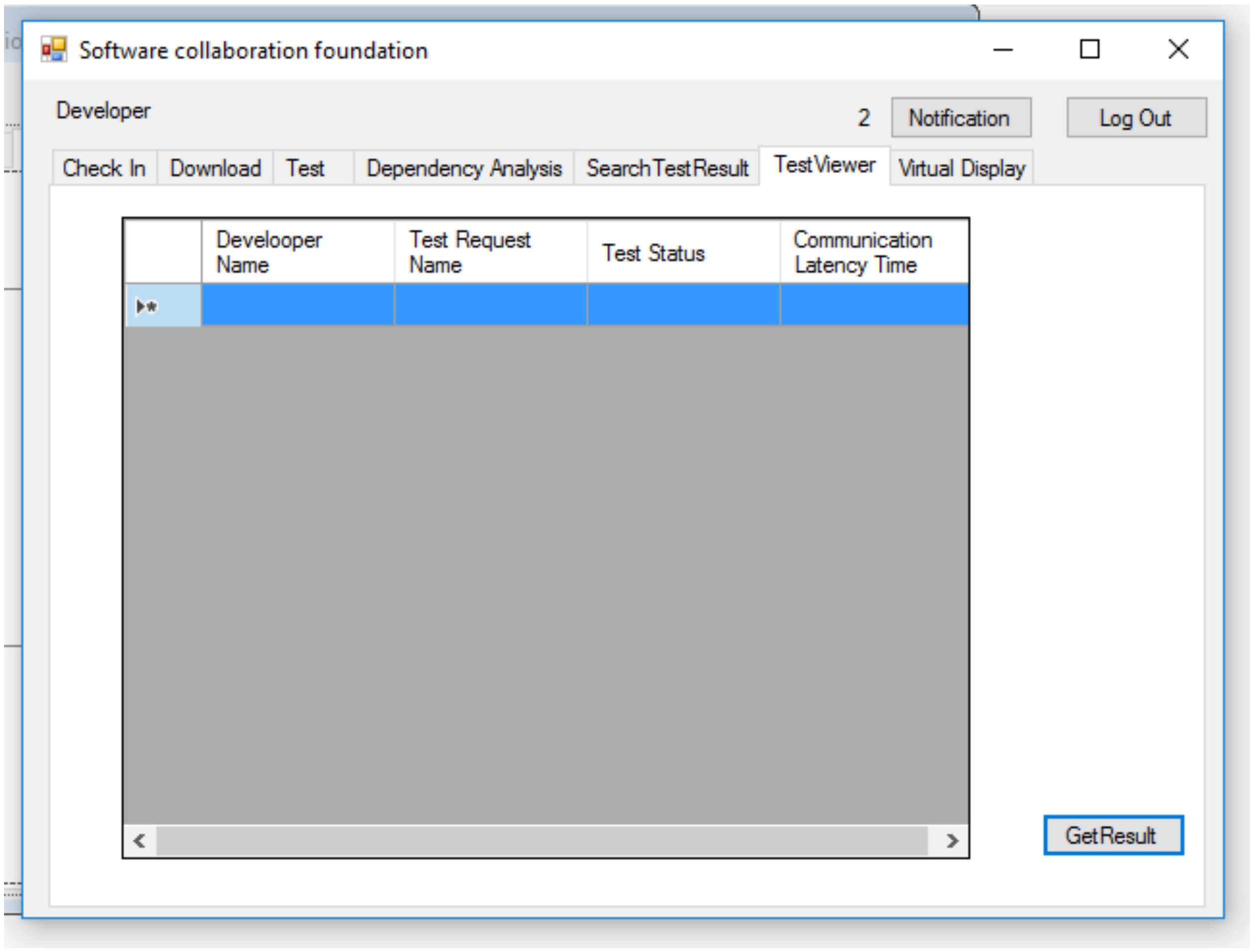
z

```
Abcz.txt
abc.txt
pqr.txt
Abcz1.txt
abc1.txt
pqr1.txt
Abcz2.txt
abc2.txt
pqr2.txt
```

Figure 11Window to search test result

6. Test Viewer

When user has just performed the test, then details of that running or completed test execution is on this tab. User can also get logs by selecting any of the test execution.



Develooper Name	Test Request Name	Test Status	Communication Latency Time

Figure 12 Window for Test Viewer

7. Virtual Display

Virtual display is collaboration tool which can be used for the functionality available in the GUI.
When user click on the Drawing mode enabled, use have to give ID of one of the user, then they both can only communicate using drawing tool since we have restricted number of users to only two when drawing tool is in use.

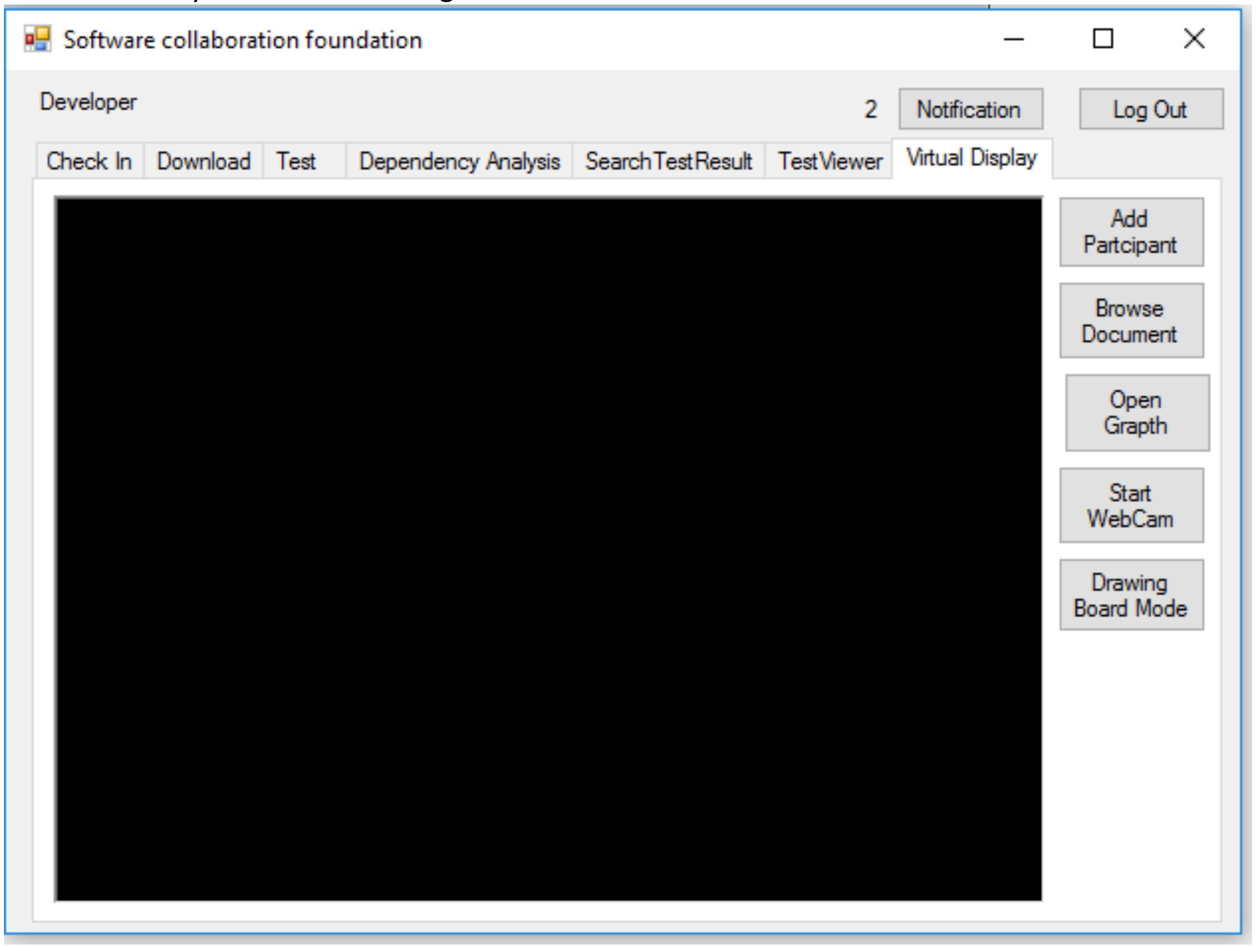


Figure 13 Window tab for Virtual Display

8. WorkPackageInfo

This GUI is used to modify, add work package information in collaboration server. User can also go to the related task by browsing the work package dependency tree view.
We have put the snapshot of this tab in Manager's section, but it will be same for all three user.

5.3.3. QA's GUI

QAs will have same tab and GUI as developer have. But, they have Test suit submit tab rather than send check in request tab

Submit test suit tab

QAs can submit the test suit(test driver) they have created using this tab.

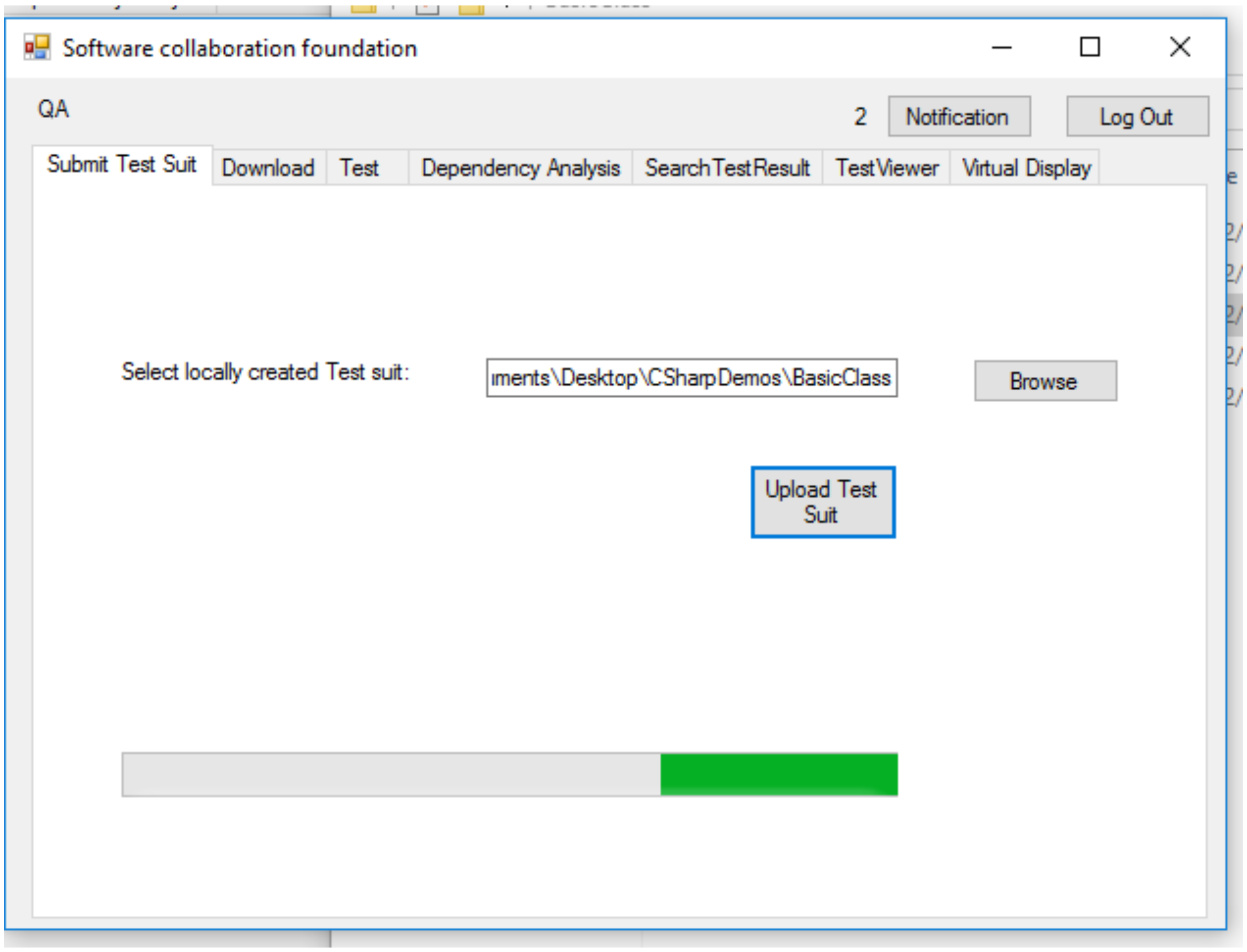
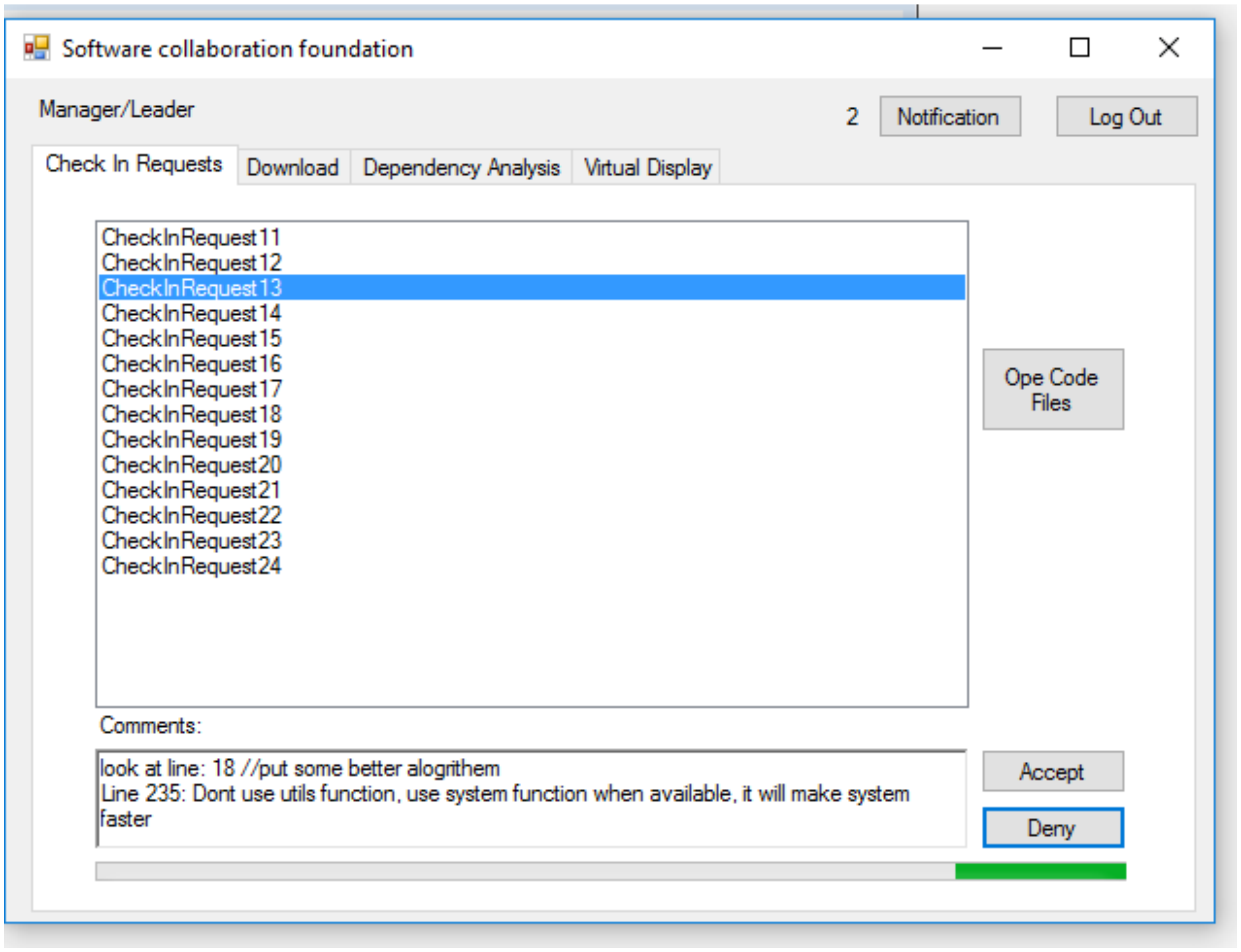


Figure 14 Window to submit Test Suit for QAs

5.3.4. Project Manager/leader's GUI

Project manager will have just the two GUI which is unique. Except that, it has other 2 same tab.

1. **To accept/deny check in request**
   Using this GUI, manager/leader can accept or deny the check in request with some comments. This comments will come as a notification to the developer.

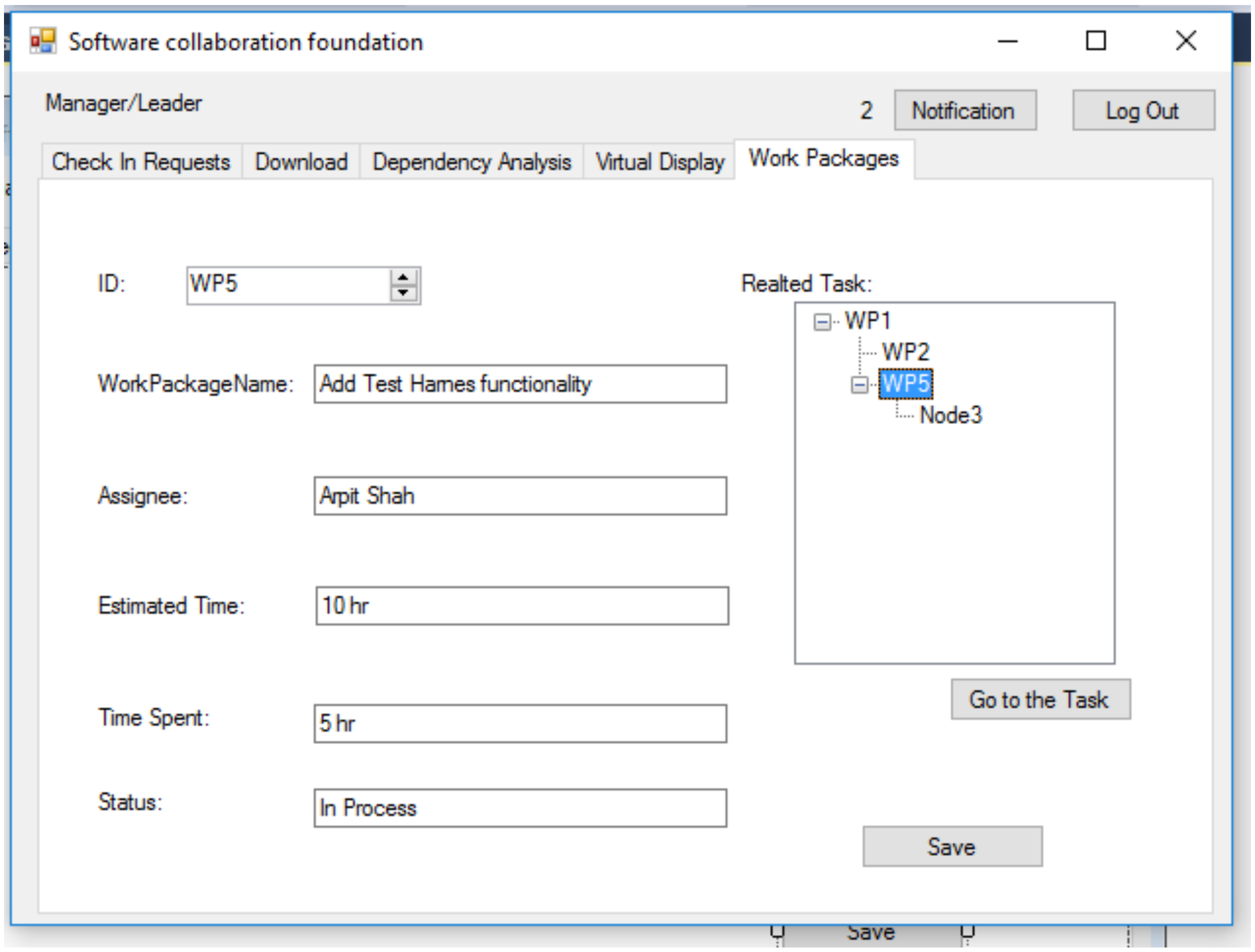


CheckInRequest11
CheckInRequest12
CheckInRequest13
CheckInRequest14
CheckInRequest15
CheckInRequest16
CheckInRequest17
CheckInRequest18
CheckInRequest19
CheckInRequest20
CheckInRequest21
CheckInRequest22
CheckInRequest23
CheckInRequest24

look at line: 18 //put some better alogrithem
Line 235: Dont use utils function, use system function when available, it will make system faster

Figure 15 Window to accept/deny check in request

2. **WorkPackge Info**
   This GUI is used to modify, add work package information in collaboration server. User can also go to the related task by browsing the work package dependency tree view.
   To edit the existing work package, user have to select work package Id from the Id tab, and all the field automatically rendered. Then, after changes user should click on save to store that information on collaboration server.
   This, GUI will be common for Developers and QAs.


Software collaboration foundation

Manager/Leader

2 Notification Log Out

Check In Requests | Download | Dependency Analysis | Virtual Display | Work Packages

ID: WP5

WorkPackageName: Add Test Hames functionality

Assignee: Arpit Shah

Estimated Time: 10 hr

Time Spent: 5 hr

Status: In Process

Realted Task:

- WP1
  - WP2
  - WP5
    - Node3

Go to the Task

Save

Figure 16 Window to add/update work package

5.4. Activities

Below is the activity diagram for client server. I have described all the client activities below in this diagram to give proper view about Client activities:

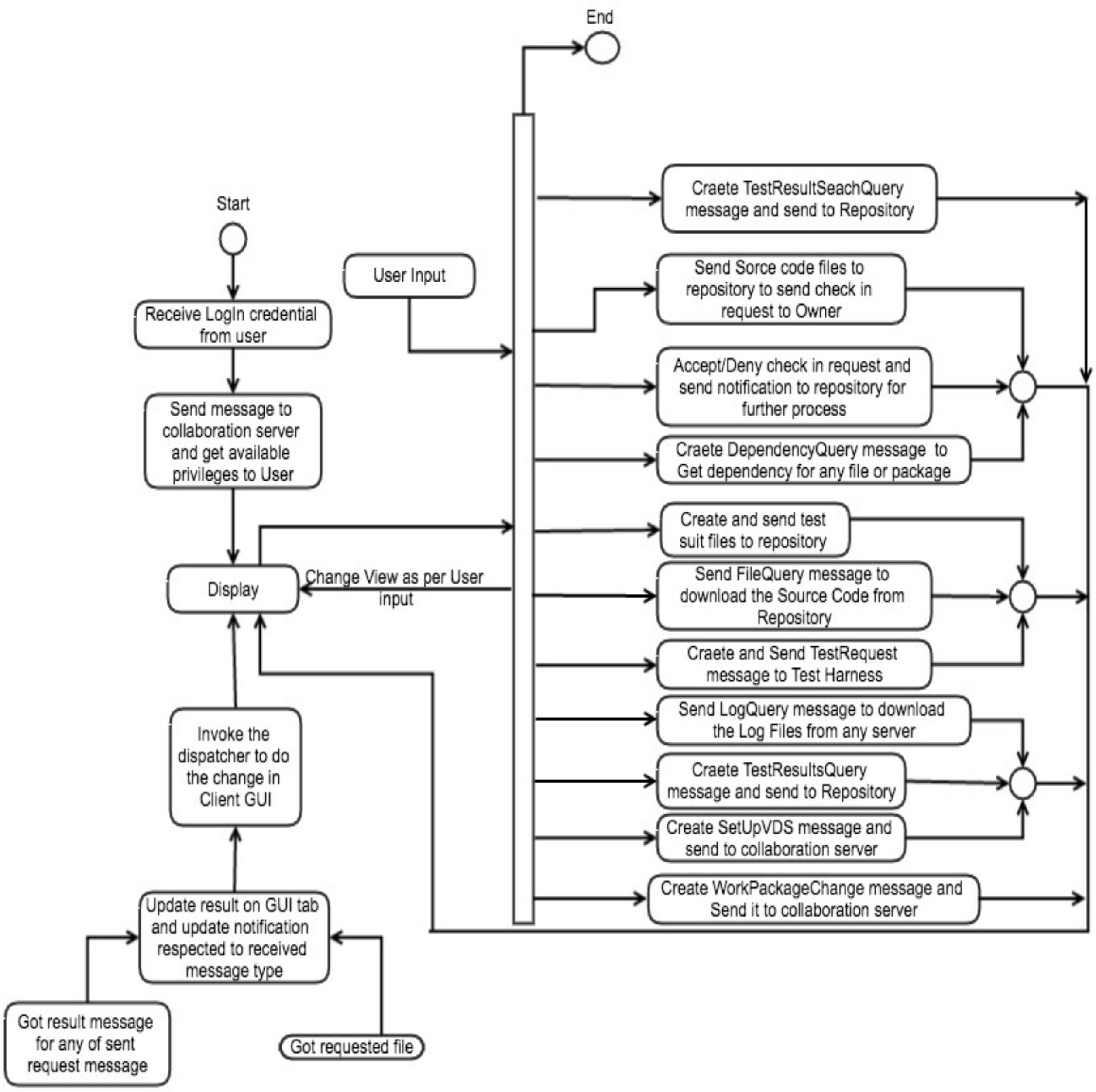

End
Start
User Input
Craete TestResultSeachQuery message and send to Repository
Receive LogIn credential from user
Send Sorce code files to repository to send check in request to Owner
Accept/Deny check in request and send notification to repository for further process
Send message to collaboration server and get available privileges to User
Craete DependencyQuery message to Get dependency for any file or package
Create and send test suit files to repository
Display
Change View as per User input
Send FileQuery message to download the Source Code from Repository
Craete and Send TestRequest message to Test Harness
Send LogQuery message to download the Log Files from any server
Invoke the dispatcher to do the change in Client GUI
Craete TestResultsQuery message and send to Repository
Create SetUpVDS message and send to collaboration server
Create WorkPackageChange message and Send it to collaboration server
Update result on GUI tab and update notification respected to received message type
Got result message for any of sent request message
Got requested file

Figure 17 Actvity Diagram for Client

I have added information about all the type of messages client server can send or receive ;and activity that happened because of that process.

Client will change its display according to the user's input for selection of tab. Different tab will provide different functionality to the user which I have described below:

1. **Send Source code files to repository to send check in request to Owner**

   User wants to send updated file or new file to the repository. The basic operation should be file uploading to the repository, but because of single user policy, user cannot do direct check in to the code files.

   Check in files will be added to repository in Temp Folder instead of adding into main base line as per Single Owner policy. Check in request files will have one or multiple test drivers attached with that check in request. As soon as user sent request for check in to the repository, client automatically sent **CheckInRequest** message to Repository and then Repository send the notification to manger about this CheckInRequest with information about attached work package, code files, test driver and documentation.

2. **Accept/Deny check in request and send notification to repository for further process**

   When manager will get notification on his/her UI about team member's check in request, it totally depends on Manager's decision that, whether he/she want to include this file in current base line or the files need more change. If Manager/leader found the any problem or correction with changes, they will deny it with some correction comments.

   If manager accepts then, repository will decide the version number of that file and checked in the code.

   For both of the cases, client server will create CheckInDecision Message and sent it to repository using SMS.

3. **Create DependencyQuery message to Get dependency for any file or package**

   User can select any namespace, package or files and select to get dependency by requesting repository. Whenever any object is selected to get dependency, client server will create **DependencyQuery** message and send it to the repository.

   As a reply, it gets Result **DependencyQuery** message, which has the list of objects which is dependent on the selected object and also some extra information about the selected object. User can use this function to check dependency of any files from the baseline. Example like, if user is working on any files, then it is necessary to check dependency of other files on the file which user is working on. If user won't tackle this thing with care, then uploaded file or checked in file may create some issues because of dependency.

   So user should check dependency information about the files while working on it, to get better idea.

4. **Create and send test suit files to repository**

QAs are most common user of this functionality. When developer checked in any file, he/she will attach test driver to test that source code. But, Quality Assurers are the user who creates custom test suit (test driver) to run different type of test execution on baseline. QAs creates the test suit on local client machine and upload that test suits to repository by browsing the client.

5. **Send FileQuery to download the Source Code from Repository**

User can download the source code and get local copy to work on that. So, User will have button to download latest copy of whole baseline. It will send the root o
User can also download the particular version of the file, by browsing repository remotely.
In both cases, client will send FileQuery with single filename or multiple filename.

6. **Create and Send TestRequest to Test Harness**

If user wants to test any package/module on primary level, user can send test request to the test harness just by selecting a package/module by browsing repository files from Client GUI. User don't have to select test driver for primary test because the information of that file/packages primary test driver is already in the metadata file of that file.
If user (particularly QA) wants to execute any custom or advanced test, then user have to select test suits he/she wants to execute.
In both cases, client server request Load Balancing server to allot Test harness server to send test request. For that it will send GetTH message to Load Balancing Server. After it gets test harness address with least load, client server will establish the connection with that Test harness server, and embed all the information about files from GUI in TestRequest message. It sends this Test request to allotted test harness for execution using SMS communication module.

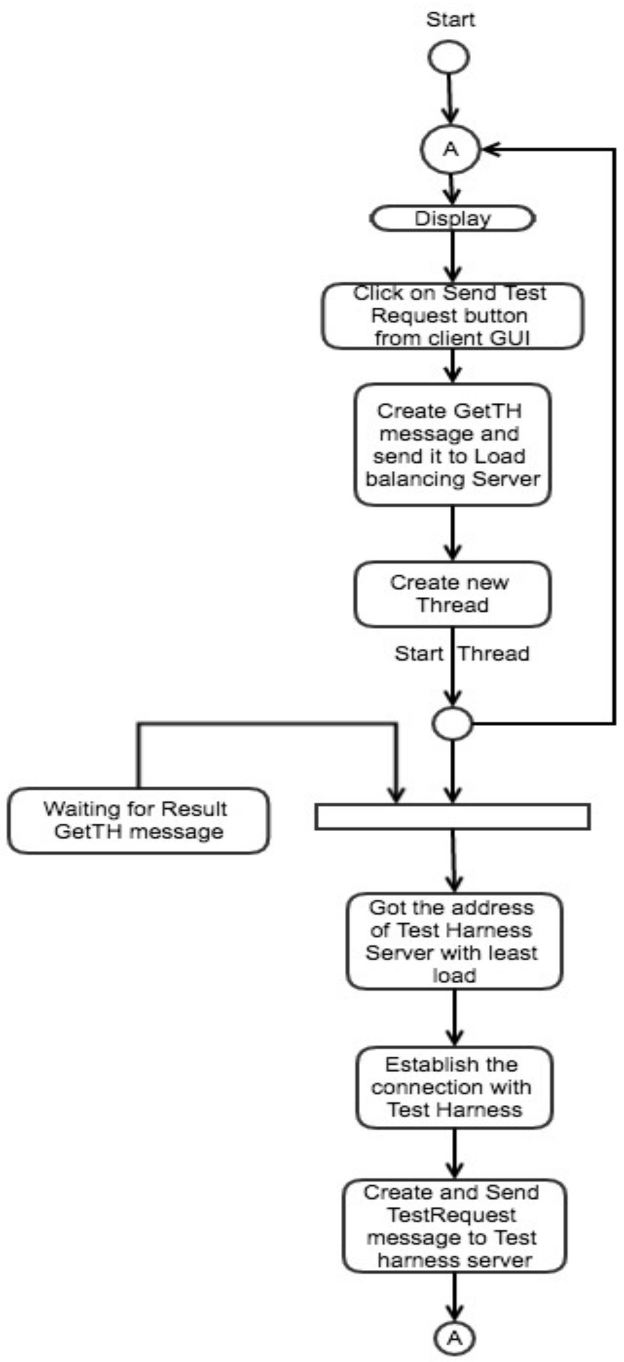

Start
A
Display
Click on Send Test Request button from client GUI
Create GetTH message and send it to Load balancing Server
Create new Thread
Start Thread
Waiting for Result GetTH message
Got the address of Test Harness Server with least load
Establish the connection with Test Harness
Create and Send TestRequest message to Test harness server
A

Figure 18 Activity Diagram about GetTH message and test harness server connection

7. Send LogQuery message to get the Log Files from any server

We have logger on each server which logs every activity of that server. If user want to analyze or debug the server by looking at its activity logs, then user have to select server from available list of servers. Client server will create LogQuery message using SMS message template and will send it to the selected server.

As a result, file will get updated to the requester client.

8. Create TestResultsQuery message and send to Repository

When test execution is completed, user will get notification about it. User just have to select the test request which he has sent previously. It will create **TestResultsQuery** message with name of Result log file and send it to Repository requesting to upload that file to client.

As a result, the requested file gets uploaded.

If user want to search log files for any past test request, then user can search it by using keywords. Client server will create **TestResultsSearchQuery** with that keywords, and repository will send list of files having that word. Then, user have to select any of the files, and he/she can have that logs by sending **TestResultsQuery** message to repository.

9. Create SetUpVDS message and send to collaboration server

User can share codes, change the position of the different window, schedule meetings, open documents, live chats using VDS. It has a lots of functionalities, so whenever user try to tinker with VDS, Client server create and send **SetUpVDS** message to collaboration server by noting every change in predefined XML message template.

10. Create WorkPackageChange message and Send it to collaboration server

When user will select the tab name Work Package, it will have functionality to browse all the work package from collaboration server. Any user will able to change detail in work package. User will select the Id of the work package from the list box and click Get Info, which will send WorkPackageInfo message to collaboration server using SMS.

We have stored all the information in collaboration server using SMS. So, collaboration server will get the message and find the data in NoSqlData using Work Package Id. This will send the data back to Client. This message populate each field from this message, which shows its current status. In that you can change the data you want, and store it using Message **WorkPackageChange.**

User can also create new work package and send it by creating **WorkPackageChange** message. Since in this new work package, it don't have work package Id. So, the message with same type, will be treated as different.

11. Receive SetUpVDS message

Whenever client receives **SetUpVDS,** it will fetch all the data and set up its Virtual Display(VD) accordingly.

5.5. Structure

Here, I have demonstrated the structure for client server with possible partitions. Client server consists client GUI, Logger, Timer and SMS.
SMS consist the structure we already have discussed to create and send message to the destination. It also helps client to process received message.

Below is the structure diagram for client server:

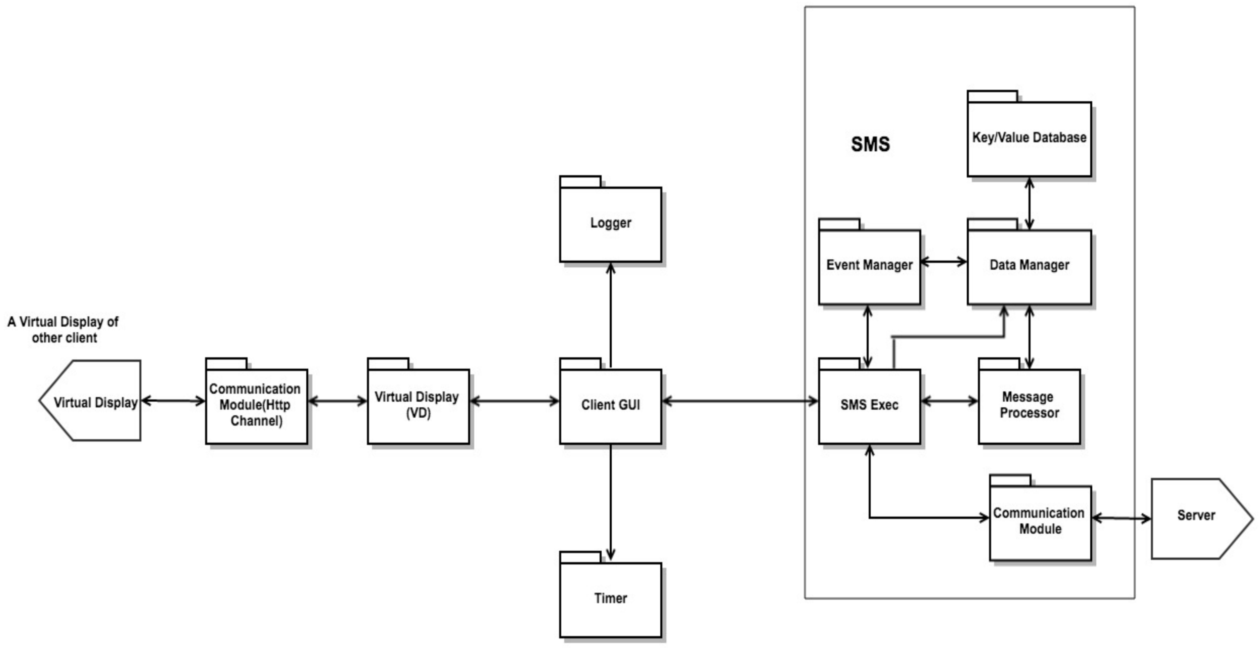


Figure 19 Client Structure Diagram

5.5.1. Client GUI Package

The Client GUI package comprises classes, which defines the User interface of this application. In this application, we are using Windows Presentation Foundation (WPF), which is a graphical subsystem for rendering user interfaces in Windows-based applications. This works as a core in the client side and let user to get view of different screens according to requirement.

Responsibilities:
The main responsibility of this package is to render all the view properly. In simple words, it is basically a communication medium which users are using to talk with application. Therefore, it reads the data from client side, sends data to SMS to loads XML files and utilize message processor to create a message for it. It contains main driver class who run mostly all the activities.

Interaction with other packages:
The client GUI package interacts with timer to measure communication latency time. It also communicates with Logger package to log every activity occurred on client side.

5.5.2. Timer

Timer package is mainly used to track the processing time of any actions. It will be used to measure total execution time or communication latency of activities. Whenever any process or communication starts, we start the timer. Whenever it ends, we again stop the timer and that is how we measure the time for any activity.

Responsibilities:
It is used to measure the communication latency time.

Interaction with other packages:
It interacts only with Client GUI, which notifies Timer when starting or ending of any activities at client side.

5.5.3. Logger

Logger is activity tracker of Client. It keeps logs of each activity occurred in client. It also stamps time and user details along with the logs to identify which user had invoked this activity.

Responsibilities:
Its responsibility is to store each logs on log file with activity information with timestamp and user details.

Interaction with other packages:
It interacts with Client GUI packages to support the functionalities.

5.5.4. Virtual Display(VD)

Virtual Display is used by clients to interact. They can share documents, code files, diagrams, webcam views using VD. It is very helpful for group communication between big teams.
VD has drawing tool which can used to share real time board drawing.

Responsibilities:

If clients are communicating using virtual display, then for each change in VD, VD will send SetUpVDS message to collaboration server using SMS. Change in VD can be changing layout, opening code files, documents, diagrams, starting webcam view.
For using drawing tool, only two clients can communicate on virtual Display. The reason is sharing real time drawing need very high speed communication. So, it should be done using Http messages between minimum number of user which we have restricted to two.

Interaction with other packages:
It interacts with Client GUI module. It also interacts with the other communication module which use Http messages, if drawing tool is enabled.

5.5.5. SMS

We have already described SMS on high level. In this section I am going to describe the specific service I have used provided by SMS in this server.
For this partition, Client server will send data by reading client GUI, and send it to SMS Exec who create appropriate message using pre-defined stored message template and message processor; and send it to the destination server using WCF module.
It also receives the message, processes it using Message processor and message template. It sends that data to GUI, and GUI will render the data on respective interface.
We already described how it creates message, how it parse the message using message processor in previous section.

Responsibilities:
Its responsibility is give message processing and communication service to client server.

Interaction with other packages:
It interacts directly with Client GUI and get the data or instruction to create message to send. It also receives the message and send the data to GUI by parsing it.

5.6. Critical Issues

5.6.1. Different features for different users

All users don't deserve same privileges. The developers shouldn't be given equal privilege as the software architect. Otherwise, it can be harmful for the system.

Solution:

According to user privileges, the client window is generated after log-in and verification of user privileges.

5.6.2. Caching

If user is requesting for some information during the same session or any near preceding sessions, then he will have to wait for the same time as he waited for the first time for the answer. This is an aspect which could be and should be improved upon.

Solution:

By using a proper caching structure at the client side this issue could be resolved. Caching means creating temporary copies of files which were recently used and checking them before fetching new files in the client side. The logic could for this could be thought as, whenever a request for some data is sent first the cache is searched for that data and if the data is found then its version number is matched with at the server side it is decided whether the cache has the latest copy or not. And if the cache is hit with latest copy then the client won't have to wait until the whole file is fetched and download it, it will just use the copy from the cache and latency in the communication would be reduced significantly.

6. Load Balancing Server

We already have brief introduction of Load Balancing Server while we discussed about the architecture of SCF. In this section, we will talk about detail activity of each process and how it happens. Load balancing server helps client to find best test harness server which executes client's submitted test request in least time of waiting.

6.1. Concepts

6.1.1. Efficient use of Test Harness Server

For big federation, we need to add multiple test harness server to increase the work efficiency of the team. Load balancing server is to manage the load distribution between Test Harness server. We shouldn't fill one Test harness server with 500 test request while other one have just 10 request in queue. Therefore, Load balancing server will help to get better performance by managing the available resources.

6.2. User and Uses

Client Server and Test Harness server will use this Load Balancing server for uses we have mentioned below.

- **Allotment of Test Harness Server to Client**

Client send request to Load balancing server to get Test harness server with least load. Load balancing server will search and allot Test Harness server with least load.

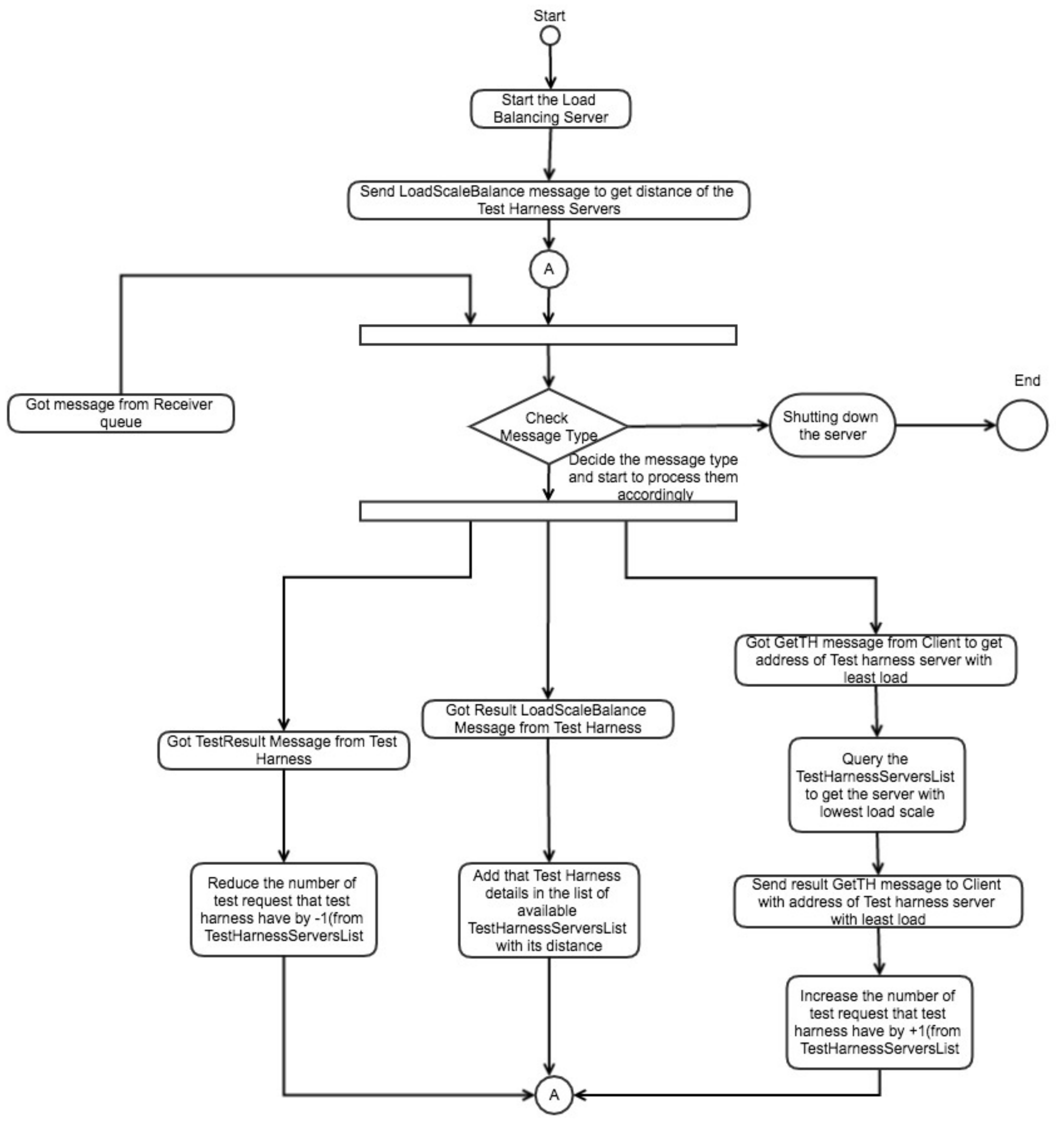
- **Count Load Scale of each Test harness Servers**

Load Balancing Server will have real time load scale count for each Test harness Server. Using that count, it allots test harness server address which has least load scale to requested client.

6.3. Activities

Load balancing server's main functionalities to keep eye on balanced distribution of load on Test Harness Server to get better performance with optimum resource utilization.

Below, I have demonstrated server activities diagram with description of each activity:



Start
Start the Load Balancing Server
Send LoadScaleBalance message to get distance of the Test Harness Servers
A
Got message from Receiver queue
Check Message Type
Decide the message type and start to process them accordingly
Shutting down the server
End
Got TestResult Message from Test Harness
Got Result LoadScaleBalance Message from Test Harness
Got GetTH message from Client to get address of Test harness server with least load
Query the TestHarnessServersList to get the server with lowest load scale
Reduce the number of test request that test harness have by -1(from TestHarnessServersList
Add that Test Harness details in the list of available TestHarnessServersList with its distance
Send result GetTH message to Client with address of Test harness server with least load
Increase the number of test request that test harness have by +1(from TestHarnessServersList
A

Figure 20 Activity diagram for load balancing server

Here, I am going to demonstrate each activities of load balancing server.

- **Send LoadScaleBalance message to get distance of the Test Harness Servers**
  After all the server starts, Load balancing server will send LoadScaleBalance message to each Test Harness server with it, and wait for the response so that it can add distance for each of them in the TestHarnessServersList.

- **Got Result LoadScaleBalance Message from Test Harness**
  After sending LoadScaleBalance message to each Test Harness server, it will get result LoadScaleBalance message from all of them with distance. So for each result LoadScaleBalance message, Load balancing server will enter all the details of Test Harness Server in one List called TestHarnessServersList.

When client want to send test request to Test harness server for test execution, Load balancing server help client to decide the server to send the request from multiple Test Harness server.

- **Got GetTH message from Client to get address of Test harness server with least load**
  It will get GetTH message from client which is requesting an address of Test harness server with least load.
- **Send result GetTH message to Client with address of Test harness server with least load**
  After getting **GetTH** message from client, Load balancing server go through TestHarnessServersList and find the Test harness Server with least Load scale.
  It will get the address of that test harness server and server **and result GetTH** message back to requester client.

- **Increase the number of test request that test harness have by 1(in TestHarnessServersList)**
  After it sends message to the client with the Test harness address, it will increase the number of test request that test harness have by 1 in TestHarnessServersList**.** By doing that, we are maintaining the real time Load scale count for each Test Harness Server on Load Balancing server, therefore we do not need to query test harness server to measure the Load Scale Count.

After, the test execution completed on Test harness Server, it notifies the Load Balancing server by test result message.

- **Got TestResult Message from Test Harness:**
  - Test Harness server will send empty message of type TestResult to Load Balancing Server when test execution get completed.
  - After Load Balancing Server get empty message of type TestResult from any Test Harness, it will search that Test harness by the address of the sender into TestHarnessServersList, and reduce the number of test request that test harness have by 1.

6.4. Structure

Here we have designed Load Balancing Server(LBS) with a motto of optimum resource utilization, so we have taken care while designing that adding LBS won't add any extra cumbersome process in SCF process which decrease performance instead of increasing it.

Load Balancing Server consists LBS Executive, Logger, Communication Module and TestHarnessServerList Manger, Message Processor.

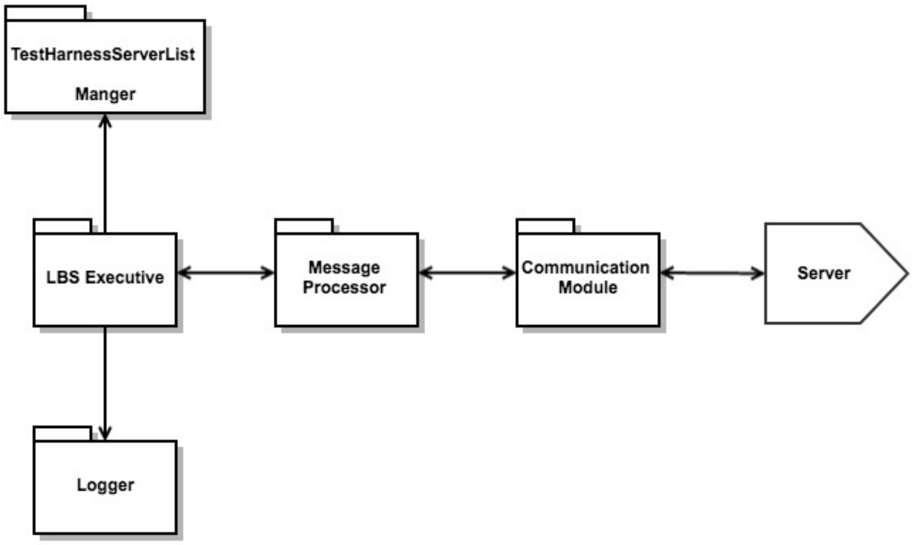


Figure 21 Structure Diagram for LBS

6.4.1. TestHarnessServerList Manger

It contains the actual data of all the available Test Harness Server connected to it and other information related to Server like, distance, Load Scale Count, Number of Test Request waiting in Blocking queue of that that server. This all data are real time data.

Below is the data which is stored in TestHarnessServerList:

```
class TestHarnessServerList
{
string testHarnessAddress;
int distance; // distance of test harness server
int Nreq;     // number of test request in blocking queue of test harness server
int LSC;      //Load Scale Count
}
```

If any of the test request get dequeued and test request execution got completed, that means load on that test harness server is decreased. So, we decrease the number of test request in the TestHarnessServerList for that server.

To get this type of real time data, we can do two things:

1. Querying each test harness server to get the number of request in its blocking queue
2. Test harness server itself give some light weight notification about its status

While thinking about both approach, first approach seems expensive since we have to query each test harness continuously to get the result we want.

Second approach seems less expensive than first. It is understandable, it will impose some load on Test harness server since server has to send some notification. But, we are counting test request on each test harness server by using this TestHarnessServerList Manger. When test harness got allotted to any client, we increase the count of Nreq. When test execution got completed, test harness sends first empty light weight TestResult message as notification to LBS. So, we decrease the count for Nreq sender Test harness Server.

Interaction with other package:
It interacts with LBS executive. LBS executive modify the data in the list, add new data in the list, fetch the current data from this TestHarnessServerList using function defined in the TestHarnessServerList Manger package.

6.4.2. LBS Executive

LBS Executive is main executive package which is working as distribution actor. It get the processed message from Message Processor and keep updating TestHarnessServerList.

Responsibilty:
Its main responsibility to invoke some process by using TestHarnessServerList Manager when it receives any related message .It gets three type of message. I have listed message and its reaction for that message:

1. **TestResult message**- decrease the Nreq by 1 for the sender test harness data
2. **LoadScaleBalance message** -add data of test harness in the list
3. **GetTH messag**e - send the address of test harness server with least load.

Interaction with other package:
It interacts with TestHarnessServerList Manger to modify data, to add new data or to fetch current data from the list. It also interacts with Logger to log each activity occurred in the server.

6.4.3. Message Processor

Message processor is working as a wrapper, which will be used to wrap message into a pre-defined format and will queue it into sender queue.
It also works on either way. If there is any message in receiver queue, then it will first come to Message processor. It will then process it, that's means parse it and send that content for further process.

Responsibility:
It gets message from Communication module, parse it and handed all the data to LBS executive. It also gets data from LBS Executive, create message and send it through communication Module.

Interaction with other package:
It communicates with Communication module and LBS Executive.

6.4.4. Communication Module

It communicates with client and repository server. It will have interface for communication, which holds data contracts.

Responsibilities:
It will get message from Message processor, it will mainly hold two blocking queue. One of them is sender and other is receiver. One separate thread is working for this sender and receiver queue to keep monitoring it continuously.

Interaction with other packages:
It queues up the messages, which it got from Message processor in Sender queue. It also receives and pass dequeued message to Message processor.

6.4.5. Logger

Logger is activity tracker of each server. It keeps logs of each activity occurred on the server. It also stamps time and user details along with the logs to identify which user invoke this activity.

Responsibilities:
Its responsibility is to log each and every activity of the system and keeps a track of all the

important events in the system. Logs should be very useful for developer while debugging and analyzing the system as he /she can run multiple tests on the system and a well-designed logger can record all the warnings and errors during the execution.

Interaction with other packages:
It interacts with LBS executive package to log each event.

6.5. Critical Issues

6.5.1. How to count Load Scale Count(LSC)?

Load Balancing Server counts the real time Load Scale Count of each test harness server. We have done it to get performance on test request execution by sending it to the test harness server with least load. There is no specific rule to measure the Load Scale Count.
Solution:
After thinking about the all approaches, as of now, we can just depend on communication time and number of test request in blocking queue on that server. If we have approximated execution time for each request waiting in the blocking queue, then it would be very helpful. But, then it would be too complex to get that time and it may impose a high load on the system which opposing our motto to insert load balancing server in SCF.
So, after lots of trial and error, we reached on conclusion that we should depends 80% for the number of test request in blocking queue and 20% on the communication time since Nreq(number of test request in blocking queue) is most volatile value, while change in communication time can be negligible at higher level.
So,
LSC= 0.8*Nreq + 0.2*(s/d)
Where
Nreq = the number of test request in blocking queue;
s = speed of the signal
d = distance between between Test harness server and LBS
We are not sure about this will be perfect for each scenario or not, but it will definitely help us to get better performance for most scenario.

6.5.2. Why didn’t we use SMS in this server?

Use of Storage Management Subsystem(SMS) is always be an good alternative to store data and for communication. But, here we have data of some limited servers with 3-4 column, so it would be very expensive to setup SMS for this small amount of data.

Solution:
We used simple class model to store the data of Test harness server since this data is not that much massive, implementation of SMS would be very expensive with negligible performance improvement.

‘

7. Repository

The repository server is the main source of memory storage for this system. We already have brief idea about repository from architecture structure description.

Generally, repository is used by big scale project where lots of people are participated as a team. So, repository is used to organize and manage the code with its each version of files. Repository is memory management server which is hosted so that it can be accessible remotely. Team which is working on that project can access the code from repository.

Developers can check in the code in the repository with help of project manager or product owner. Generally, developers can also check in the code, but it creates problematic chaos in source code when multiple developers started editing the same code file by downloading it on their local machine and then try to check in final changes at same time. It may create a conflict in base line and after that they have to pass through a cumbersome process of rollback.

So, we have developed a repository with SINGLE owner policy where is single person privileges to check in the source code. There will be some high position like Project manager, Project leader or product owner only have privileges to check in the source code. Other user can modify other code available in repository which is not affecting the current baseline, example like, QAs can check in their test suit (test drivers which are designed to perform custom test) without anyone's help since they are not tinkering with Source code.

Repository also handles versioning of baseline without creating any chaos because of it good design. We have created three directories in the repository to manage the baseline
(1) Temp – It contains a packages whose check in request is already sent
(2) Open Check in – Packages who are in open check in state
(3) Main- it will contain whole working base line with every version

Now, we are going to see detailed design of repository and its each part.

7.1. Concepts

- Repository is built to store baseline on center, so that every user can have access of same baseline. They can access it, modify it, correct it and update it by just creating different version without deleting the old files and the good thing is all this work is handled by repository. Repository is designed to manage package incoming and outgoing dependencies. It supports controlling the current baseline as well as supports the browsing the whole baseline we have on repository with all versions.
- It gives full disclosure of the baseline since the feature of browsing. User can have all the information about each file from its metadata file. Like, file test status (passed or failed), check in status (Open or close) etc.

- We have implemented SMS services in Repository to save decencies information from metadata manifest file for each namespace, package, file, and their path. This will help to route and get all the information about dependencies of the files.

7.2. Users and Uses

7.2.1. Developers

Developers will use this server to check in code files, to get the result files, to get the dependency for one file.

7.2.2. QAs

QA will use this server to store test suit files. Test suit files are the test driver file can be used to run the test execution and get the result.

7.2.3. Managers/Project leader

Manager will use this server to have a look at code status since we are storing test status in the metadata of each code file.
Manager/project leader will use this system for code review. When they gets the check in request from any developer, they will review the code and if there is any mistake, they will deny the check in request with some comments. They can include the correction in the comments.

7.2.4. Uses

Repository can be used in lots of way, but here I have explained some important uses:

7.2.4.1. Manage Package Dependencies

It helps to manage dependencies of the packages on each other. Each package has some dependencies which user have to care about while modifying or testing that file. Even while checking in the code, that file has some incoming and outgoing dependencies on other packages from repository. Incoming dependencies are helpful for testing and outgoing dependencies are used to build that file. So, repository is designed to store files and also with concept to tackle dependencies of each package by creating its metadata file by itself after check in using SMS.

7.2.4.2. Controlling current baseline

The other main use is to control current baseline. User can change the version of selected current baseline. If user wants to update to next available version, he/she can update to the latest version he/she wants. If user wants to downgrade the version, then that is also possible using Repository.

7.2.4.3. Explore the baseline and its state

Implementing Repository will help user to explore the baseline. It can browse the full baseline of source code with each version of the file. It also has access to know test status of the source code that whether it pass the test or failed.

7.2.4.4. Open Check In

Repository is giving option to do open check in. For this, user don't have to send any check in request to the manager or higher authority. Repository will have other folder for Open Check in files, which is used to store all the open check in files. Repository will not do any automatic dependency analyzing or version changes for this files.

7.3. Policies

7.3.1. Single Owner Policy

- **Why we need it?**

When there is big federation and big numbers of developers are working on the same source code, the probability of creation of chaotic situation is high.

The reasons are:

1. If multiple developers are working on the same code, and they checked in at the same time, then it will create different version for all of them. They have to merge all the code, since none of them was working on the updated copy. Merging proses is very hectic, and we should try to avoid it as much as we can.
2. Merging is still possible thing. Developers have to waste some time, but it can be done if the changes in the single code files are at different place. If they are at same place, then it is impossible to merge the code except you refactor all the changes.
3. If any developer check in some wrong code or bad implementation, which may stop the working system. It is possibility that the checked code may work, but bad implementation ruin the efficiency of the system.

This is the reason motivates designer to implement Single Owner Policy.

- **Policy Definition**

According to that policy, only single user can check in or change the source code. This will help to avoid mostly all conflicting situation. It also helps to reduce the merging scenario in real time software development.

Because of this, every checked in request is review by project leader/project manager who have very good technical skill. So, the possibility of getting error in system because of check in will be decreased.

7.3.2. Versioning Policy

- **Why we need it?**

Versioning policy is to reduce the version confliction of the different files. This occurs mostly after user check in the modified file. If user is giving version name, then there is a possibility of creating file with same version. This problem may create confliction while check in process, Otherwise, it may be completely overwritten, so we can't find old version.

- **Policy Definition**

We create metadata file for each package, each code files in which we describe the version number. If it is source code file, then we additionally adding version number as comment in header of the package while user check in. So, during checking user have to select the old file whose updated/modifies version is being checked in right now. By doing that, repository automatically decides the version number of that file using version manager. According to policy, we are creating major version each time user check in the code. Example like, 1.0,2.0,3.0 etc.

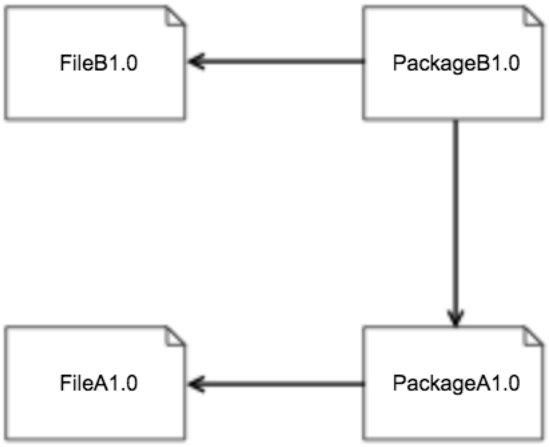
7.3.3. Updating Policy

- **Why we need it?**

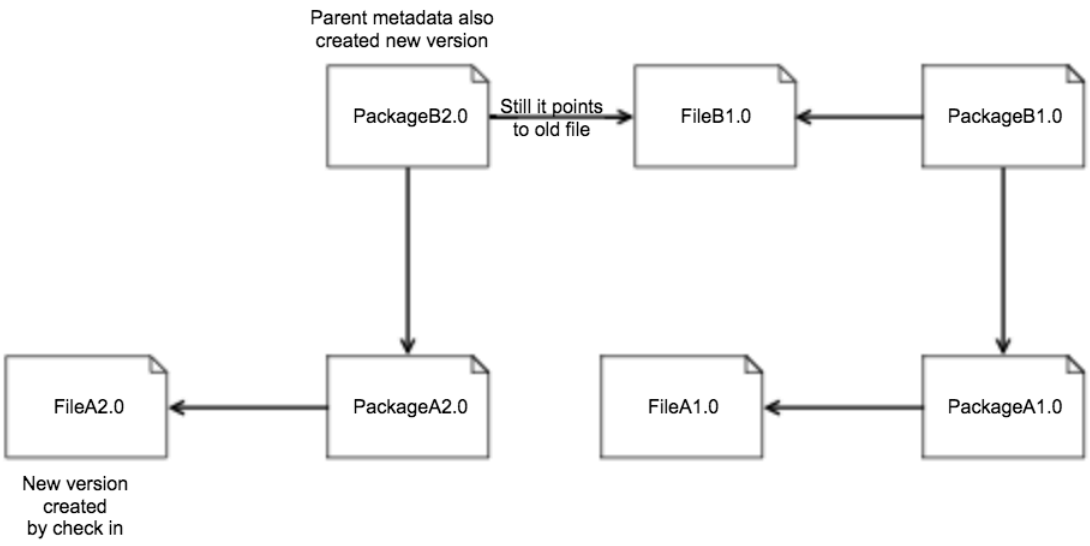
Updated version of part of file/package creates confusion that how we are going to tackle both version and their incoming and outgoing dependencies. We have to support the developers who are working on the old baseline as well as developers who wants work on new updated baseline. Therefore, we have to manage both files, and their dependencies.

- **Policy Definition**

After completing check in process, we are automatically creating new manifest metadata file for the parent file/package of the new checked in file. This new manifest packages will link to the old file, but point to the new checked in file. It will give an option for user to update its selected bassline if he/she wants. Otherwise, his/her root will stick to the old baseline. It will just work fine since we haven't changed any of the data of old baseline.
Example like,
This is part of my old baseline:



I am checking in the file FileA1.0 which creates its new version FileA2.0.
So, after checking in my that part of the baseline will look like this:


Parent metadata also created new version
PackageB2.0
Still it points to old file
FileB1.0
PackageB1.0
FileA2.0
PackageA2.0
FileA1.0
PackageA1.0
New version created by check in

7.3.4. Check In Policy

- **Why we need it?**

Since we are following single user policy, every user wont able to check in the code. So, it need the clarification about how check in will work?

- **Policy**

  Because of the single owner policy, once developer finalize the changes, he/she have to send check in request to Manger which will be handled by Repository server. When user finalize to send the check in request, which comes with all the data file, which will be stored in Repository Temp folder and Repository send that check in request to Manager.
  When manager accepts it, it will move to the Main baseline.

7.4. Structure

Repository is structure who works as backbone of whole SCF by storing and managing the data. As I have discussed, it supports many other functionalities apart from just storing the data. Here I have demonstrated structure of the Repository with its detailed description of each server.

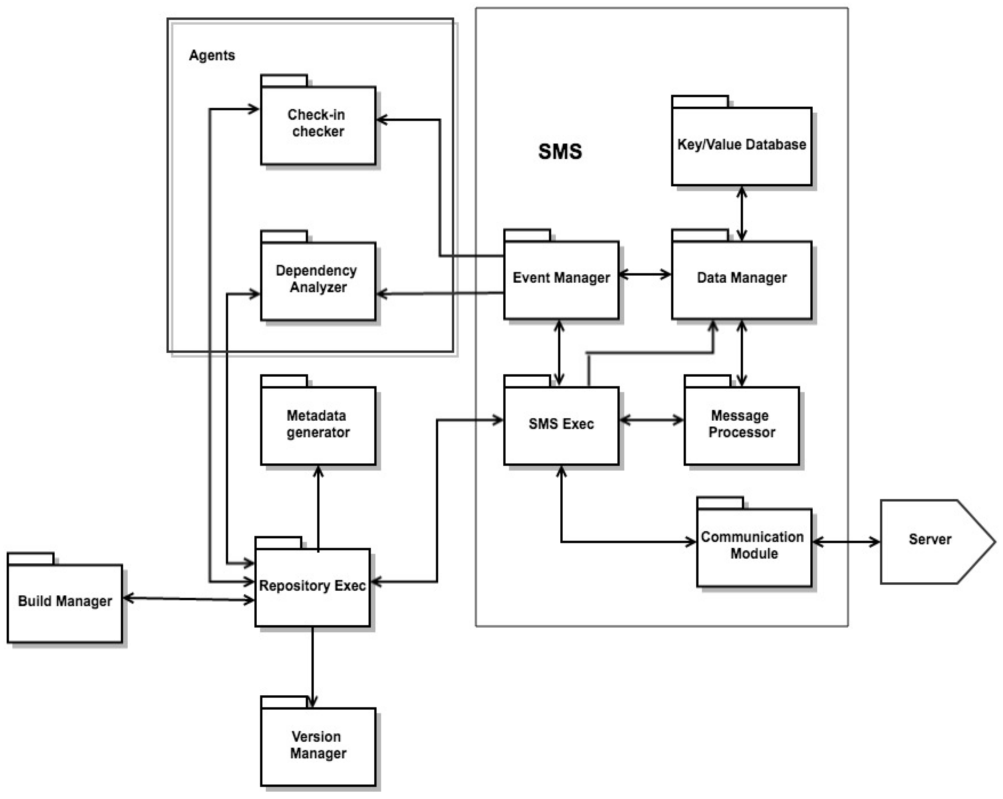


Figure 22 Structure Diagram of Repository

Repository consists Repository Exec, Agents, SMS, build manager. We have two Agents in Repository which will be used by SMS event manager to get data about repository.

1. Dependency Analyzer, which will analyze the dependency and create metadata file for each check in file and package.
2. Check in checker, which will go through each file and scan the status of the file. If it says, Open Check in, then it reports to the Manager.

Here I have described the Repository and its each package from its structure diagram.

7.4.1. Repository Exec

Repository Executive is the main server of the Repository server since it stores all the source code files, test result files, documents, manifests metadata file for each file, package, namespace. It will have three main folders to handle baseline files:

Responsibility:

- Repository Exec will work as storage which will store all the files. It still works as main executive which have to deal with all the incoming and outgoing message.
- If any check in accepted by any manager, then It get message from client through SMS about **CheckInDecision**. If it is accepted then, the Source code get moved to the main base line and check in notification generated and sent to the Developer using SMS. If it is denied, then just one notification generated about denial of the check in with Manager's comments.
- It helps both configured agents by providing the files they want from Repository storage.
- Whenever it gets dependency information for particular code file from dependency analyzer, it updates the Metadata file for that code file using metadata generator.
- Whenever actual check in happens after acceptance from manager, it will move the source code file with its other information from Temp to Main baseline. It stored that file by creating new version of that file. It decides the version with the help of version manager. It creates metadata file including all the information using metadata generator. At this stage, it won't have dependencies information, it will be provided by dependencies analyzer afterwards.
- When checked in file has dependencies information from Dependency analyzer, it will create New metadata for each Parent which has direct or indirect dependency on this checked in file and update Dependency table which is stored in SMS . Then, it will search through its inner dependencies from Dependency table, create test request for check in code files and other files which is dependent on that and send it to the Test harness for testing. When it will get test result, it will update it in metadata file of that code file.

  When it received message type **TestResult,** then it will fetch file stream from the message and store it in repository.
- It supports **FileQuery** Messages and send requested file to requester.
- It also supports **BuildQuery** Messages. It finds the build in the build server, if the build with same version is available then it sends that file, otherwise it will fetch file from main repository, build it and send it.

- Whenever it gets DependencyQuery Message, it will find that file from its name and its version number from Dependnecy List. Then, it will get path of the metadata file and send all all the information from the metadata file.

Interaction with other packages: It communicates with Metadata generator, version manager, Check in checker, SMS, Dependency Analyzer.

7.4.2. Version Manager

Version Manger is used to create and decide new version for new checked in code file. It helps a repository for version control which can be seen as key functionality which reduce the chances of chaos.

Responsibility: Version Manger will be responsible for deciding the version number for new checked in source code files. It will decide version number and insert it into source code files as comment in header.

Example like, It will add "\\ #" followed by version number, like if source code is of version 1.3 than it will add string "\\#1.3" on header of the file. So that, we can know about the version of the code file, even without using the metadata file of it.

Interact with other package: It communicate with Repository Exec to help it about versioning.

7.4.3. Metadata Generator

Metadata generator is the key part of the repository since the whole management of repository's different kind of files depends on their metadata file associated with it.

Responsibility: Whenever any new check in happens, the requirement of creating metadata arises. We have designed a repository in a way that it will generate XML metadata file for each new source code file, package, namespace.

Whenever any file get acceptance in CheckInDesicion message, the message will have structure like this,

```
<Message>
<Type>CheckInDecision</Type>
<Decision>True</Decision>
<CheckInObjectPath>(Path on old object on Repository)</ CheckInObjectPath>
<DevelopersInfo>(Developer Email)</ DevelopersInfo >
```

<Date>(Date Of upload)</Date>
<DocumentPath>(Path of the Document)</DocumentPath>
<RelatedWorkPackage>(Work Package Id)</ RelatedWorkPackage >
</Message>
It will get all the necessary information to fill the metadata template from this received message. It will get version number of the file, by using comment we created in header of the file by version manager. It wont have dependencies information, so it keeps that as a blank. It will get filled once Repository Exec get dependencies information from Dependencies Analyzer.
Created Metadata will look like this:
<MetaDatafile>
<Name>(Name of the file) </ Name >
<Path>(Path Of the file on repository)</ Path >
<Versionnumber>(version number)</Versionnumber>
<DevelopersInfo>(Developer email id)</DevelopersInfo>
<TestDriver>(Name of the Test Driver)<TestDriver>
<DateOfCreation>(Date of creation)</ DateOfCreation >
<TestStatus>(test is passed or failed)</TestStatus>
<RelatedWorkPackage>(Id of an work package for which this check-in is being done)</RelatedWorkPackage>
<Dependencies></ Dependencies >
</MetaDatafile>
It stores this type of metadata file with each file, package, namespace which depicts all the information about them.

Interaction with other package: It just communicate with Repository Exec and provide functionality to create metadata file when required.

7.4.4. Agents

For the sake of process simplicity, we have configured two agents who is invoked by Event Manager of the SMS. These agents are configured to do predefined work process on the basis of the given input. Here I am describing two agents which are working on event manager's input

7.4.4.1. Dependency Analyzer

Dependency analyzer is agent which is invoked by Event manager of SMS. This will get the name of the checked in content and start finding dependences for that.
It has two components to help him to analyze the dependencies:

1. Tokenizer List

It collects name of all the class, structs, interface, template defined in the file as a key. It also stores the name of the file with its namespace as value. It is basically a dictionary sort of object.

2. File Streamer

It helps to open and stream the file to search for the selected strings.

Process:

- When it gets the name of the new checked in file, it will use streamer to open and read the file. It stores the tokens in tokenizer list with its filename in which it defined.
- While reading the file, it also check whether this file have instantiated any class objects which is there in tokenizer list. If it finds any this type of implementation, then it will get the value from that dictionary which is the name of the file. It adds that name in dependencies list of the new checked in file.
- After completing the process, it hand over the dependency list to Repository Exec, to get it updated in the file metadata.

7.4.4.2. Check In checker

User can do Open Check in of the file if he/she is not sure about the check in. Open check in facility store your file in allocated folder in repository without disturbing the baseline.

Whenever the user decides to go for close check in, the file will get moved from OpenCheckIn to temp folder and check in request send to the Manager.

Whenever Manager wants to see the file which are in open check in, it tries to browse that folder. At that time, request is sent to Repository, SMS receives that request and start the Check In checker agent using its event manager. Agent will finds and returns all the file with open check in status.

7.4.5. Build Manager

Build Manager helps repository to create image file of source code. It manages the information about the image file it has in cache memory and version of that files.

Before building any files, it fetches version number from its file's header which we had added as a comment and try to find the build image file in cache. If it is there in cache, it won't get build again. If it is not there then, build manager will go to build it. If file is successfully built then, it will add the information in this table.

Table data will be like this:

File Name	Version Name
TestCode1.dll	2.0
TestCode2.dll	7.0

Responsibility: Whenever it gets request from Repository Exec requesting build image file for the requested list of files, it fetch the source code files from Repository and provide build image files back to the Repository exec to get it send to the requester.

Interaction with other package: It communicates with the Repository Exec only and give it the build image file it wants.

7.4.6. SMS

We have already described SMS on high level. In this section I am going to describe the specific service I have used provided by SMS for this server.

Responsibility:

- In repository server, Repository Exec will provide the message data to send. SMS will use their data manager to find appropriate template to create message. It will create the message using message builder and send it to the destination using communication channel.
- It also receives the message and process them using message processer and take that data out using message template. It give this data to Repository Exec to process.
- Here, we are using NoSql database of SMS to store the Logs. Therefore, we are giving logging responsibility to Data manger. Any activity happened for any type message in repository, data manager will log that event. According to that event, if it is one of the configured event, then it will invoke agent attached with that event. This start the agent, and then agent will complete the work and send result message.
  Example like, when check in occurs, dependency analyzer agent will be invoked.
- In repository, we use SMS NoSql database to store Dependency table for each baseline. SO, if baseline is updated, then information is added in repository, but it still have data of old package dependency to support user who are using old baseline.
  We will have table like this:

Name	Dependencies	Version No.	Path	Test Status
Namespace1	P1,P2	1.0	../abc/	Pass
P1	F1,F2	4.0	../abc/	Fail
P2	F3,F4	2.0	../abc/	Pass

Here, we are adding path to metadata file since, sometimes we don't need just dependency information but we need some extra information. At that time, we will fetch metadata file rather than getting dependency directly from the table. Example like, as we have discussed, DependencyQuery message from client is one of the message like this.

Interaction with other packages: It interacts with the Repository Exec package for passing received message data and taking data from Repository Exec to send.
It also interacts with the Agents to invoke them when some activity or events occurred.

7.5. Activities

Repository is the busiest server in this SCF. It handles all the load of process related to file and data management. As part of this responsibilities, numbers of activities occurs on Repository server.
Below is the activity diagram of all the activities occurred on repository.

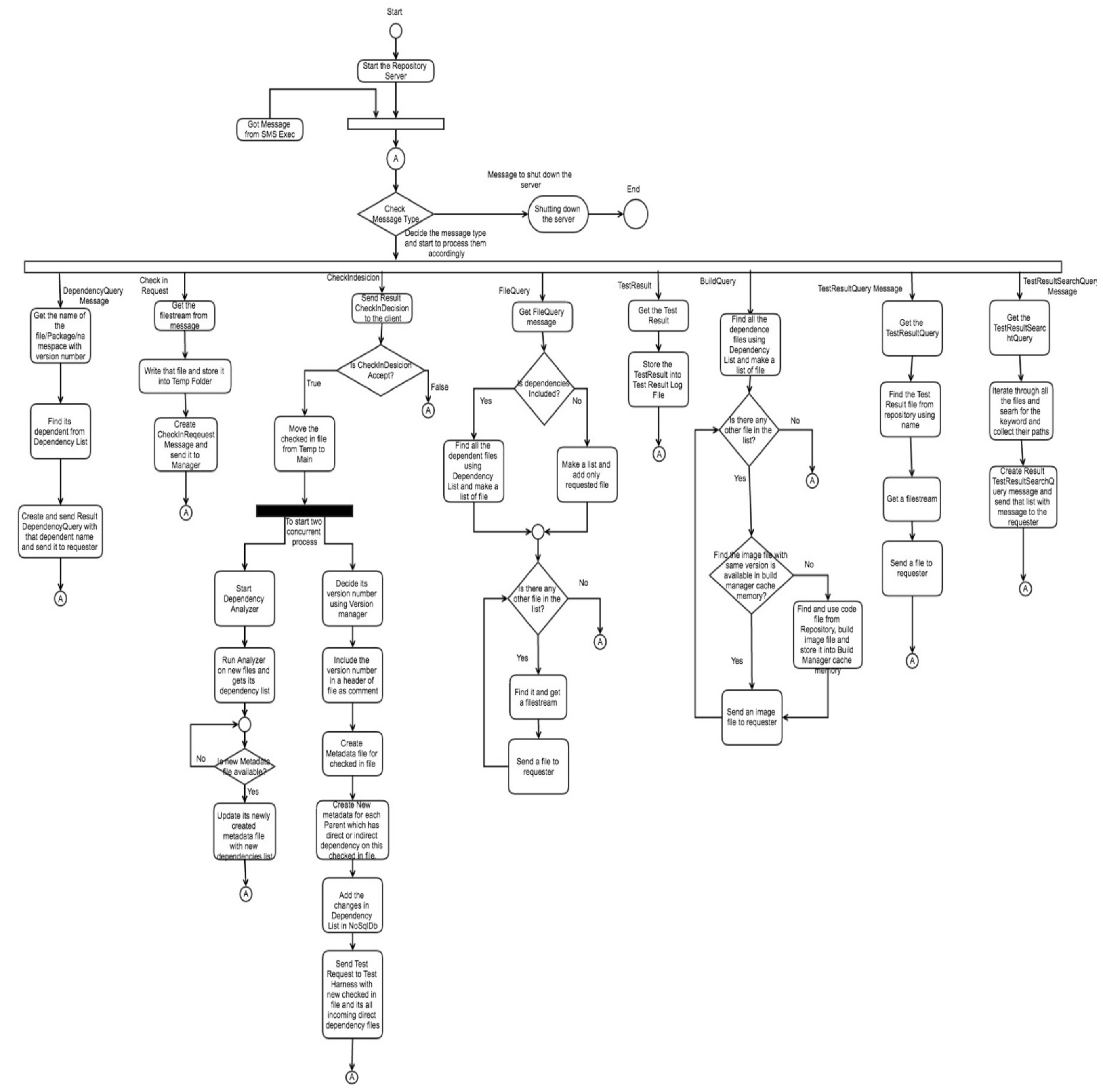


Figure 23 Activity Diagram of Repository

In repository, we can see from activity diagram that each activity invoked because of some type of received message.
Here, I am going to demonstrate each activity invoked by different type message receiving.

7.5.1. CheckInRequest Message

- When check in request comes to Repository Exec, it has the file to check in and some related documents and test driver to test that code. Repository will take all three streams and store it in the Temp folder. It will again create CheckInRequest message with same details and send it to the Manager notifying him/her about all the details of these files.

7.5.2. CheckInDecision Message

- After manager get the CheckInRequest message, manager go through all the files user wants to check in. If it seems good to manager then, he/she will accept the CheckInRequest. As CheckInRequest got accepted or denied, CheckInDecision message sent to the Repository to initiate the check in process.
- If it is accepted, then Repository Exec will move that file and its related data to Main baseline. It will start two concurrent process.
  1. It will decides its version number using Version Manager and include it in header of file as a comment. It creates metadata file for checked in file and fill all the information available from message.
     It creates new metadata file for each Parent file which directly or indirectly dependent on that file. It will send all the changes to Dependency List and add this information in the Dependency list.
     Then, it sends Test Request to Test Harness with new checked in file and it's all incoming direct dependency files to test all this files.
  2. On a new thread, it will start dependency analyzer which is configured agent. It is started because of check in event. The agent is invoked by SMS event manager as logger logs the check in activity.
     It will analyze the new file and collects all the tokens. It stores all the token into Tokenizer list. It will then search whether this file contains any of the token from tokenizer list except the token declared in the file. According to that result, it will create the list of dependencies of the file.
     It will wait till metadata file is available and after that it will update the dependencies in the file.

7.5.3. FileQuery Message

- When repository gets FileQuery message, first it will look for whether client has asked for dependent files or not. If it has asked for it, then it will make the list of all the dependent files. If client has not asked for it, it will still make a list with one file.
- The, it will iterate though the list, find each file and send it to a requester.

7.5.4. TestResult Message

- When repository Get the TestResult message, then it will get the stream from that message.
- It stores that file into Test Result log folder.

7.5.5. TestResultsQuery Message

- Test Results Query come up with test result file name which requester wants. We will search in the repository with filename.
- If we find the test result log files, then we will send that file to the requester. If cant find any file, then we will send notification saying "Requested file not found".

7.5.6. TestResultsSearchQuery Message

- When Repository get this message, it will fetch the keywords from the message,a nd start finding file having this type of keywords in the content or name and collecting its file path in the list of sting if that file have this node.
- At the end of the file, it will convert that list of string to string with file name separated by comma(;). It will send this data in Result TestResultsSearchQuery message and send it to the requester.

7.5.7. BuildQuery Message

- BuildQuery message is sent by Test Harness server since that is the only server which wants image file instead of code files to execute it.
- BuildQuery message contains the name of the file Test Harness wants. But, whenever build query comes, we always search for dependent file for that file and send all of them instead of just sending a single file. It will help Test Harness while loading that file and also while executing test on that file.
- Before building the file, it will search for the image file in cache of Build Manager. If it gets same file with same version, then it won't go and build it. But, if it doesn't, then it will fetch that file from repository, build it, store the image file in cache, and send it to the requester test harness.

7.5.8. GetBaseLineSummary Message

- When Repository get the message, it will just fetch the Dependency list, which we have maintained, from SMS. It will send Result GetBaseLineSummary Message to the requester attaching this Dependency List.

7.5.9. DependencyQuery Message

- When repository gets the DependencyQuery message, it will fetch filename and version number from the file. It will then use that data and find the file information from Dependency List which we have stored in SMS.
- It will get path of metadata from there, get the information from metadata file. Then, it will create Result DependencyQuery message and send it to the requester.

7.6. Critical Issues

- **Concurrent File Access**

  When multiple user tries to access the same file from repository, it will not allow concurrent access of that file and throw execution like the file is not accessible since someone is using it already.

  Solution:

  We can put one small try and catch wrapper around the code file accessing code. When it tries to access the file which is already in use, it throws exception. It waits for some time and again try to access it. If it again fails, then again follow the same procedure. It will do this for pre-defined number of time. If it get successful to retrieve the file, then further process will be started otherwise it will send notifies client and repository server.

- **Testing before check-in**

  According to the model which we have implemented, it seems not possible to go for automatic testing before check in. Since, we don’t have dependencies information while the files are in Temp folder. Once it gets checked in by approval, it will be moved to Main baseline where metadata file is created for that and; then we have dependencies information. So, after that, we automatically start testing of the newly checked in file and other files which depends on that file. It will get full proof workability of new file. But, after that if that tests get failed, then we marked that file as “Failed” test status in its metadata and developer who sent check in request get notified about that. How we can test it before check in to reduce the occurrence of this failure?

  Solution:

  As it seems automatic testing is not possible without check in, user can download the test driver file and source code file including its dependencies at local level. Then developers can do modification they want and do open check-in in the repository. Then, he/she should go for custom testing by providing information of both test driver and test code with its dependent files, and send that test request to test harness. If it gets successful, it should go for check in request. In same way, user can test other files which will be depends on the modified file after check in of that file.

- **Performance**

  Each time, when we want to find dependency of the files, we had to open the metadata file of that file and we had to get dependency data by reading that file. If the occurrence of this process is seldom, then we can ignore this. But, this is process which occure often. This reduce the performance of the Repository.

  Solution:

  We have created Dependency List table in our design to deal with this issue. Whenever any check in happens and it cause addition of the metadata files in the repository, we add dependency information from that newly created metadata file to our Dependency list to keep track on the baseline we are maintaining in the repository. With help of this table, we don't have to open metadata files and read it when we need dependency information. We can just refer this table to get dependency information for all the version of the baseline.

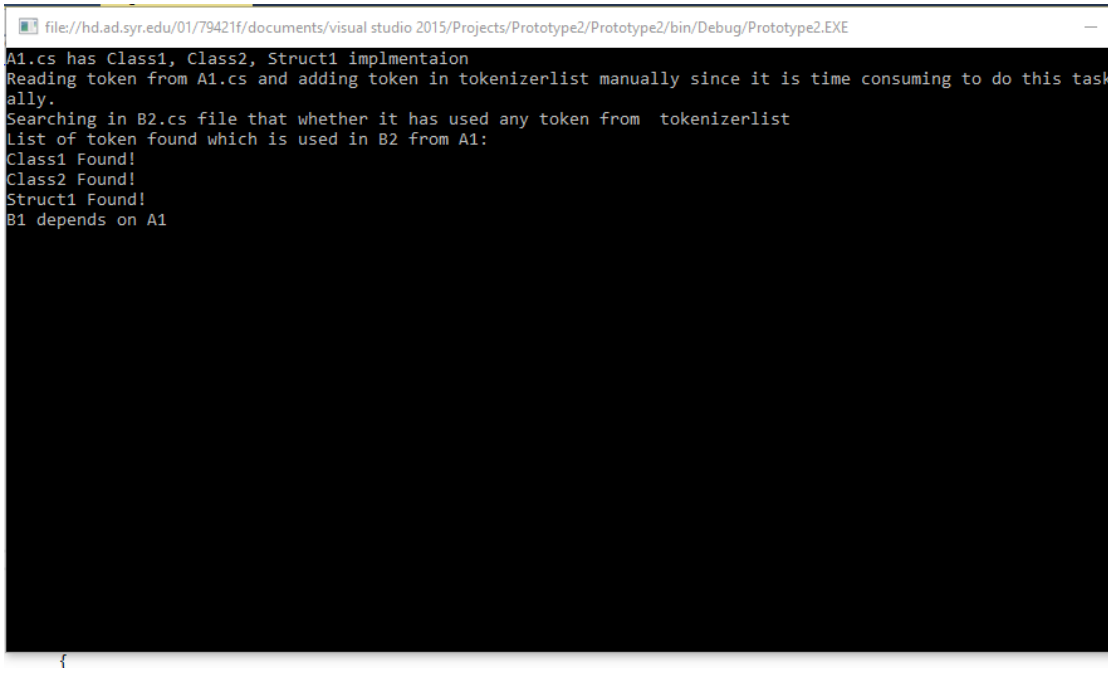
- **Workability of Dependency analyzer**

  Before starting this agent, we thought two approach to get the dependency of the files.

  1. Using reflection
     - This approach will not work when we don't have dependent files of file for which we are checking dependency. Therefore, this approach is useless since to get the type using reflection we need dependent file first. So, from bunch of files, we can find the dependent file. But, if that file is not available at that time, this approach will not work for our design.
  2. Using file tokenizer reading
     - As a **solution** of this issue, we thought about creating maintained tokenizer list and to find the dependency using that list.
     - For feasibility check, we have done prototyping on this,

       Prototyping

       - We have done prototyping for this Agent. Actually, building this whole stuff, who create tokenizer list and then serach for that token using streamer si very big and time consuming project. So, to clear the idea and feasibility, we have created small prototype in which we create tokenizer list manually by entering value
       - After that, tokenizer list which is created for package A1, we run that tokenizer to get the dependency of B2 using streamer. And we successfully find that token, that proves that B2 depends on A1. Additionally, our approach will work even after full development of Dependency analyzer agent.
       - Here is the screenshot of the output:



```
file://hd.ad.syr.edu/01/79421f/documents/visual studio 2015/Projects/Prototype2/Prototype2/bin/Debug/Prototype2.EXE
A1.cs has Class1, Class2, Struct1 implmentaion
Reading token from A1.cs and adding token in tokenizerlist manually since it is time consuming to do this task
ally.
Searching in B2.cs file that whether it has used any token from  tokenizerlist
List of token found which is used in B2 from A1:
Class1 Found!
Class2 Found!
Struct1 Found!
B1 depends on A1
```

Figure 24 Screenshot of prototype -2

- For more details, I have attached link to my code in Appendix.
- By doing this, we learn how dependency analyzer will work using tokenizer and file streamer.

8. Collaboration Server

Collaboration Server is the main server in the SCF and software management subsystem is extensively used by this server.
This server is mainly used for the coordination between different servers in the system and also between the teams currently working on the project.

It supports project management by storing work packages, work schedules, job description and by providing other collaborative tools like virtual display system. We already discuss it briefly in the architecture part.

8.1. Concepts

- Collaboration server is used by big teams to get their work organized and shared between them. This server is works as centralized server with which every client is connected.
- It helps in easy work package management since this data in collaboration server reflect actual process of each work package.
- Collaboration server is designed to give them facility to interact with each other by using collaboration tool like VDS. The Concept behind developing VDS, which is attached to collaboration server as tool is to set up functionality like view source code, documents, diagrams, sketches, and webcams to enhance personal and team interactions on clients' Virtual Display(VD).

8.2. Users and Uses

8.2.1. Developers and QAs

These users will mostly use this server for work package management. They are user od VDS which help them to set up their Virtual Display (VD) while meeting or any communication using VD.

8.2.2. Manager

Managers are mostly use this server for work package management. They use to see the status of each of them. Manager's also use VDS to set up their VD.

8.2.3. Uses

8.2.3.1. Collaboration between big team working on same project

It is very useful when team which are working on the project are too big. If it is just a team of four or five people, then they can survive and work without collaboration server. But, when we talk about hundredths and thousands, collaboration server will be necessary which will keep coordinating with each individual as a helper and guider. It will do all the organizing and management job on behalf of individuals, and let them concentrate on other important work.

Impact on Design:

It leads us to implement versioning policy and check in policy which helps to reduce the rework and helps to avoid chaotic situations in development.We also designed a collaborative tool like VDS which can be used to describe project related thing and to interact between development groups using webcam views, and chats.

8.2.3.2. Easy work package management

Work packages are most focused thing for Project Manager. For, big project and big teams, it would be like impossible for project manager to go through each work packages, and know the status of that item. Managing work packages easily, should be good usage of this collaboration server.

Impact on Design:

We have designed one agent who will do this cumbersome work on behalf of manager and create one brief summary. It will be sent to project manager as a notification on scheduled time, so that manager don't have to jump on each package by querying collaboration server for it.

8.2.3.3. Efficient resource handling-email to every person using data we have

Efficient Resource handling consider optimum use of resource available in the SCF. When numbers of developers are working on this system, we consider them as resource too. With the use of collaboration server, project manager will have data about which developer is working on which item. So, it will help him to take care that no resource is without work. As soon as one task, completes, either developers assign one work package to himself, or manager will do this work.

Project manager also use it as a notifier which can be used to notify the number of developers working in the team.

Impact on Design:

We have to keep data (ex. Email address, Name etc.) of all the developers working in the collaboration so when needed, manager ca sent group mail by extracting the details from server.

For easy resource managing, we are maintaining real time status for each work packages. Manager can easily query the work packages and get who is working on which package. He/she also can get this information by daily notification using notifying agent.

8.2.4. Cooperation binds individuals together to evolve solution (chats, or real time screen sharing helps when stuck)

Collaboration server provides a collaborative tool like VDS. For big teams, it will be very helpful if developers can collaborate and work by coordination of each other. It two developers who are working on same architecture, they can talk and help each other to get better design instead working in isolated team and submitting your work.

Impact on Design:

We have designed VDS and include functionalities like chatting, webcam meetings, real time screen sharing, etc. By using real time screen sharing, user can share their ideas by drawing on VDS. It will be real time screen sharing, so each of the user using the VDS from same collaboration server, can see it at same time.

VDS also provides chatting facility which helps users to chat and share their ideas on the work they are doing.

8.3. Structure

Here, we have divided this collaboration server in the following module. Below is the list of module and tools, we have used to design collaboration server:

1. Manager Summary Notifier
2. Collaboration Exec
3. SMS

4. Virtual Display System
5. Collaboration Repository

Below is the structure diagram for this server:

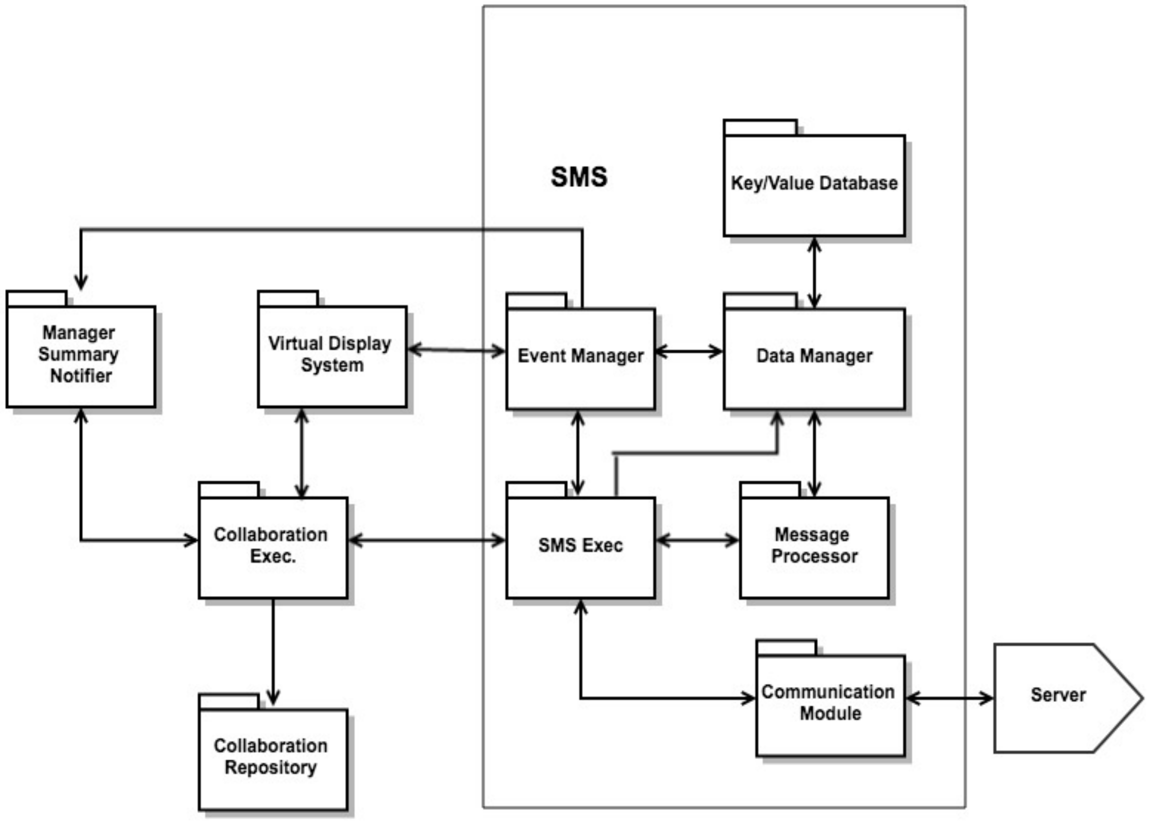


Figure 25 Structure diagram of collaboration server

We already have brief idea about working on collaboration server since I have describes it on high level in the architecture part. Here, we are going to demonstrated each server, is responsibility in detailed to give better idea about collaboration sever:

8.3.1. Manager Summary Notifier

It is an Agent. Agents are an interesting software artifact, it has given analytical abilities to perform automated tasks by its developer, and its applications are generally mobile. Manger Summary Notifier is developed to notify manager with summary report at scheduled time.

Responsibility:

Manager Summary Notifier has responsibility to collect all the data about work packages, its progress from collaboration server. It tells collaboration Exec to retrieve the data about current baseline and its test status of each package. It collects all that data, creates one summary report and send it to Manager (one of the client server) at scheduled time.

While collection the data of work package, it any of the work package has exceeds the invested time than what is decided, it will send the different notification to assignee which is working on the task.
This agent is invoked by SMS event manager. Whenever event manager get notification about scheduled time, it will invoke the agent to do the work we have discussed above.

Interaction with other packages:
It interacts with the Collaboration Exec, Event Manager of SMS.

8.3.2. Collaboration Exec

Collaboration Exec works as executive of this server. It manages most of the events and provide support to other packages. This will process all the messages, send them and receive them using SMS.
Responsibility:

- Its responsibility to provide information when any package wants it. If it is the information about work packages, then it will just fetch that information from collaboration repository and provide the information to the requester.
- If it is about baseline, then it will get send message to repository requesting information it wants. When it gets that information, it will send that information to the requester package.
- It also modifies the information of work package, whenever client try to modify it. Client send the message with work package information. If that package exists in the Collaboration Repository, then it will update the information, otherwise it will create new work package with that information. Then, it will send update the Work Package Dependency List.

Interaction with other packages:
It interacts with the Virtual Display System, Manager summary Notifier and Collaboration Repository.

8.3.3. Virtual Display System

It is the most important tool of the collaboration server. Actually, it can be viewed as collaboration tool or plug-in which enhance the experience of using collaboration server. It helps collaboration server to provide functionality like viewing source code, documents, diagrams, sketches, and webcams to enhance personal and team interactions.

Responsibility:

- Its actual responsibility to set up each client VD and get all of them in sync. So, that everyone will have identical screen across the meeting.

- Whenever SMS will get message to set up some VDS properties, it will log that event. At the same time, event manager will get that message using SMS data manager and send it to the virtual display system. The VDS system will have list of client who are in the meeting. It will create the message about this change and send this message to each client who are in the list.
- So, basically it provides functionality for sharing real time screen, document, codes, chats between users.

Interaction with other packages:
It communicates with event manager and Collaborative Exec.

8.3.4. Collaboration Repository

Collaboration Repository is used to store information about all the work packages.

Responsibility:

- When Manager Summary Notifier gets invoked, it will get dependency information of work packages and its paths. It will request Collaboration Exec to send the information by fetching work package from that path, which points to collaboration repository.
- Basically, Collaboration repository is used to store the files which is retrieved by Collaborative Exec by using path it gets.
- Work package has all the information about that work. Example like, Work schedule, assigned individual, progress, estimated time to complete, dependent task.

Interaction with other packages:
It interacts with collaboration Exec, and provide the files it wants.

8.3.5. SMS

We have already described SMS on high level. In this section I am going to describe the specific service I have used provided by SMS in this server.

Responsibility:

- SMS is used by this server to store the dependency information and path to work packages. This will help to find the work packages without any querying the Collaboration repository.

- We are using SMS NoSql database to store the logs of each activity. That logs help to send the notification of any activity to event manager to start the Agent associated with that event.
  Example like, if the current time will be the time which is scheduled to send summary information to manager, SMS use that event manager to invoke the Agent which will create summary report and send it to the Manger.
- Apart from this, SMS has common responsibility to send data, which is getting from Collaboration Exec, by creating message using message template and message processor; and send it to the destination.

Interaction with other packages:
It interacts with the Collaborative Exec only.

8.4. Activities

Below, I have demonstrated the activity occurring in the collaboration server.

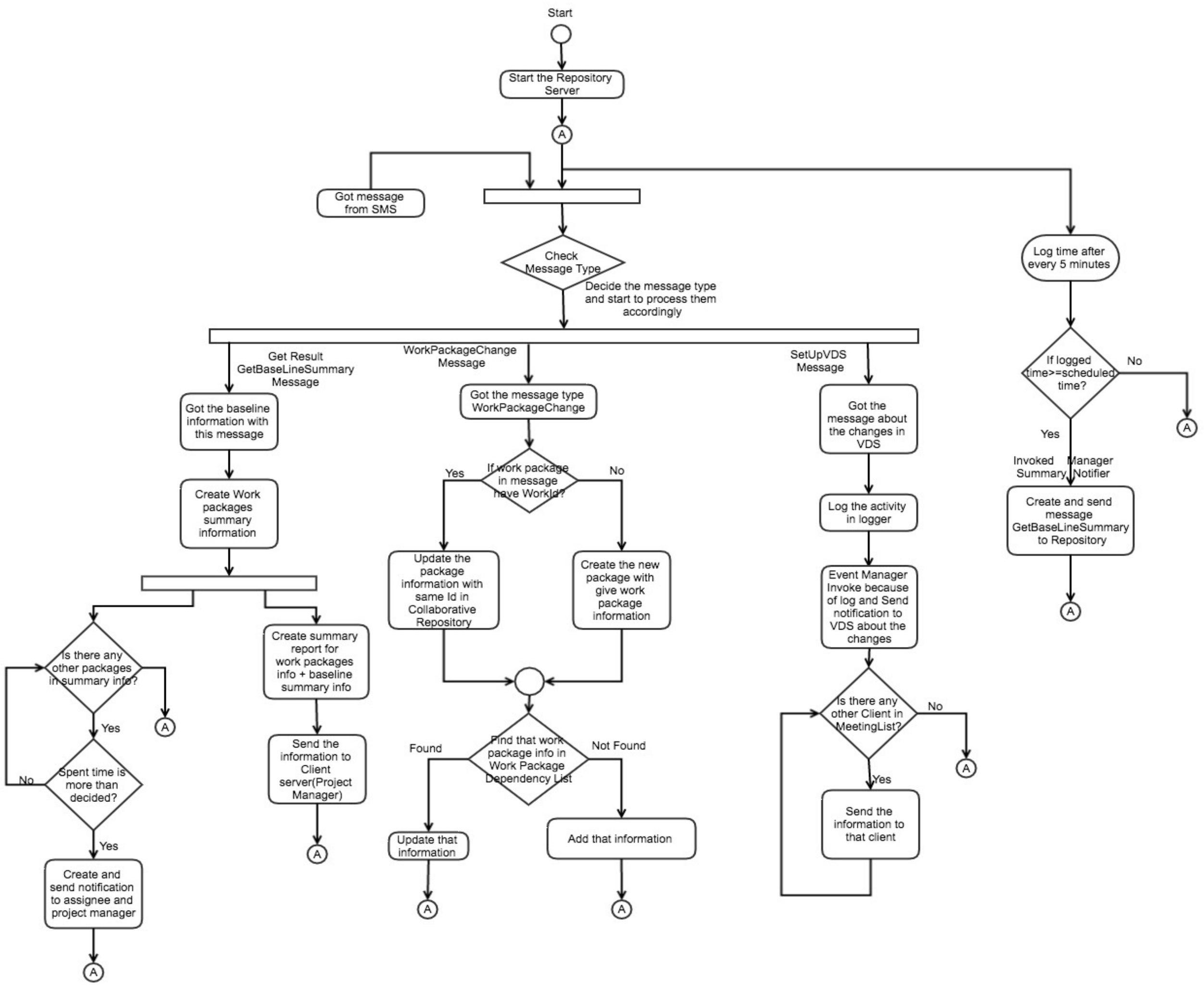

Start
Start the Repository Server
Got message from SMS
Check Message Type
Decide the message type and start to process them accordingly
Log time after every 5 minutes
Get Result GetBaseLineSummary Message
WorkPackageChange Message
SetUpVDS Message
If logged time>=scheduled time?
No
Yes
Invoked Summary
Manager Notifier
Create and send message GetBaseLineSummary to Repository
Got the baseline information with this message
Create Work packages summary information
Got the message type WorkPackageChange
If work package in message have WorkId?
Yes
No
Update the package information with same Id in Collaborative Repository
Create the new package with give work package information
Got the message about the changes in VDS
Log the activity in logger
Event Manager Invoke because of log and Send notification to VDS about the changes
Is there any other Client in MeetingList?
Send the information to that client
Is there any other packages in summary info?
Spent time is more than decided?
Create and send notification to assignee and project manager
Create summary report for work packages info + baseline summary info
Send the information to Client server(Project Manager)
Find that work package info in Work Package Dependency List
Found
Not Found
Update that information
Add that information
A

Figure 26 Activity Diagram for Collaboration server

Here, I have defined activities I have demonstrated in the activity diagram. I have categorized the activity and demonstrated them only which is enough to describe whole activity diagram.

8.4.1. Activity for getting the Result GetBaseLineSummary Message

- When Collaboration Server get the Message type GetBaseLineSummary Message, it will get Copy of Dependency List from Repository Server which will have information like,
  Name= name of the file/package/namespace,
  Dependency=list of the dependency this file/package have
  Test Status= status of the primary test
- It will be used to create the summary of the whole baseline Repository server contains. After that, it will go through each work package collaboration server contains and add them into that summary file.

- It will look through that summary file and if any work package has invested time exceeded than decided time. It will send notification to assignee and manager both for that work package.

8.4.2. Activity for WorkPackageChange Message

- When collaboration server gets the WorkPackageChange message, it will take that data and look for the work package id field in that message.
- If work package id is NULL, that means it is new work package. Therefore, it will store that work package and assign in Collboration Repository and assign it new work package ID.
- If there is work package ID, then it will search for that ID in Work package Dependency list, and get the path of that file. It will fetch that file from that path and update that file.
- After that, it will again go to Work Package Dependency List and try to find the same work package using its ID. If it is able to finds, then it update the information, if it is not able to find, that means its new addition. It will add that information in that maintained list.

8.4.3. Activity for SetUpVDS Message

- SetUpVDS is message is basically sent to start the VDS meeting or to change some properties of VDS. Like layout of the windows, to open any file, to show any diagrams, to set up webcam views, to add or remove any participant from the meeting.
- Whenever collaboration server gets SetUpVDS message, it will go access the data from the message and send that data to Virtual Display System. VDS will have the list of the participants.
- If it is about adding a new participant, then it will send all the data VDS have as of now, to that client to set up the virtual display at the client side. If it is any other property change, then it will send this data to all the connected client.

8.4.4. Invoking Manager Summary Notifier

- Collaboration server will log time after every 5 minutes in SMS. When it logs time more than or equal to scheduled time, it will invoke the Manager Summary Notifier.
- After getting invoked, it will create and send GetBaseLineSummary to Repository server to get the summary of the whole baseline Repository contains.
- When it receives the Result GetBaseLineSummary message, it will start the activity we have mentioned in previous sections.

8.5. Critical Issue

- **Latency in VDS communication**

Virtual Display System is very important collaborative tool which is used for real time activity. If it has latency in communication, then we have to suffer with less interactive system which won't be preferable. When users want to go for real time screen sharing for sharing codes, documents, diagrams; latency will not affect our system that much. But, when user(client) use drawing tool, it will make the system totally useless since latency makes this functionality totally not interactive.

Solution:

When user tries to use drawing functionality of VD, we should use http messages instead of WCF channel. It will give faster communication than WCF. It will help to resolve the latency issue.

Impact on Design

We have to change the communication channel we are using for this design and add http messaging channel for just drawing utility. We will send little messages about the dimension to the destination, it will be faster and affective. Since we can send each dot from the drawing user is making, to the destination, and it will look like real time drawing. If we add number of VD with this type of communication, then latency time will be increased again. So, we will restrict the clients size up to two only. So, only two clients can interact using drawing tool, they can't add more than two clients during meeting which used drawing tool.

9. Test Harness Server

It is a main server of this application, which plays important in test execution.

Test Harness is the main server or hub where all the clients will get connected and start sending their test request for execution. Test harness will execute all the test request using its multithread and pass each of this result or logs to Repository server to store it. Therefore, we can use it afterwards whenever it needed.

9.1. Concept

- Test harness is designed to enable automation testing for this SCF. It essentially takes test driver file and test code, it run that test driver file on test code file and return the result.
- It enables the functionality to run custom testing. Any type of testing user wants to execute; he/she have to write test driver files for it. Test Harness will need build file for that codes, it will load their assemblies, find the files which are inherited from ITest interface, and run their test function to execute the test. It will get the test result and send it to requester client and repository to store it.

9.2. Uses and Users

9.2.1. Developers:

They use test harness server for testing check-in file. They use results and logs to scrutinize the tests and detect erroneous zone in source file. It is possible that test driver may have bug, looking into log can help them to identify bug.

Impact on Design:

After checking in of any file, repository will automatically send the test request to test the new checked in file. That test request also contains the files which depends on that checked in file for testing purpose. Hence, it will test all those files, and it get result of that test request, then repository update the metadata “teststatus” of that newly checked in file according to that test result.

9.2.2. QA Team:

They use test harness run custom tests using test suits. They run that long and complex tests and scrutinize each test’s result using result and log provided by test harness server. It helps them to confirm the properness of functionality.

9.3. Structure

Test Harness Server contains this packages to perform the functionality.

1. Test harness executive
2. Test Directory Generator
3. Timer
4. Child AppDomain Manager
5. Load&Execute
6. XML decoder
7. Blocking queue
8. SMS

Below is the package diagram of Test Harness Server descripting all the packages:

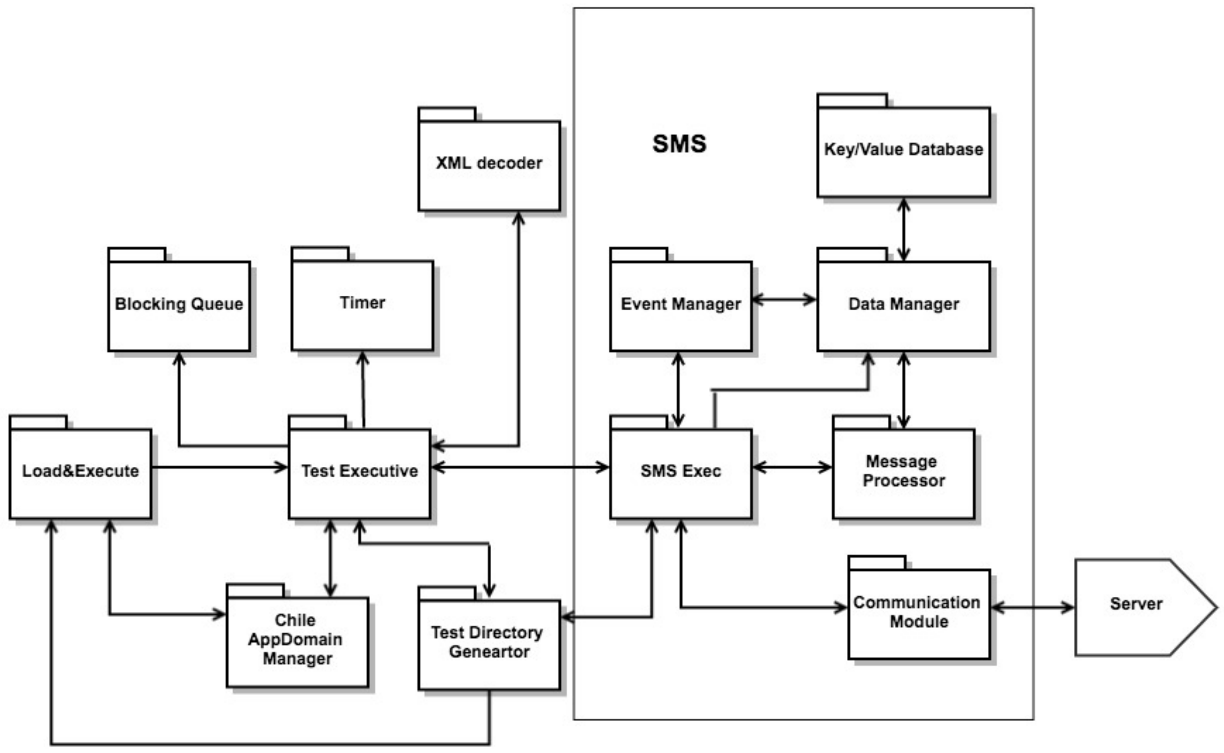


Figure 27 Structure Diagram of Test harness

9.3.1. Test Harness executive

It is main executive package, which is initiating requested test execution in test harness server. It will get test request from Message processor, and enqueue it into main blocking queue of the Test harness. Main Blocking queue is shared queue between each and every clients. Any of the client send the test request, it will be enqueud it this Main Blocking Queue. Therefore, Main Blocking queue will be collection of test request sent by different clients.

It will spawn threads to dequeue the test requests and send each of them to XML decoder to get all the details by parsing all the important data. Then it will send this request to TestDirectory generator to create test directory. Then, It will send the name of that directory to child AppDomain Manger for further process.

It also associated with Timer. It starts the timer when test execution is started. It stops the timer when test execution completed or test results are available. It helps to track how long test request takes to complete, so it is basically total execution time.

After test execution is completed, it will get the Test Results.It unloads the child Appdomain.It deletes that temporary TestDirectory and send Test Results to requester client as a notification and Repository to store it.

Responsibilities:
It will create new thread to handle each test request by dequeuing it from Blocking Queue, then thread will be headed towards creating TestDirectory. It will send that name of the TestDirectory to the Child AppDomain Manager for further process. Its responsibilities also consider placing the whole test execution Test Results into repository, which is very important thing.
It also played important role in creating and deleting temporary test directory for every request.

Interaction with other packages:
Test Executive will communicate with SMS to send messages or to receive processed messages.
It also communicates with Blocking Queue, to dequeue the Test Request from it.
It will decode it with the help of XML decoder. Then, it will interact with TestDirectory Generator by sending decoded XML data. It also communicate with Child AppDomain Manager to create child AppDomain.
It also communicates with load&Execute to get the Test Result after test execution got completed. It interacts with timer for when to start and when to stop it while test execution is running.

9.3.2. XML Decoder

This module decodes/read the XML test request coming from Test executive, unpack all the information, and send it to the Test executive again. This information is further processed into Test Executive. It will get this type of XML test request, which it has to decode or parse.

Ex. <?xml version="1.0" encoding="utf-8" ?> <testRequest name="TestRequest1">
<author>Arpit Shah</author> <authortype>Developer</authortype>
<test name="First Test"> <testDriver>TestDriver1.dll</testDriver>
<library>TestCode1.dll</library>
</test> <test name="Second Test">
<testDriver>TestDriver2.dll</testDriver> <library>TestCode2.dll</library>
</test> </testRequest>

Responsibilities:
Its main responsibility is to decode the XML and send back that parsed information to test executive.

Interaction with other packages:
XML decoder interacts with just Test Executive.

9.3.3. Timer

Timer package is mainly used to track the processing time of any activities. It will be used to measure total execution time or communication latency for any test request.

Responsibilities:
It is used to measure the processing time. It will invoke high resolution timer to record start time when test execution starts, and record stop time when test execution got completed.

Interaction with other packages:
It interacts only with test executive which notify timer that when to invoke it and record start and stop time.

9.3.4. TestDirectory Generator

It is used to create temporary directory for each test request. It will use author name and DateStamp as a key. It will use it to create that directory with unique name(i.e. AuthorName_DateSatmp). It will get deleted after test execution completed.

Responsibilities:
It's used to create temporary Test Directory with name AuthorName_DateSatmp. Then it sends request to Repository server to send BuildQuery the DLLs which is mentioned in the test request information. It will also have delete function, which is used to delete this directory when test execution completed. Conclusively, its main responsibility to provide required DLLs to test Harness during Test Execution and if it can't get required DLLs from repository, it will send error message to client.

Interaction with other packages:
It mainly interacts with Test executive and SMS Exec.

9.3.5. Child AppDomain Manager

It is very small class which is used to manage all the test execution in child AppDomain. It gets name of the temporary testdirectory from the Test executive and invokes the process on same thread. It will load Load&Execute package and initiate the process in child AppDomain which give isolation for each test execution.

Responsibilities:
It creates child AppDomain and invoke further process/test execution in it. Its main responsibility is to give isolation to each running test execution on different threads. Therefore, even one of the test execution gets failed, it won't affect the other test execution. It also unloads the child AppDomain and send notification to test Executive about the completion of the execution.

Interaction with other packages:
It interacts with Test Executive and Load&Execute package.

9.3.6. Load&Execute

This package is most important package of this Test Harness since it is executing test requests in it and storing its result in Test Results object. When Child AppDomain Manager loads this package and start calling methods from this package, it will start execution. It will load all the DLLs files from temporary TestDirectory, find all the test driver and test codes, execute Test Driver on the particular test code and return the result of the test execution. It will store all the Test Result into Test Results object.

Responsibilities:
Load&Execute package is providing the functionality to load the DLLs(required Test drivers and Test codes) from particular folder locations and load them into child AppDoamin. It executes the Test Driver and gets the results. Simultaneously, it will log test result for each test execution. After, the test execution completed, it will combine all the test result into Test Results object and send it to the Test Harness Exec.

Interaction with other packages:
This package mainly communicates with Child AppDomian Manager and Test Harness Exec. It also interacts with TestDirectory Generator to fetch the required DLL files from the temporary created directory.

9.3.7. Blocking queue

It is main blocking queue of test harness which will be shared queue. It can be used to enqueue the test request coming to Test Harness.

When test harness wants to process it will dequeue it, process it, execute it and send the test result.

Responsibility:
Its responsibility is to work as container who can let enqueue the test request in the shared blocking queue. It also has to deal with thread, so if there is not test request in the queue, and any thread tried to dequeue it, it should be blocked till test request won't get enqueued in the blocking queue.

Interaction with other packages:
It interacts with Test executive only.

9.3.8. SMS

We have already described SMS on high level. In this section I am going to describe the specific service I have used provided by SMS for this server.

Responsibility:

- SMS has common responsibility to send data, which is getting from Test Harness Exec, by creating message using message template and message processor; and send it to the destination. It also communicates with Test Directory Generator, and helps it to download require build files from Repository server.
- We are using SMS as Logger in Test Harness Server which log each and every activity occurs in Test Harness. It can be helpful to debug if Test Harness Server is facing any problem.

Interact with other Package:
It communicates with Test Executive and provide them received messages data to process. It also gets data from Test harness Exec(Test Results) to send to Requester.
It communicates with Test directory generator and download the required build file, which test execution will want.

9.4. Activities

Below, I have demonstrated the activity occurring in the Test Harness server.

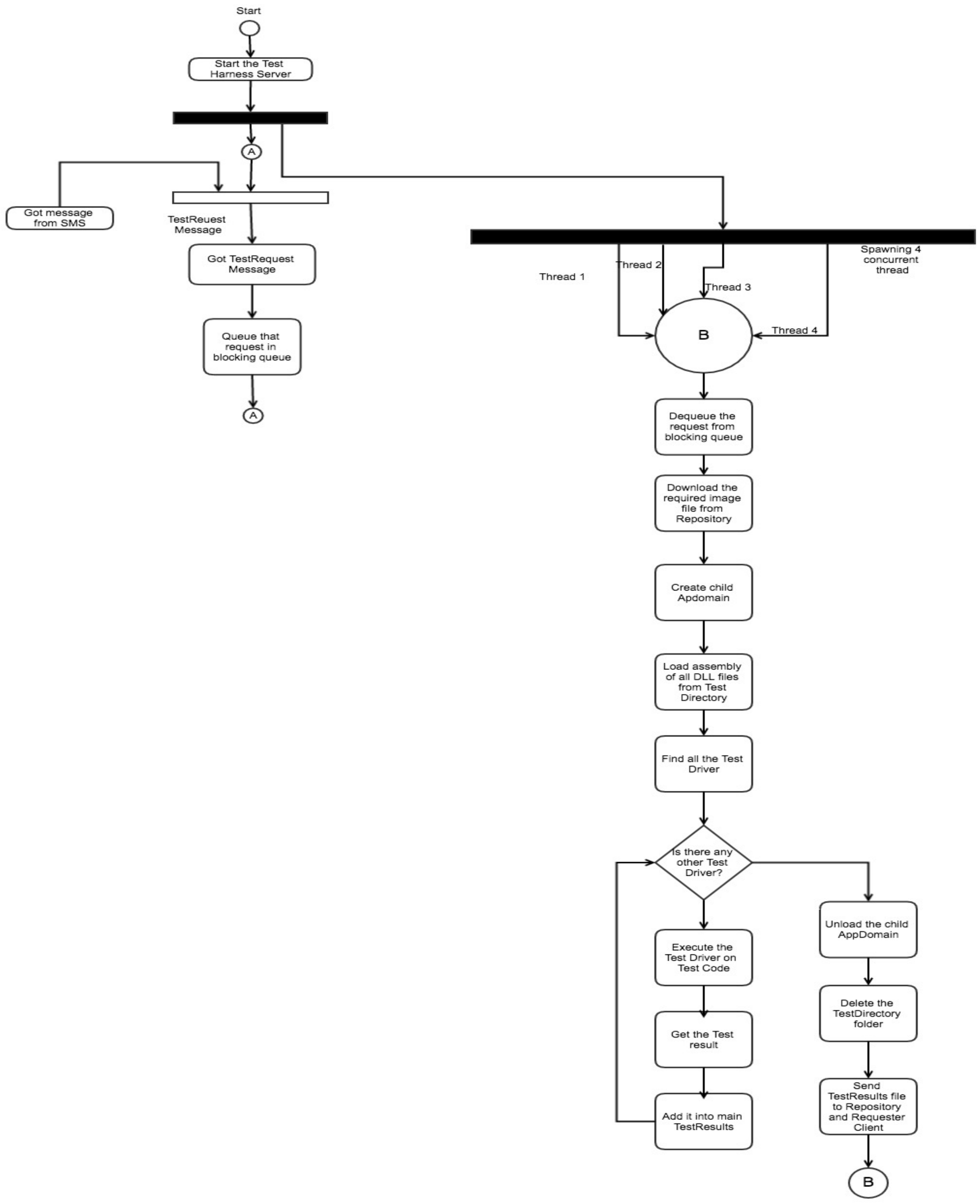

Start
Start the Test Harness Server
A
Got message from SMS
TestReuest Message
Got TestRequest Message
Queue that request in blocking queue
A
Spawning 4 concurrent thread
Thread 1
Thread 2
Thread 3
Thread 4
B
Dequeue the request from blocking queue
Download the required image file from Repository
Create child Apdomain
Load assembly of all DLL files from Test Directory
Find all the Test Driver
Is there any other Test Driver?
Execute the Test Driver on Test Code
Get the Test result
Add it into main TestResults
Unload the child AppDomain
Delete the TestDirectory folder
Send TestResults file to Repository and Requester Client
B

Figure 28 Activity Diagram for Test Harness

Here, I have defined activities I have demonstrated in the activity diagram. I have categorized the activity and demonstrated them only which is enough to describe whole activity diagram.

9.4.1. Activity for receiving Test Request message

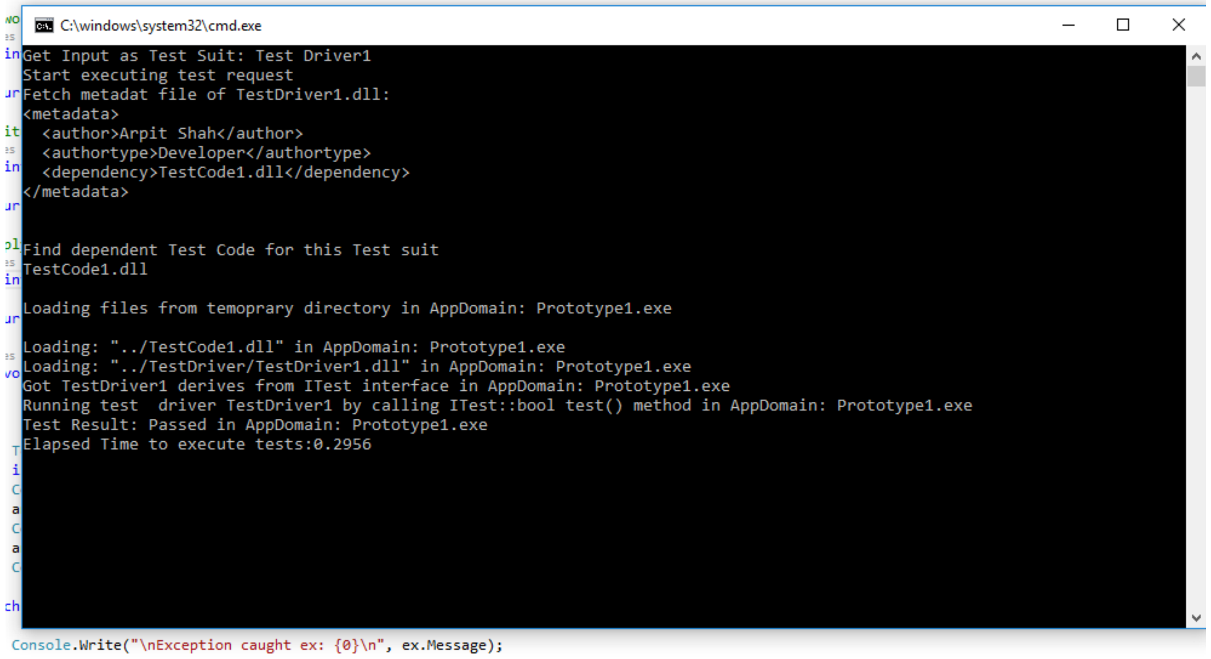
- When Test Harness get the message about Test Request from any client, it will just queue up that message in the blocking queue of the Test Harness.

9.4.2. Activity about Test Execution

- Test harness spawn 4 thread on Test Harness Blocking queue. All of them will get blocked till any request comes in test harness's blocking queue. Whenever any of the Test Request is queued up in that blocking queue, one of the thread will dequeue it and start execution.
- **Dequeuing test request from Main Blocking Queue**
  It will continuously dequeue Test request from Main Blocking Queue. This test requests mainly contains details of test name, test drivers, test codes etc.
- **Decoding XML Test request**
  Test executive will send XML test request to XML decoder to parse the data from each XML. This process will be done on same thread. So, that thread will get this XML test request decoded.
- **Request for required DLLs**
  Test executive will call TestDirectory Generator for this process. TestDirectory Generator will have parsed data object of that XML test request. It will create temporary TestDirectory which named as a AuthorName_DateStamp. Then, it will send BuildQuery request to repository to send all this required files with its dependent files. When it will get those files, it writes those files into this TestDirectory and store it over there.
  If it can't get any of the files, it will send error message to client about unavailability of DLLs.

  #### Prototyping

  - Here, as a process, repository will get all the dependent files on the base of test driver's metadata file. It will fetch name of all the dependent file and send it to the test harness, then test harness will load that file and run test execution.
  - To check the feasibility, of this reading metadata file, finding dependent, sending them to test harness and then test execution, we have done prototyping for this process.
  - We have created the one program and provide it one test driver with metadata file and one test code to run execution on them. We are reading metadata file of this test driver, finds the dependent test code, and then load their assembly into the appdomain and we were successful to run the test by doing this process.
  - By doing this, we learn how to read metadata file of the test driver to find its dependency.
  - Here, is the screenshot of the output of prototype.



```
Get Input as Test Suit: Test Driver1
Start executing test request
Fetch metadat file of TestDriver1.dll:
<metadata>
  <author>Arpit Shah</author>
  <authortype>Developer</authortype>
  <dependency>TestCode1.dll</dependency>
</metadata>


Find dependent Test Code for this Test suit
TestCode1.dll

Loading files from temoprary directory in AppDomain: Prototype1.exe

Loading: "../TestCode1.dll" in AppDomain: Prototype1.exe
Loading: "../TestDriver/TestDriver1.dll" in AppDomain: Prototype1.exe
Got TestDriver1 derives from ITest interface in AppDomain: Prototype1.exe
Running test  driver TestDriver1 by calling ITest::bool test() method in AppDomain: Prototype1.exe
Test Result: Passed in AppDomain: Prototype1.exe
Elapsed Time to execute tests:0.2956
```

```
Console.Write("\nException caught ex: {0}\n", ex.Message);
```

Figure 29 Screenshot of prototype 1

- For codes, we have attached the link to the codes in appendix.

- **Creating Child AppDomain Manager**
  When TestDirectory is generated, test harness will create child AppDoamin Manager and start further test processing into it to give isolation to other process, which are running on some other threads. It will load the Load&Execute package and start calling the methods to load and execute those DLLs.
- **Loading All DLLs**
  Child AppDomain will support test execution process after this point. All the required DLLs which are in the TestDirectory will get loaded in this particular child AppDomain. It will also search for Test Drivers by using reference of Interface (Note: we are using ITest interface in our case.) to execute them.
- **Execute test driver file**
  Test Harness will execute this test drivers and test the functionalities written in Test code. It will perform this execution and store the results into the object of TestResuls. Then, it stop timer.
  After that, it will unload child AppDomain since test execution is completed. It will also delete that temoparary TestDirectory that we created using AuthorName_DateStamp as a key. It will then send TestResults to Repository and send notification to client about completion about test execution.
  At that time, client will have name of the author and datestamp from recorded time, it

will simply request repository with that name to get Test Result files.

9.5. Critical Issues

9.5.1. Performance

Performance will be a big issue for all the type of users whether it is Developers or QAs as I have discussed in use cases. When numbers of users increase, number of thread workers will also increase for the test executions. It will harm the test harness execution speed very badly. So, on one hand threading is increasing performance of test harness execution, but on other hand excessive amount of thread decreases performance.

Solution:
We will implement thread pool to limit the amount of threads, which can spawn in application execution. This thread pool will block the numbers of thread if it reaches on its threshold value. Therefore, the number of running thread will be same at that time, and it will not affect the performance.

9.5.2. Threading deadlock

When multiple test request will run on different thread and try to access locked shared resources, this situation occurs. If one thread has locked any shared resources and another thread is waiting to access, it and something occurs so that the thread holding the locked item is waiting for the other thread to execute.
Solution:
Simple solution to avoid threading is to avoid complex design. We also have to take care for lock ordering, improper lock ordering will lock that resource and then deadlock occurs.

9.5.3. Memory Management

We are creating temporary Testdirectory for each test request and storing required files in that folder. This will increase memory consumption of test harness server since we are creating this file on this server. We are fetching all the required DLLs and storing it into it. When numbers of threads will come and start test request execution, memory consumption will be started increasing.

Solution:
As a solution, we are deleting that temporary Testdirectory, but it will affect after the thread complete the execution and we need its solution while execution is running. Therefore, to control

memory problems, we have to design thread pool by considering Test Harness server memory capacity and processing speed.

9.5.4. Loading Identical DLLs into

When one thread has one test request to execute and it has numbers of test case or test driver, which are repeating in the test request file because of some reason. Our test harness does not know about which assemblies are loaded previously in particular child AppDomain, so it will try to load it again, it throws exception.

Solution:
We will have to implement some exception catching handling to continue our test execution. Since we already have that DLLs loaded, so even if we continue the process by catching exception, it will not create any problem in future.

9.5.5. Invalid XML files

It should be very common, that a user may provide invalid XML files or not enter proper input. When, parser will try to parse it and execute the request, it will throw some exception because of this improper input or XML files.

Solution:
When XML parser is trying to parse the files and get the requests necessary to send, it must be very good, to check whether the input is valid. If input is not valid, it should not let user to go ahead with this invalid input.

9.5.6. Execution time is smaller than time unit

Sometimes test execution time is very small that we cannot measure it with used time unit. So, even test execution is done, it will show 0 seconds as execution time with some test results which is very confusing to any user.

Solution:
To keep track of every single activity precisely, we use High-resolution timer. This will give us more precision on time measuring. After the high-resolution timer is used, if there still be a 0 elapsed time, it may reveal a bug in the design, because the whole procession should never smaller than 1 microsecond/nanosecond.

10. Future Expansion

This system can be expanded by manier ways since It has numbers of servers which do different different activities. We can improve the performance as future expansion. We can also add some more functionality using this system and server only.

Since we are keeping all the details in repository and we are logging every activity in log files. That means, we have bunch of data, which can be used by doing different type of data operation. As an example we can Developer rating system.

Developers rating system: The repository maintains record of all successful and failed check-ins from any developer. These records can be used to rate the developers. In case of bad performance, the project manager or team lead can make decisions. The repository records can be very useful for this purpose.

We can use that logs expand this SCF with some more functionalities.

11. Appendix

-Both prototypes is in the same folder as this document.

12. References

-wiki pages
-Jim Fawcett handouts