Software Collaboration Federation

Project 5 CSE-681

Shishir Bijalwan
12/7/2016

Contents

Executive Summary

With ever growing dependency on software in the modern world, it has become indispensable to have a bug free software application. Software development is complex and often a difficult process requiring synthesis of many disciplines from requirement gathering to deployment and maintenance. The complexity involved with software development increase more if we need a big team to develop the application as managing itself become an important task.

In this project we have developed a Software Collaboration Federation (SCF), which makes the process of software development much easier. It provides inbuilt utilities which helps a person with any designation (Developer, Tester etc.) to do his task more efficiently by automating the process. For example, automating the part of integration testing or the automating the part of project status report. As software development cycle involves many stages and adding all the utilities for them in one system would have made its working complex and design would not have been scalable. So we have divided our federation into many sub systems where each sub system has his own set of responsibilities. These subsystems are:

- Test Harness: To provide utility of integration testing
- Repository: Has the responsibility to store data for SCF
- Build Server: Provides pseudo production environment for compiling code
- Collaboration server: Provides the management capability to the SCF
- Virtual Display Server: Provides a common platform for discussion
- Client and Virtual Display System: Provides GUI to interact with above servers

We have talked about the activities and package structure of each of these system individually in detail in this document. The current version of SCF will be used by the Project Manager, Technical Architect, Developer, Quality Assurance engineer, Business analyst, Subject Matter Expert and the customer. Each of the user have their own access level. For example, a customer can only use a Virtual Display System to attend meetings. He won't have access to another utility of the SCF.

During the development of software collaboration federation, we have taken special care of some critical issues which are vital for proper functioning of the application. Some of them are:

- Communication: Communication becomes a critical issue in the working of large application. As the number of type of message that flow in the system increases exponentially.
- Load: When we talk about load it is the frequency at which the servers are sent request i.e. number of request a server has to handle. If the number is too large it can bring the performance down.
- Process Flow Management: SCF provides so many utilities but what should be the sequence of steps using it. For example, the Project manager has requested for logs

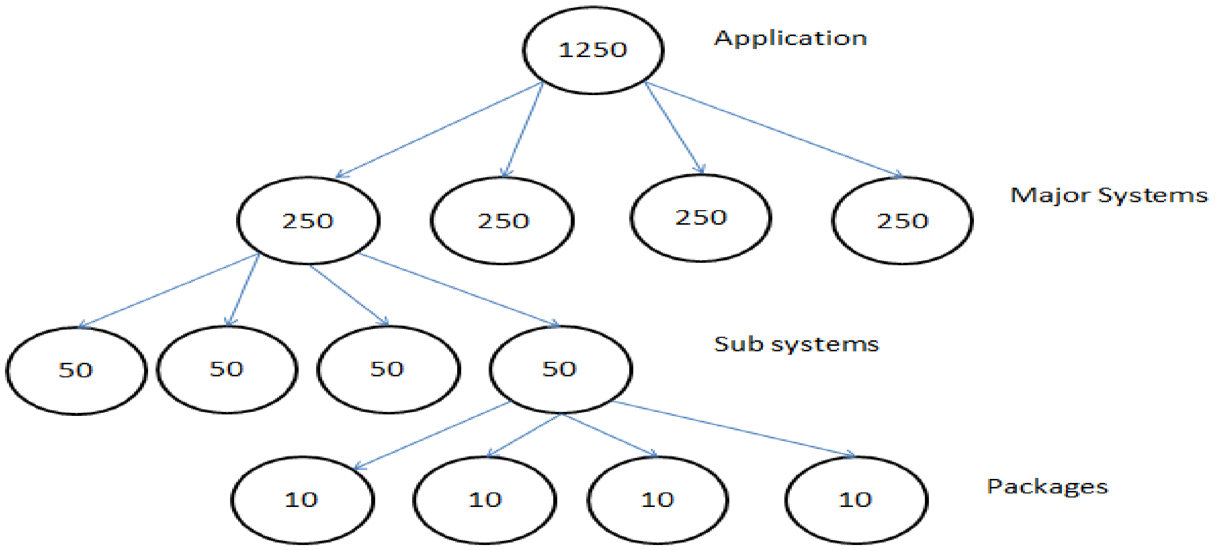
before the developer could run the test using test harness or design team has started designing before the requirement document is finalized.

- Ease of use: All the utilities of an application are useless if they are not easy enough to be used. If the user feels that using the application is difficult then he will never use it.

Federation

Software development is the process of computer programming, documenting, testing and bug fixing involved in the creation of software product. The software Collaboration Federation has been designed in such a way that makes software development process efficient. Before going into Federation let's take a look on some common issues faced during development without federation:

- Communication: In cases when the teams are at different location, situations are very common when some information is not communicated to teams at other location and can lead to development in wrong direction.
- Testing: To understand the need utility of automating testing let us take an example. Suppose you work in a big software company and developing banking software for a Bank of America.



Application can be divided into major systems. Each major system will have many sub-systems, which in turn will have many libraries and packages. So for example we test each package 10 times. This will make us test each sub system 50 time, which will make us test each major system 250 time and will make us test the whole Application around 1250 times. Testing so many times will bring down the efficiency of the team down.

- Blasting Production deployment: During the deployment of software in production environment we face many unseen issues. These issues could be environment issues example database version available at production does not support particular feature or any other compatibility issues.
- Product status monitoring:

Concept

The basic concept behind the development software collaboration federation(SCF) is to provide a tool to can make software development efficient. To picture clear, we will take each phase of development and understand the role federation plays in it.

- Requirement gathering: All the software development methodologies used today require the step for requirement gathering. In some development techniques such an Agile, a customer side representative needs to be involved in whole development process. The SCF virtual display system helps in this phase of software development. The VDS allows the customer, business analysist and subject matter expert(SME) to sit at different locations and do discussion about the software requirement. It also helps in documenting these requirements save them in server, from where any other team member can access it. It reduces time lost in travelling.
- Designing: The Tech architects and lead developer are involved in this phase of software development. The VDS again helps in discussion on designing the software application. They can use features of VDS such as Document display, video conference, chatting and Common writing area for discussing and record maintenance. The major advantage of using SCF here is that no team will have old design document in case it has been updated.
- Development: Once the design process is over. The manager or the social tracker decides discuss and assign task to individuals. Here again tools like task manager and project status manager of SCF's collaboration server comes into use. Using task Manager, the social tracker can keep a check on the progress of development. The developer updates his daily progress in the task manager tool, which is monitored by the social tracker or Manager.
  In large project in general a task depends upon the completion of some other task. The project status report can be used by the developer to know if the dependent task has been completed or not so he can proceed. The developer uses the facility of getting the latest design document from the collaboration server.
  Another major issue that development faces is approvals. The SCF provides a common platform where project approvals can be taken. It helps in escalating the process of approval and monitor the time lost in it.
- Testing: Integration testing is a major issue faced by large project. Manually doing that task bring the efficiency of the team down. The use if Test Harness utility of the

SCF helps in automating the process of integration testing. Using this utility of the federation takes away the burden of manually running test for every dependent package. Once the user clicks run test option for one package. The SCF creates the reverse dependency graph for that package and test all the packages dependent on that package.

- Deployment: Once the product is ready SCF helps in the planning of deployment of the software. In SCF we have the facility of builder server which acts like a pseudo production environment. This build server is somewhat similar to the server in which the application will be deployed. So SCF keep logs of all the issues faced in building software in this server and these logs are used to plan the production implementation to by bypass the unseen issues faced during deployment.

Uses, User and Graphical User Interface

Uses

The SCF provides the facility to develop a stable product for customer in an efficient way. To make this task of development efficient, SCF provides various tools which are used by different users of federation. Some of the major uses of the SCF are:

- Used to do automated integration testing
- Used to upload and download code files in repository
- Used to monitor work progress on a task using Task Mgmt. tool
- Used to share error or issues amongst the team using Issue Tracker tool
- Used to as discussion platform during designing, development and testing
- Used to prepare production planning document using collaboration tool
- Used to monitor to monitor the status of the project
- Used in Requirement gather phase with Customer.

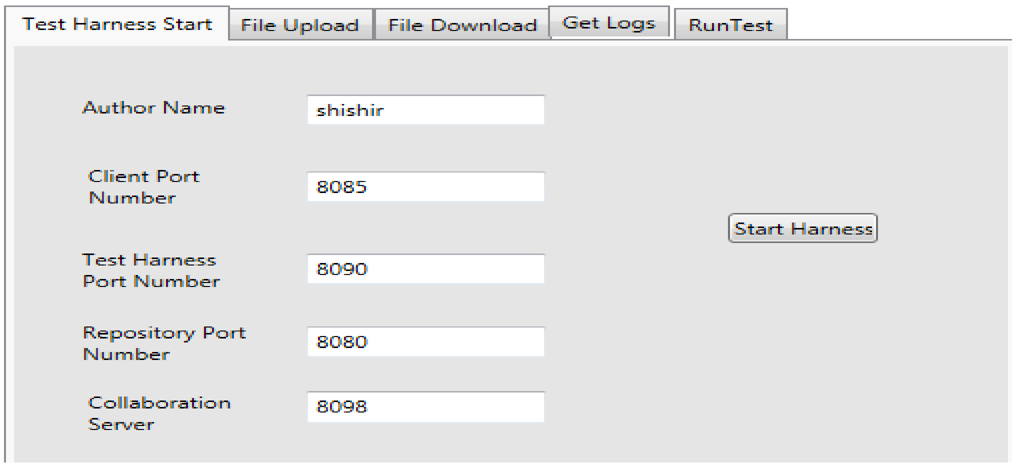
Above we have highlighted only the major uses. We will talk about them more when we will talk about each part in SCF individually.

Graphical User Interfaces

In this section we are going to talk about all the interfaces provide by the SCF. Please consider each interface as a tab in the GUI. After this we are going to talk about the users and refer these interfaces in those discussions.

User Interface to start Client

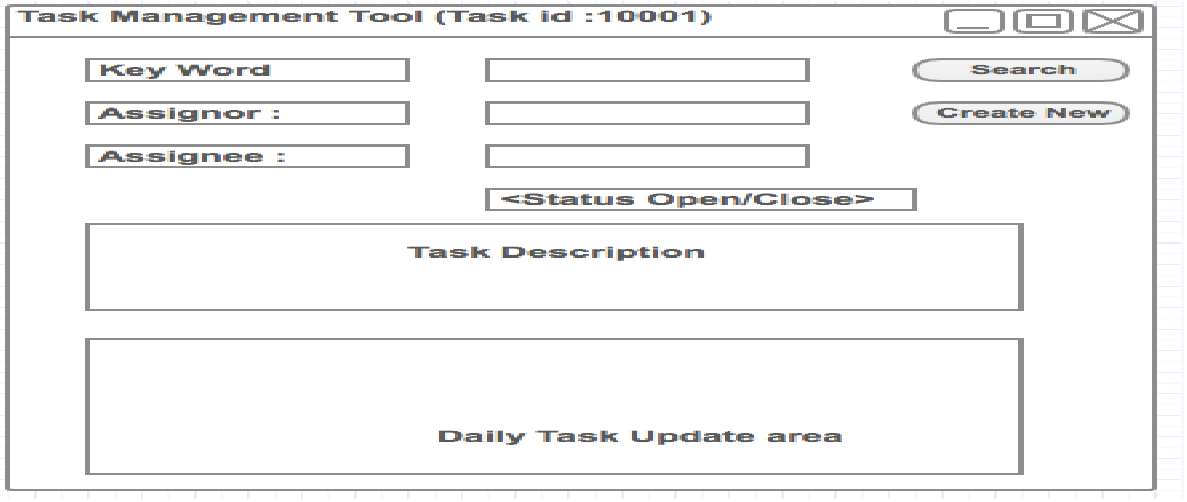
This interface has been designed, so that user can't do any other task before he provides the port number for himself and the Repository, Collaboration server and the Test Harness he wants to connect. This is because we can have more than one test harness and repository. Till the user clicks provide ports number and click on start harness (now it will be start Federation) all the buttons in another tab will be deactivated.



Author Name	shishir
Client Port Number	8085
Test Harness Port Number	8090
Repository Port Number	8080
Collaboration Server	8098

Task Management Tool

This tool is provided by the collaboration server. The purpose of this tool is to maintain a record of work progress of each task. This data gives better visibility to the work done by individual resource and can be used predicting the velocity of individuals.



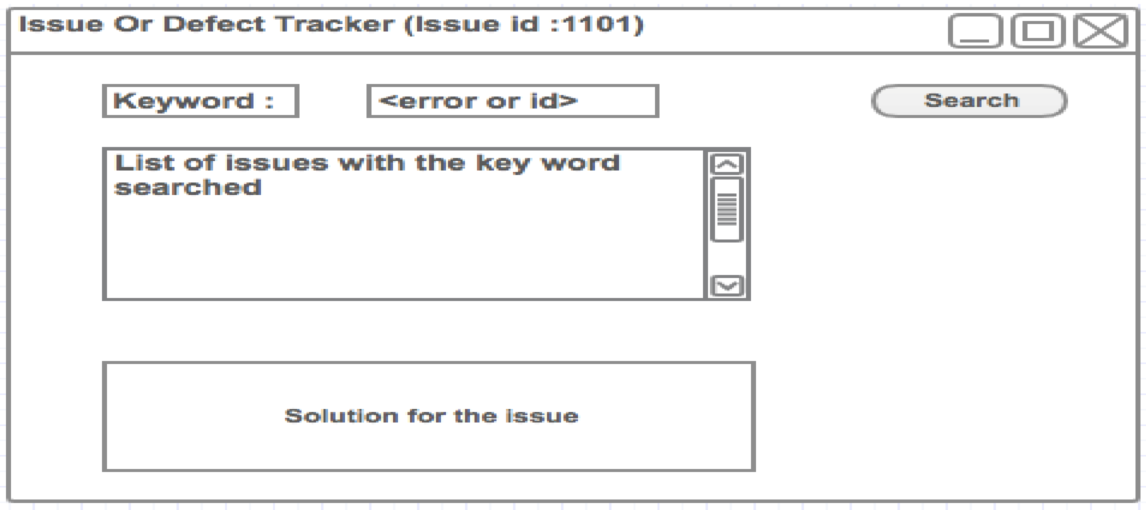
The Task Management Tools hold following information's:

- Name of user who created the Task
- Name of person to whom task is assigned.
- Description of the task to be done.
- Attached with required documents and approvals.
- Daily updates by developer till task is closed.

Issue Tracker

Issue tracker is a tool that will help in knowledge sharing between teams and follow up. Suppose a developer got an issue which coding and spends two days in solving it. If tomorrow some other member of the team gets same issue, and he doesn't know the first developer. He will end up wasting one or two days on same issue.

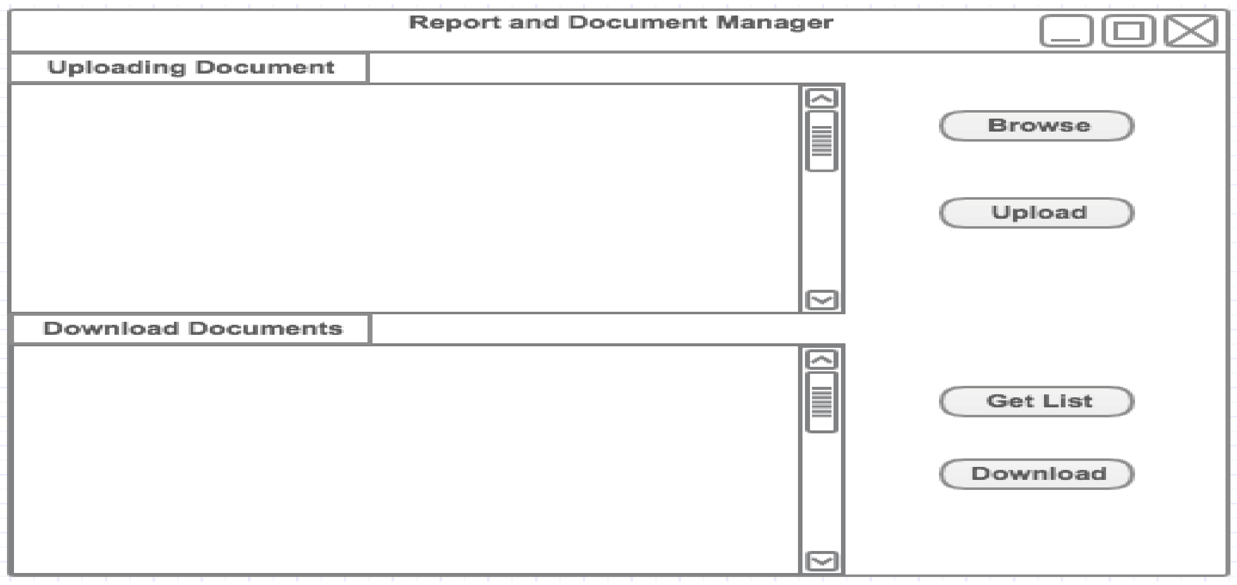
The Issue tracker is a place where developer can update the issue that they faced and the solution for it. Now any other developer can search for the issue and its solution without wasting same amount of time.


Issue Or Defect Tracker (Issue id :1101)
Keyword :
<error or id>
Search
List of issues with the key word searched
Solution for the issue

In this tool the user will be able to search issue with the help of a key word or an id.

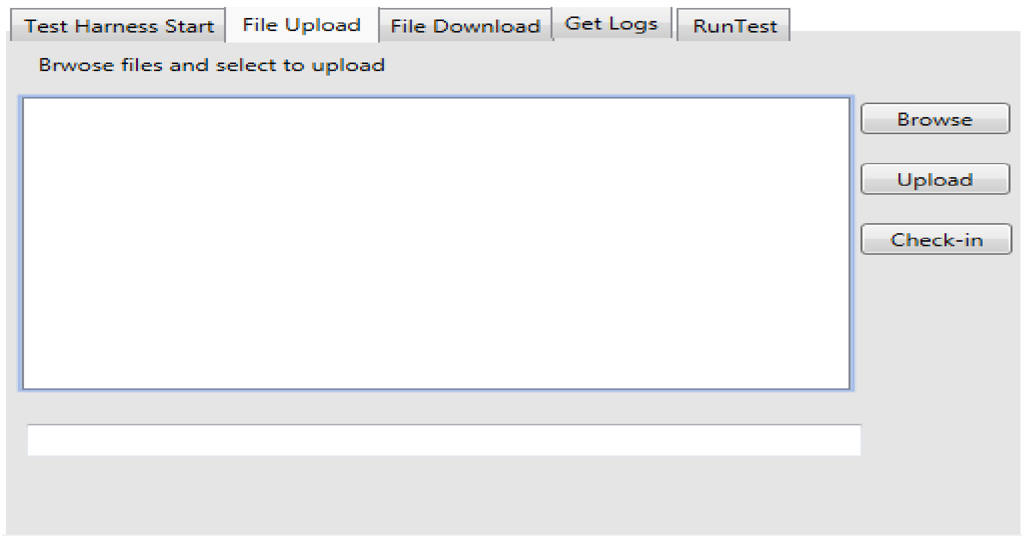
Document Management Tool

This is another tool provided by the Collaboration server. This tool can be used to upload as well as download project related document from the collaboration server. Some documents for example can be Design document, Requirement Document or OCD.

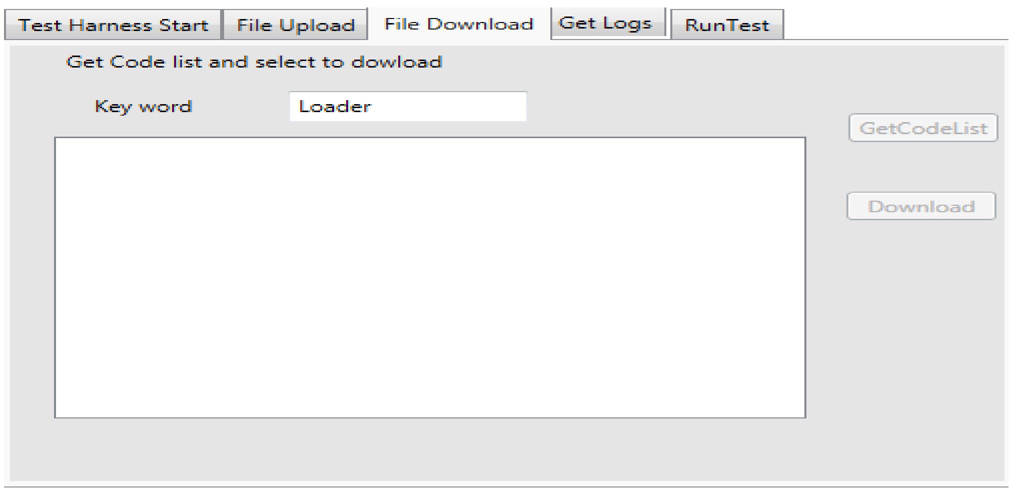


File upload and Download Interface

This Interface provides the facility to the developer to upload his code to the repository as well as to download files from the repository. Once the upload has completed the user has the option to do check-in close by clicking on check-in button. Till then check-in will be open.

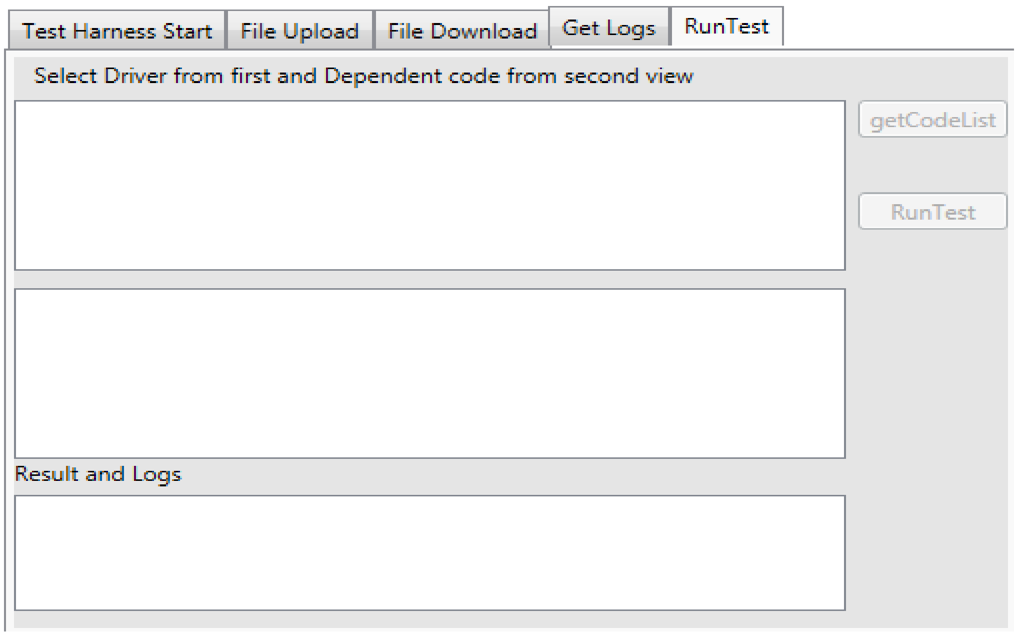


During download the developer ask for the list of files in repository and then select a file amongst that list to be downloaded. To selection simple we have provided an option to search based on key word



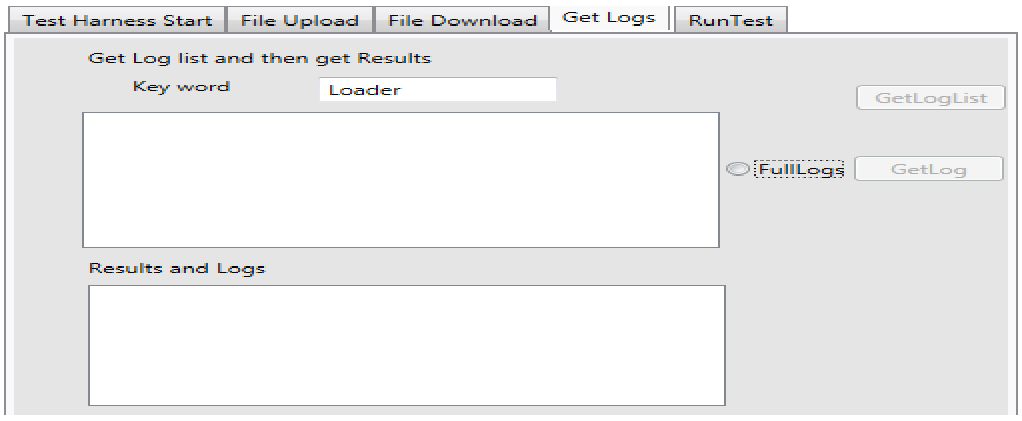
Running Test

In order to run the test using test harness, user needs to provide the name of the driver and the code files required. The UI below provides the user a list to select these files and run the test. Once the test has been run by the Test Harness the results will be displayed in the result window.



Log Request UI

This user interface has been provided for the developer, tester and manager to retrieve logs of old test. The user can get the list of logs present in the repository using get log list button. To get a short list we have given the option to get list based on a key word. Then user can select a particular log to retrieve it from repository. We have also given the option to just print the result passed/failed or get the complete logs.

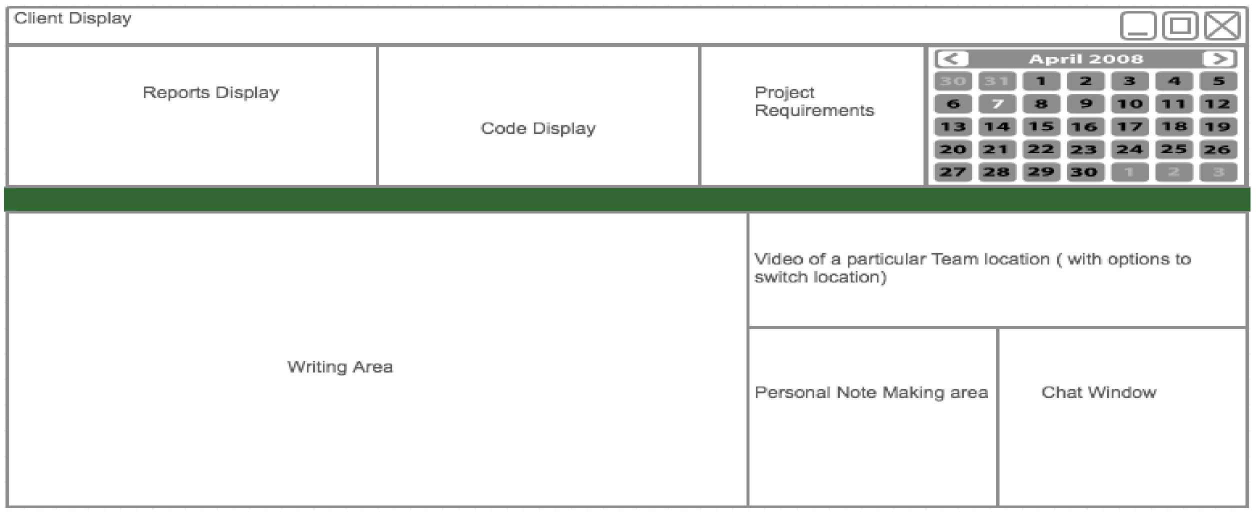


Virtual Display Client (Part of Virtual Display System)

The SCF provides the facility for teams sitting at different location to come together for software development. The virtual display is a touch responsive window, where each block could be magnified by clicking on it. This display has following blocks

- Report Display: This section is to display documents and reports in form of graph. This block can be used to display the graph for project development.
- Code Display: This area is used to display code from repository.
- Project Requirement: This section displays the requirement stated by the customer for the project.
- Task Manager: This section is displayed by a calendar. The user can click on a date and set the task for that day.
- Writing area: This area has been designed especially for the designing and task setting. This section of the window can we used for free hand writing and same changes will be reflected of all other user screen.
- Personal note making area: This area has been designed for note making and discussion which team at one location wants to keep to themselves.
- Chat Window: The chatting option allows user to talk between them in person.
- Video section: This section is for video conferencing.

The Virtual display system will have active voice from all location to discuss amongst them. When a user clicks on any of the block a pop up opens which allow him to do stuff related to that block. Example when user clicks on code display block he will get option to search for another code file and request it from the repository.


Client Display
Reports Display
Code Display
Project Requirements
April 2008
Writing Area
Video of a particular Team location (with options to switch location)
Personal Note Making area
Chat Window

Users

In this section we are going to talk about different users in brief. We are going to discuss them in detail along with the impact they made on the design latter in individual section of SCF sub systems. Below are various users of the SCF:

1) Developer: A developer is the person who writes the code for an application. He plays a vital role in software development life cycle. A developer is going to use the SCF for following purposes:
   - To store his code in Repository
   - To run integration Test on Test Harness
   - To get the latest document example design document
   - To share his knowledge regarding errors and defects with other developers
   - To update his daily work in the Task Manager
2) Quality Assurance Engineer: The role of a Quality Assurance (Q.A.) engineer is to monitor each phase of software development to ensure that the software adheres to the standards set by the client. A Q.A engineer uses the SCF for following purpose:
   - Run test using Test harness to make sure all requirements are being fulfilled.
   - To get latest document like the acceptance test case document.

- To update the developer on the Task management tool about the issues in the developed package.
- Use VDS (Virtual display) to discuss the acceptance test cases with the client or onsite present at client location

3) Manager: The main responsibility of a project manager is:

- **Manage resources:** The SCF helps the project manager in managing his resources. He can have a better visualization by seeing the reports that which component of the project has issues and will need extra resource. As well as helps in keeping a note of the resources doing a great job by check the task updates on the task management tool. Acknowledgement of good resources is a very important part of being a manager.
- **Handling Client**: The toughest job for the manager is to pacify the needs of a client. In general, a client is a person without any technical knowledge about software development. They are more concerned about the outcome rather than the internal working. In this case again a test harness helps the manager. With the help of a report the project manager can explain the status of the project in terms of facts and figures to the client.

4) Tech Architect

Tech Architects are the people in the project who design the software architecture. They use the SCF for following purpose:

- The Tech Architect use the Virtual display system to discuss and design the software architecture amongst them and include the leads sometimes. They use the common writing area to design the software architecture.
- They use the Document management tool of the collaboration server to keep every higher authority updated with the latest design document.

5) Business Analysist or Subject Matter expert

As most of the time the customer for which we are developing the software will be in a different country or city. So earlier B.A. or S.M.E used to travel to their work location and used to prepare the requirement document. But now with the facility provided by the VDS they can sit in their office and can connect to customer to discuss project requirements. They will also use the Document management tool to upload the latest requirement document into the collaboration server.

6) Social Tracker

Social tracker is a person in agile development whose job is to distribute the task amongst the developer. We don’t give the choice to the developer to select his/her task as in that case we might face situation when no one take up a task as it was difficult.

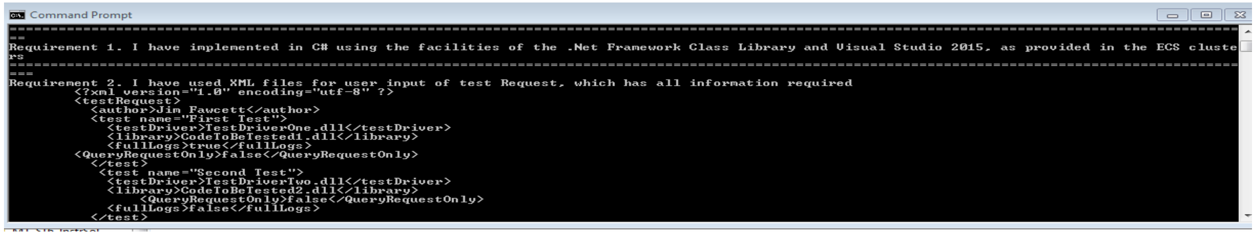
The Task management tool is used by the social tracker to assign task to the developer and monitor their work progress. The tool helps them to know about the individual capabilities of the developer and decide the timelines for next task.

7) Customer

In development methodologies like Agile the one representative from the customer side needs to be involved constantly in the development cycle. To make the life easier for the customer to travel to the development location at the end of every development cycle, VDS comes into picture. With the help of Virtual display system utility of the SCF client can attend the meeting at the end of each development cycle by sitting in his office.
It also helps in requirement gathering phase and acceptance testing phase.

8) Instructor/T.A.
The main purpose of the T.A. /Instructor for using this application is to evaluate its functionality against the requirements stated in project 5.

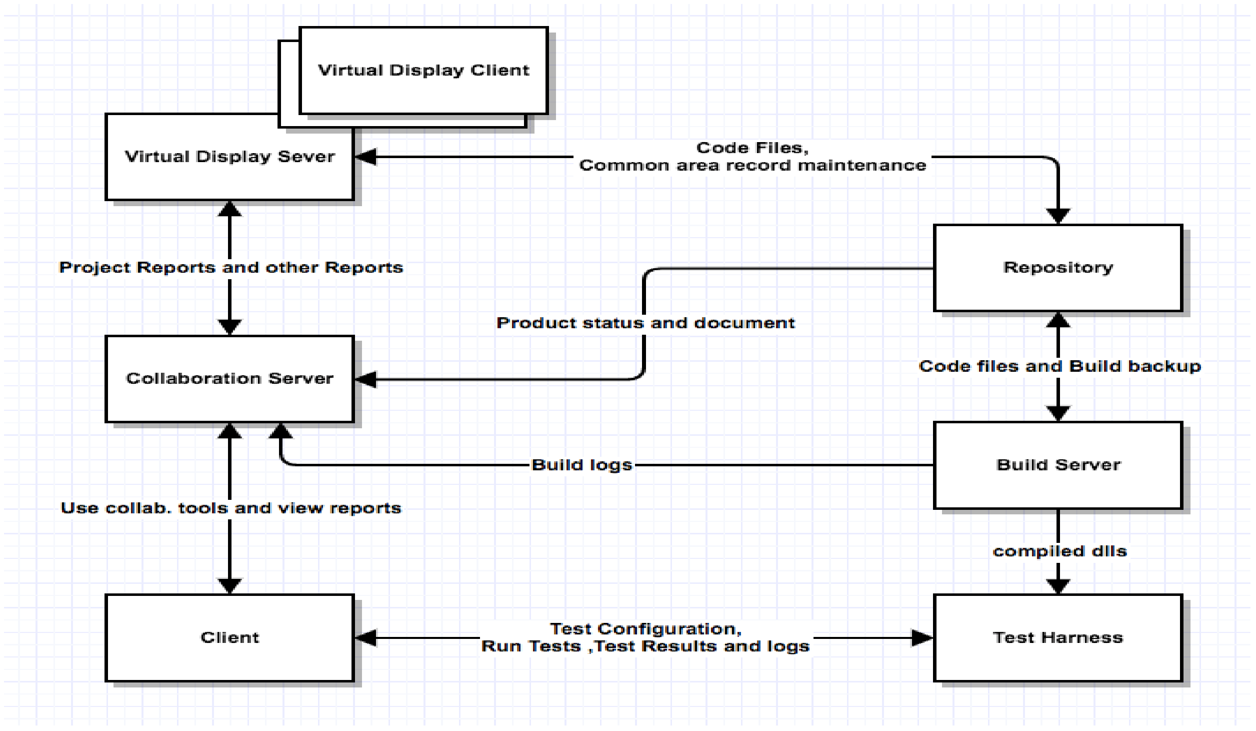
We will create two files run.bat and compile.bat for them, which will help them to compile and run the application. It will require minimum efforts from the side of the user. Once the application starts it will start printing the results on the console in the same order as given on the project requirement page. The results in console will look file below:



```
==================================================================================================================================
==
Requirement 1. I have implemented in C# using the facilities of the .Net Framework Class Library and Visual Studio 2015, as provided in the ECS cluste
rs
==================================================================================================================================
===
Requirement 2. I have used XML files for user input of test Request, which has all information required
        <?xml version="1.0" encoding="utf-8" ?>
        <testRequest>
          <author>Jim Fawcett</author>
          <test name="First Test">
            <testDriver>TestDriverOne.dll</testDriver>
            <library>CodeToBeTested1.dll</library>
            <fullLogs>true</fullLogs>
        <QueryRequestOnly>false</QueryRequestOnly>
          </test>
           <test name="Second Test">
            <testDriver>TestDriverTwo.dll</testDriver>
            <library>CodeToBeTested2.dll</library>
                <QueryRequestOnly>false</QueryRequestOnly>
            <fullLogs>false</fullLogs>
          </test>
```

Structure and Partitioning

We have divided the responsibility of the SCF between 5 different serves. Each of these server have their own set of responsibilities. The interaction between these units can be explained by the block diagram.


Virtual Display Client
Virtual Display Sever
Code Files,
Common area record maintenance
Repository
Project Reports and other Reports
Product status and document
Collaboration Server
Code files and Build backup
Build Server
Build logs
Use collab. tools and view reports
compiled dlls
Client
Test Configuration,
Run Tests ,Test Results and logs
Test Harness

Client

Client system of the SCF has been designed to use the web services of the Test Harness, Repository and the collaboration server. It provides a graphical user interface to the user to interact with other servers. It interacts with other servers for following task:

- Repository: The developer uses the repository for storing his source code. The GUI provides the option to upload and download a file from Repository. It also uses repository to retrieve old logs.
- Test Harness: Once the code files have been checked in the Repository. The user uses the Run Test utility of the client to test do the integration testing of the code which takes help from the Test harness server to the task. Once the test has run, the Test harness returns back the result to the client.
- Collaboration: The client uses the collaboration server for software development management. Tools such as Task tracker and Issue Tracker helps in knowledge sharing and monitoring the flow of development.

Test Harness

The Test Harness server provides the system for automated integration testing. The Test harness uses the Test Drivers to conduct the task of testing. The test harness has been configured in a way that it can be used to run test if a driver follows certain criteria. We can't use take up any code and use it as Test Driver. A Test Driver must inherit from ITest interface and should have a test() function, in order to be used for testing. The Test Harness interacts with other servers to do the job of testing:

- Client: The test harness works on Test Request and the client needs to provide information about the test to run in that Test Request. The client sends a test request message with test information to the test harness, which uses it to run test.
- Build Server: The Build server contains the current baseline image. The test harness requests the build server for the files required and load them in assembly to run test.
- Repository: Once the test has been executed the results and logs are sent to the client and the Repository. The reason for sending them to the repository is for storing them for future use and report preparation.

Repository

As any other system, in SCF the repository is the storage house of the whole system. In the current system, it has been used to store code, logs and documents. Other system in SCF interacts with the Repository in following ways:

- Client: The client uses Repository for storing source code and to retrieve old logs.

- Build server: gets the information of the file checked in by the user and uses it to create the build of the application being developed.

- Collaboration server: The collaboration server uses the information provided by the repository for preparing project status report. It uses the test run logs stored in repository to know the packages which are up and running as well as failing.

Build Server

Build server helps in avoiding situations where the developer says “The code is running on my system. I don’t know why it is failing at client location”. The build server provides a production like environment in which the code can be compiled. In order to compile the code and provide information for production implementation planning the build server interacts with following servers:

- Repository: Whenever a user check-in a file the repository informs the build server about it and the build server request the file from repository to create a new build of the application being developed.
- Test Harness: The test harness requests the compiled code files for running the test drivers from the build server.
- Collaboration server: The collaboration server needs build server logs in order to prepare a document, which would help in having an issue free production deployment.

Collaboration Server

The purpose of adding the collaboration server to the SCF is to provide the project management facility to the SCF. It provides tools for keeping the tack of task which the managers and social tracker can use to monitor the work progress of project. It also provided utility for knowledge sharing which increases the overall productivity of the system. It interacts with following machines:

- Client: The client uses the collaboration server for various task:
    1. Task Management Tool: The client (Manager/Developer) uses this tool for maintain a record of work progress on a particular task.
    2. Issue Tracker: The client uses the issue tracker tool for knowledge sharing of a particular issue he/she faced with other team members. It helps the other team members to solve issues faster and increase productivity of team.
    3. Document Management Tool: The whole process of software development required lots of documentation like requirement document, OCD etc. This tool helps user in uploading the latest version of each of these document, which will be available for each member of team. Hence no one will be working on an old version.
- Build Server: Build server provides its logs to the Collaboration server which uses it for preparing the implementation help document.
- Repository: The collaboration server uses the information provided by the repository for preparing project status report. It uses the test run logs stored in repository to know the packages which are up and running as well as failing.

- Virtual Display server: uses the collaboration server for getting reports such as project status reports or any other document such as design document.

Virtual Display Client

This part of SCF provides a discussion platform for team present at different location. It provides a walls size display at each location which is touch responsive. The whole screen has various sections to display code, reports, provide chatting and video conferencing facility. It also provides an area where user can write or design stuff. It interacts with the following systems:

- Virtual Display Server(VDS): The virtual display client uses the VDS to interact with the rest of the SCF system. It uses the facility of chatting, video conferencing, report manager and common area data management to display useful content on screen

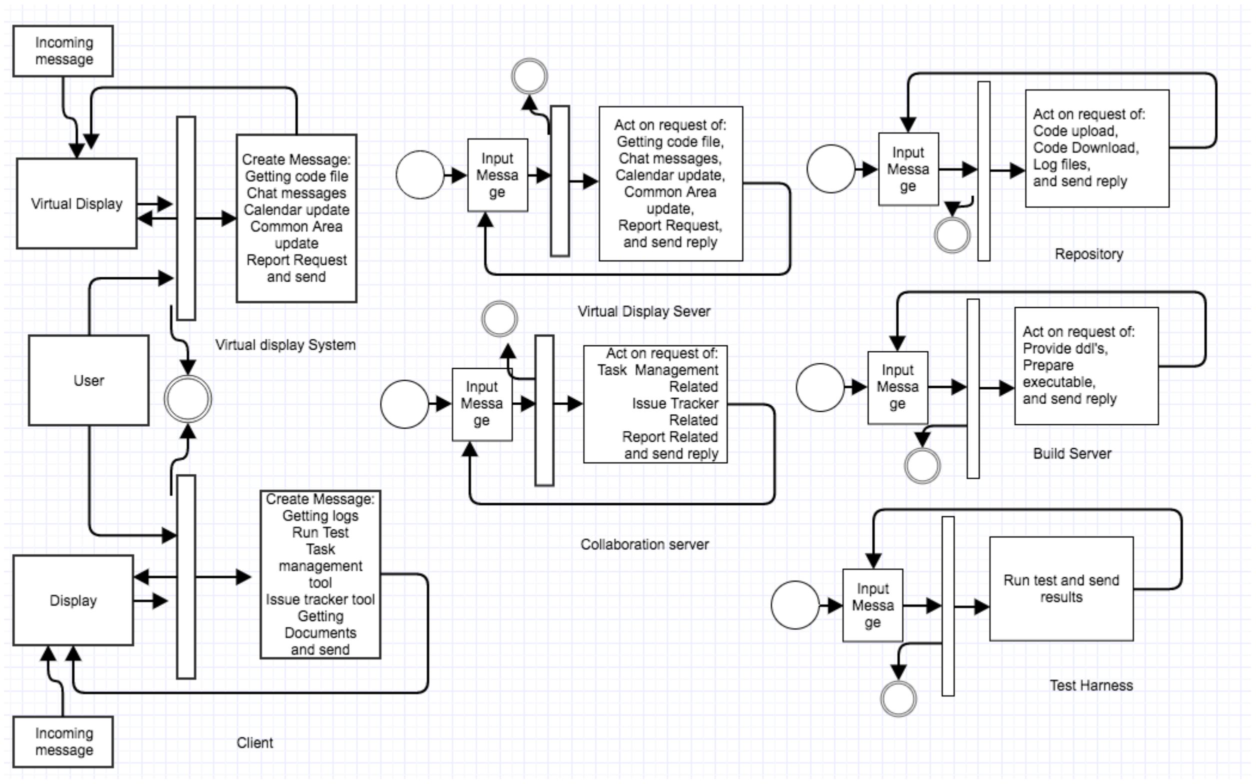
Virtual Display server

The reason of having a VDS outside the collaboration server is to provide common content to be streamed at different location with least possible delay. All the content that the user see on the Virtual display client is actually present at the VDS. The reason for keeping it at one place is to common data for all the clients. It interacts with following systems:

- Repository: The VDS uses the repository to get the code files, which needs to be displayed on the virtual client.
- Collaboration Server: The VDS uses the collaboration server to get all the documents required example the project status report, task report etc.

Key Application Activities

The SCF provides various functionality to the user which help them in software development. In order to complete these task various sub systems of the SCF interact with each other. We will discuss the activities of each sub system in detail latter. For now, let us take an overview of the activities done by these sub systems


Incoming message
Virtual Display
Create Message: Getting code file Chat messages Calendar update Common Area update Report Request and send
Virtual display System
User
Input Message
Act on request of: Getting code file, Chat messages, Calendar update, Common Area update, Report Request, and send reply
Virtual Display Sever
Input Message
Act on request of: Task Management Related Issue Tracker Related Report Related and send reply
Collaboration server
Input Message
Act on request of: Code upload, Code Download, Log files, and send reply
Repository
Input Message
Act on request of: Provide ddl's, Prepare executable, and send reply
Build Server
Input Message
Run test and send results
Test Harness
Display
Create Message: Getting logs Run Test Task management tool Issue tracker tool Getting Documents and send
Incoming message
Client

Client Activities

The Client sub system of the SCF is to provide a graphical user interface to the user to use the services provided by Repository, Test Harness and Collaboration server. Once the client has started the system by entering the port numbers for each server. He could click on various tabs to get the below task done

- Use Task Mgmt. Tool and Issue Mgmt. Tool of collaboration server.
- Use Repository utilities like file upload and download.
- Run integration testing using Test Harness.

Once the request for one of these task is sent the client will go into wait state for either another user input or till an incoming message comes to the client. Once an incoming message comes the client system extract information from it and display it on the screen.

Virtual Display System Activities

The virtual Display System provides a touch sensitive display to the user. Once the user provide input on the screen by either touching various options the background code does one of the following

- Does chatting and video task
- Does Common area related task
- Does document, code and report related task

Once done it goes in wait state till the system receives an incoming message. Once the message is received it updates the display based on the type of message.

Repository Activities

Once the repository system receives an incoming message from other sub system. It de-queues it and act based on the type of message. The type of messages the repository can act are:

- File upload and download
- Result and logs saving as well as retrieval

Once the task has been completed it will send a reply message to the client.

Test Harness Activities

Once the test harness system has started. It will wait for messages from the client for integration testing. Once the message is received by test harness it will send the request to the repository to give required file to test harness. After the files have been transferred to the test harness, the test harness will run test on those files with the help of test driver and sent the results to the client and repository.

Build server Activities

The build server once started waits for an incoming message from the repository to create the dll files of the newly checked in files. The build server will request those files from the repository and compile them to generate dll files. It will inform the client about the new build status and will send the build logs to the collaboration server. Once the executable images of the files have been created the build server can provide them to the Test harness when requested.

Collaboration server Activities

Collaboration server will wait for the incoming messages from the repository, build server and the client. Based on the type of incoming message collaboration server can do the following

- From client: It can generate report, provide status of a task, provide information about an issue.
- From Builder: Use the information to generate production helper document.
- From Repository: Use the data provided to make entry for project status report.

In case of message from repository and build server, the collaboration server does not send any reply. In case of message from client the collaboration server will do some task and then send a reply.

Virtual Display sever Activities

The virtual display server once started will wait for an incoming message from the Virtual display client. Once the message has been received it take care of 2 types of request i.e. chat & video related and common area related. In case of file related request, it sends message to the repository or the collaboration server for the file. Once the request has been completed the server will send a reply to all the active client in the current session.

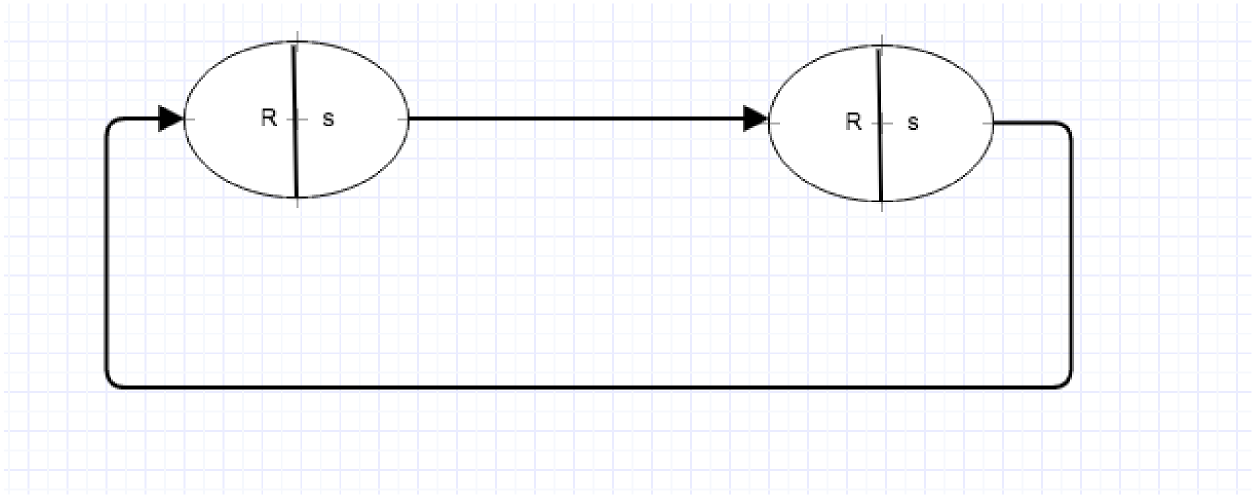
Communication

We are using asynchronous mode of communication for data transfer between two machines in SCF. Each system in SCF has a sender and a receiver port. Whenever some message is put in sender block queue of a machine, its sender sends it to receiver of another machine over the WCF channel. It does not wait for the response of the another machine. In our application we are transferring data in two ways:

- Messages: We are sending messages over WCF channel which carry following information
    1. To: URL of Receiver
    2. From: URL of Sender
    3. Author: Name of User
    4. Type: tells about the type of Message
    5. Timestamp: For holding date time

6. XML Body: To hold operation oriented data.

- File Transfer: We are using this type of data transfer for files. We read file in chunks of data (1024 or less) and send it over the communication channel where file write operation takes place using these chunk of data. For this mode of data transfer we pass below information in advance before transferring the data:
    1. File name
    2. Size of the file in bytes



As we are using WCF for communication between various system present in the SCF. We need a service contract, which needs to be implemented by a service class in the receiver package of every system. We call the service contract interface as Iservice interface. Any class that implement this interface has to write concrete implementation of these function:

1. SendMessage(Message)
2. GetMessage()
3. OpenFile(string)
4. writeBlocks(Bytes)
5. closeFile()
6. SendVideoStream()

Any system which has a service class which implements these functions can be receive data from other system by its proxy object.

Critical Issues

1) **Process flow management**: If a user tries to run the test on the test harness before sending the files to the repository. As the sequence of steps is not proper user won't get a desired result. If the user is not clear about the flow of the system, he might never be able to use it to its potential

   Solution: This can be achieved by asking for as much less input from the user as possible for doing a task. For example, instead of manually asking the user to type the name code files to run the test, we provide them a list of files they can use to run the test. This will act as a reminder to the user that he has not checked-in a file.

2) **Ease of use**: All the utilities of an application are useless if they are difficult to be used as the user will not prefer using it or will make many mistakes using it.

   Solution: Provide a Graphical user interface which require as less input from the user as possible. As well give them the option of selecting rather manually entering the details.

3) **Client sending Message before starting of server**:

   As the client, Test Harness and the Repository are running on a separate machine. A situation can come that the client has started and servers are still booting. In such a situation if the client sends a message an exception will be thrown as the Host channel is still no created.

   Solution: To solve this issue we must do two things before start sending messages. We can have two buttons one to start listening and other to connect to the server. First we should start listening at a particular port then connect button should get activated. After that we should click on connect. Till the local machine is not able to connect our message sending button should be deactivated. Once both the things are done then only we will be able to start sending message.

4) **Graceful shutdown of threads:** As we know that all our servers complete the task and then wait for an incoming message. If we don't send any message they are in wait state. At this time if we close the terminal or console we won't be killing the thread properly

   Solution: To do this task gracefully we will kill a thread when it gets a quit message. Once the thread gets to know it has quit message, it will put that message back into the queue and then we will kill it. The reason we are putting back the message back into the queue is if we have multiple thread dequeening from the same queue all of them can get that quit message.

5) **Communication:** We have a very big system in SCF. How do we do communication in such a large system. One sub system might try to communicate with other sub system in a way which it doesn't understand.

   Solution: To bring uniformity in the communication system we have used a generic system for communication. We have used message class object for any communication happening in SCF. The user has the option to set type variable of the message class to pass different kinds of instruction from a generic class.

Possible Extensions

In future we can plan to add on following features to our SCF:

- Production Deployment facility: If we could add the facility in the SCF that it could deploy the code directly into the production environment without any manual interventions, then we will be able to eliminate many human errors made by us during deployment.
- Storing metadata of repository in relation database: This will make the retrieval of the dependency list much faster in comparison of reading from metadata files.
- If we could make the test harness smart enough that whenever the user checks-in his code in the repository the test harness by itself run the required test. This feature has will have its own pros and cons.

Repository

Repository is the location in the SCF which can be used to store any form of data required in the software development life cycle. As the part of project we will be using our repository to store code files and test results. The repository along with the responsibility of storing data does the following:

- Manage Package dependency
- Control current baseline
- Disclose the state of current baseline
- Explore baseline

Concept

Let us look into how the repository system is taking care of each of these responsibility:

1) **Manage package dependency**: To manage the dependency, the Repository creates a metadata file for each package when checked-in close happens. So whenever the user sends a file to Repository. It will go through two stages:
   a) **Check-in open**: When the user will send the package to the repository, the repository will check from a table the last version for that package and create a folder with version name attached to it. Example LogManager_V1. At this time, it will also create a metadata file for the package. The metadata file will look like below:

```
 <PackageDependency>

 <PackageName>Test_Harness_Executive<PackageName>
 <author>shishir</author>
 <timestamp>12-11-2016</timestamp>
<FileName>AppDomainManager.cs<FileName>
<FileName>Test_Harness_Executive.cs</FileName>
<DependentPackage>Loader</DependentPackage>
<check-in>open</check-in>
< /PackageDependency>
```

   A point to note in this that the tag check-in tells us if this package has been frozen or not. If it is not frozen any new update user make in the files of this package, no new version will be created and changes will be reflected in this folder only.

   b) **Check-in closed**: This is the time when the user knows his changes are final and changes done till now should be a new version. At this stage the check-in tag is changed closed and this version get frozen. From that point when it is changed from open to close any change will form a new version.

   Now let's look in what happens when the test harness asks for particular package. The Repository will search for each package that have dependent package as the package requested and pass them also to the test harness for testing (it will search for dependency recursively). This will help Test harness to do integration testing.

   The other good utility this add to our system is that now when the user request for a package from repository, he will get all the dependent packages along with that package.

2) **Test Results and log Files**

   The repository also helps the test harness to store the old test results and logs into it. Once the Test harness has done its testing it will send the log files and the results to the repository. The repository will use the author name, drivers name and the timestamp as the key combination to save the files.

The reason for using the above combination for key is that now the user will have the option to search for the logs based on author name as well as the driver name. A sample test result xml file with logs will look like below:

3) **Disclose the state of current baseline**

   This feature has been developed for generation of project status report by the collaboration server. Whenever a package which will be the part of current baseline is tested by the test harness it results are sent back to the repository. Repository send the update to of these data results to the collaboration server. So now the collaboration server can prepare a report of which are packages in the current baseline are up and running. Apart from that the repository will also inform about the updates made in packages and the author who made those changes to the collaboration server.

4) **Control current baseline**

   The reason repository needs to have control over the current baseline is because every package will have various version and how can the repository know which one is the part of current baseline. To make things easier at repository side we make one more change in the part when we save the file in version folder of package.

   Suppose we have a package Loader_V1 and it has four files. Now we make change in only one of the file out four. Now if the user uploads this package again, repository will form a package Loader_V2 and it will have only one file (the updated one). In the metadata file of the Loader_V2 it will be mentioned that the file location of other 3 files is still Loader_V1.

   Hence now if the repository takes the latest version folder of each package it will get the latest files for that package by reading its metadata.

Uses, User and User Interface

Uses

Repository acts as the storage house for the SCF. It provides storage for:

- Code files
- Test Results and logs

The user has the access to upload and download code files from the repository. User can also download test results and logs from repository. Apart from this in our case repository also plays an important role in preparing project status report. Whenever a new package is checked in or test results of a package change, the repository informs the collaboration server about it.

Console Interface

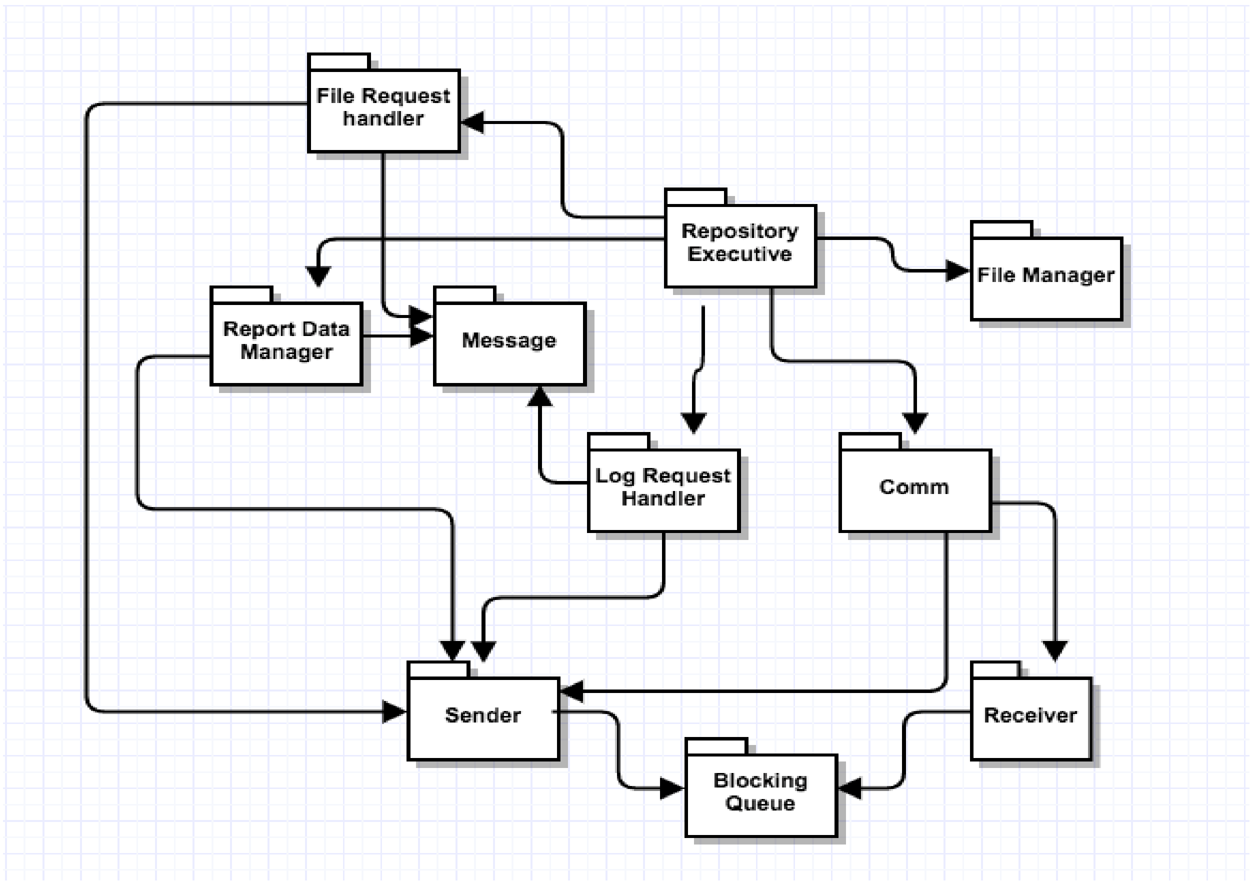
Every repository needs a port number with it can be used as a URL for sending messages and files to it. For this we have provided a console interface in which the user can pass the port number as the input argument while starting the repository

Users

We have discussed about how the users will use the Repository in the federation. Here we will discuss about the impact on design they made in test harness.

1) Developer’s impact on design
   Whenever user needs to do some enhancement in his code file. He needs all the files on which that code file depends to successful compile it on his system and do unit testing. The Repository makes this task easy for the developer by maintaining a metadata file for every package. So now if a user downloads a package then all dependent packages will also come together with that if the user want.
2) Q.A and Project Manager impact on design
   The repository gives the option to both of them to see old results and logs. It also provides the system where the user can request only the test results and not the logs.

STRUCTURE AND PARTITIONING



Repository Executive

This is the main package of the Repository system. The Task of the repository package is to control all other packages in the system to get the job done. We can divide the functionality of the Repository package into two parts for better understanding:

- For Incoming Messages: The Repository takes help of the receiver package for creation of repository channel, which the client and the Test harness can use to send messages or file to it. Once the channel is open service class of Receiver package helps client and test harness to put incoming messages into blocking queue (ReceiveQueue) of the Receiver. Then Repository de-queues the message and based on the type of the message the Repository passes control to Log Request handler or File Request Handler package for taking care of the request.

- For Outgoing Messages: Once the Log Request handler package has done the task of getting information from a log file using the FileManager package. Using the data provided by Log Request handler package creates object message class and put it in sender blocking queue of the sender package. Once the message is in the queue, the sender package will send the message to the target server using proxy object.

File Request Handler

This package helps the Repository to take care of the request for check in or sending file to the client or the Test Harness. For file sending request it uses the FileManager package to locate the code file in the repository with the help of the metadata files present in each package and then use this location in download function of the Repository service class present in receiver package. In case it is a checking request, it will save the whole package in the repository and generate the metadata for that package. During the process of check-in, it will follow the concept of check-in open and complete discussed in concept section.

Log Request Handler

This package helps the repository package in handling the log request from the client and test harness. The request can be of two types:

- Log file creation
- Old log file retrieval

It uses the FileManager package in order to get the location of the log file in case of log file retrieval request. Based on the type of request Log Request Handler uses XDocument facility to do the task of reading or writing into a xml file. Once the task is completed old log file data retrieval it creates a message class object and puts it in the sender blocking queue of the sender package.

Report Manager

The report manager package has been designed to keep the collaboration server up to date about the status of the project. The report manager package sends the update to the collaboration server in two cases

- Once the check-in has been closed the Report Manager will create a message about the new update and send it to collaboration. It will also create a metadata file at this time to keep a track of the status of this package.

```
<PackageStatus>
<PackageName>Loader</PackageName>
<VersionNumber>2</VersionNumber>
<Status>Not Tested</status>
</PackageStatus>
```

It will save the file with the package name only so that every package has only one status file.

- When the Test Results are received by the repository after completing the task of saving it. The Report manager will check the status of the package has changed or not. If the status of the package has changed it will inform collaboration server about it and update the package status metadata also. In case it has not changed it will do nothing.

Comm

The Comm package has been designed to remove the complexity from the executive package. The Comm package takes the responsibility to take care of the communication part of the system. It holds the sender and receiver of the system, so now the executive doesn't need to hold them. It also keeps the track of the proxy object generated and helps in not creating the same proxy again and again by maintain a record of proxy and URL in a dictionary.

Receiver

The receiver package acts as the listener for the Repository system. This package creates a channel for the Repository through which all other machines can send messages to it. This package has a receive message blocking queue which is used to store all incoming messages. It also provides a send message function which is used by other machines to put message into the receive queue. It also provides the facility to download a file in the repository system, this facility is used in the process of file check-in to repository.

Sender

This package provides the medium for repository to send request to other system. The sender package helps in sending messages to other machines as well as download file to other system. In order to send any request to other systems sender package create a proxy object of the machine to which the request has to be sent and then uses functions of other system service class to communicate. Whenever repository wants to send a message to other system, it puts a message in sender message queue of the sender. At the start of the repository one thread will be continuously running to send message from this sender queue.

Message

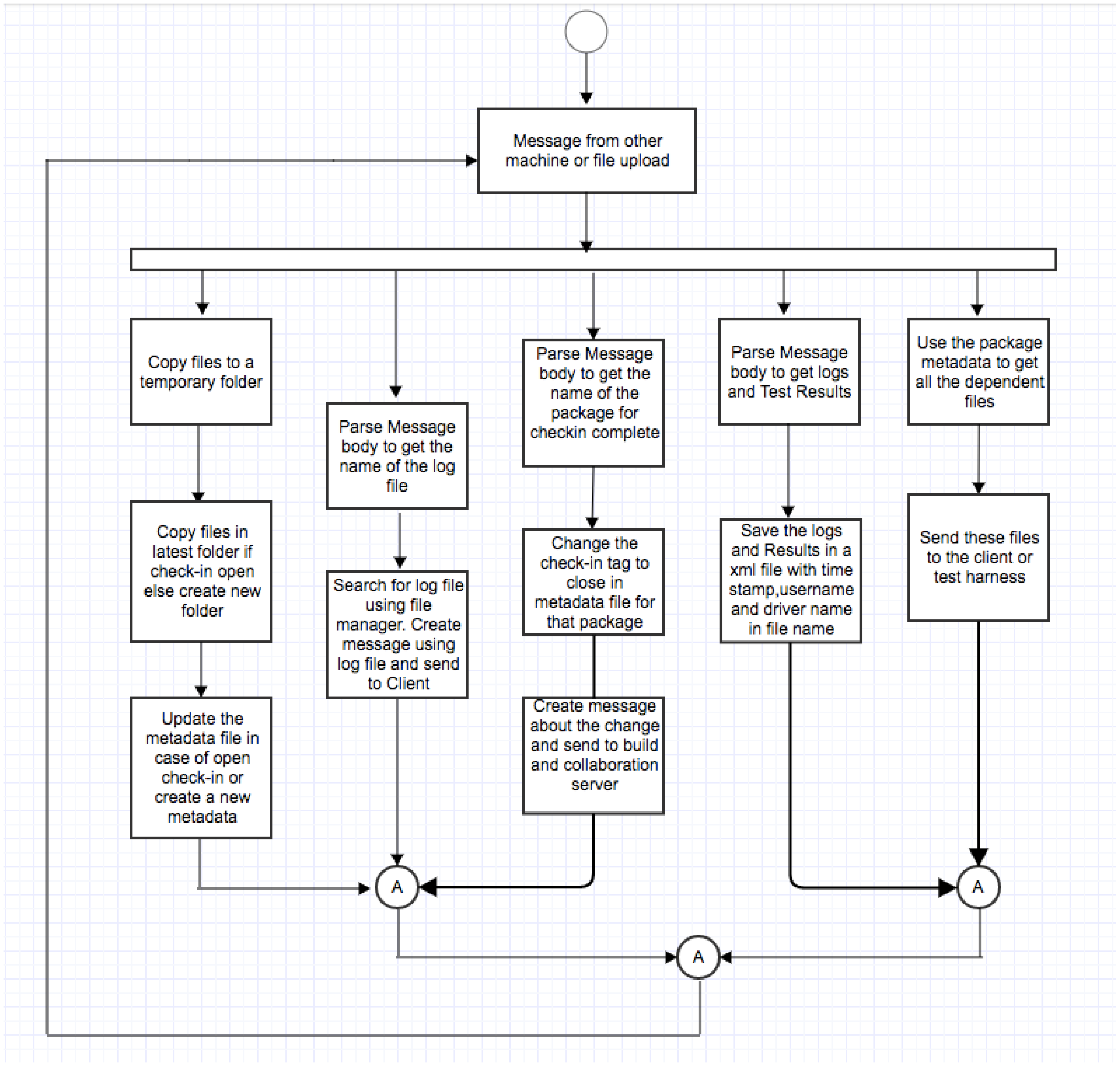
The Message package has been developed in order to provide a generic mode of communication between all the machines present in SCF. A message passes over WCF contains following information:

- To: URL of Receiver
- From: URL of Sender
- Author: Name of User

- Type: tells about the type of Message
- Timestamp: For holding date time
- XML Body: To hold operation oriented data.

KEY APPLICATION ACTIVITIES

The key activities of repository involve its interaction with the client, Build server, Collaboration server and the virtual display server. Below we have tried to explain the key activities that the Repository performs to complete the task assigned to it. We have also discussed these activities in detail.


Message from other machine or file upload
Copy files to a temporary folder
Copy files in latest folder if check-in open else create new folder
Update the metadata file in case of open check-in or create a new metadata
Parse Message body to get the name of the log file
Search for log file using file manager. Create message using log file and send to Client
Parse Message body to get the name of the package for checkin complete
Change the check-in tag to close in metadata file for that package
Create message about the change and send to build and collaboration server
Parse Message body to get logs and Test Results
Save the logs and Results in a xml file with time stamp,username and driver name in file name
Use the package metadata to get all the dependent files
Send these files to the client or test harness
A
A
A

File check-in (open)

The file check-in process happens in two parts. Let's understand them in the sequence of steps:

- When the user sends over the file to the repository using the file upload option of GUI. The files are copies to a temporary location in the repository.
- Once all files are in the temporary location the repository check is there any open check-in for this package. If so it will copy these files into that folder and update the metadata in case some change happened.
- If there is no open check-in for that package. The repository will create a new folder with a new version number in it and copy the files to it and generate the metadata files

  We will maintain a table which could be a dictionary, in which key will be package name and value will be its current version number.

File check-in (close)

When the Repository get the request to check in close for a package. It will go to the latest version of that package and check if its check-in is open. In case it is open it will close the check-in tag in the metadata file for that package:

<check-in>close</check-in>

Once the check-in has been closed the Repository will send a message to the build server about the check-in, so he can create a new build. It will also inform using message the collaboration server about it so that it can use that information in its status report.

Result and log file creation

Once the test harness has completed its test run, it sends the test result to the repository. The repository gives the control to log Manager package which uses XDocument facility to get the name of the test driver from the message body. Then it uses the author name from message, drivers name and the timestamp as key combination for the xml file name and save the body of the Message class in that file.

Once that is done it will check the old status of this package and if there is any change in status i.e. it was failing and now it is up and running. It will inform the collaboration server about the change in status.

File download

The file download process can happen in two ways:

- One when we give only the files requested by him
- The other could be given all the dependent files along with those files.

In case the Request comes from Test harness then we should send all the dependent files, but in case the request comes from the client. We give client the privilege to decide what he wants.

In case it is files without dependency we would send all the files requested to the user without checking the dependency. This type of request will generally come when the user wants a code file on VDS. As the purpose of using it on VDS will be discussion only and all dependent packages won't be required for building the package.

In case request is for files in that package along with the dependent packages. The Repository will find all the packages on which that package depends using the metadata file of that package and send it to the user. Please note that we will need to search for the dependency recursively.

For sending the files to the user we will use the sendFile facility of our sender package which will break each file into chunks of bytes and send it over the WCF channel.

Result and Logs Request

Again this type of request can be just a result request or result and log combine request.

- In case it is log and result request together. Then the job of log manager is easy. Just load the xml file into a XDocument and attach that to the message body and send the message to the requester. As our xml file contains results as well as the logs.
- In case it is just a result request the log manager loads the xml file in XDocument and then get the results from it. Once it has the results log manager creates a message class object and send it over to the client.

Critical Issues

1. **Multiple people check-in same file**: During development phase it can happen that two developer can try to check-in same code at a time. This will result into overwriting of one's code over other.
   Solution: We will follow single ownership policy where the person assigned to a code can only make changes to code file with exception to the team lead who will also have the write access over code file.
2. **When to ask for logs**: As we know that the test results are being saved in the Repository and Test is happening in test harness. So a situation can happen when the user has requested for test request and might ask logs from repository before the test has completed. So the repository won't have logs and the request will be wasted.
   Solution: We can solve this problem in a very simple way. Instead of giving user the option of manually typing the name of the log, we have an option of get log list. This function get log list will ask the Repository which all logs does it have and give the user the list. Now if the log has still not been made in the repository the user won't get in the list and can't ask for it. We can maintain a vector with log file names at Repository to give the log list to GUI.
3. **Heavy load issue**: Suppose we have only one repository for big team. Then this single repository will entertain the request for all users as well as the build servers. This might reduce the performance of the repository.
   Solution: In case of large team that too in different countries we can have one repository in each country. This will reduce the load on a repository by load sharing.
4. **Concurrent File access**: A situation might come when two users have asked for same file at same time. This will lead to request failure for one of the user as the file will already be in open state and other thread will try to open it and it will throw an exception.
   Solution: The solution for this problem is very simple. It a thread gets an exception while opening the file, make the thread sleep for some time and then try. If it fails again do the same 4-5 times. This will solve the problem of concurrent file access.

5. **Reverse dependency graph**: If we save files normally without any dependency information. Then we won't be able to know which all files to be tested again if a change is made in some package. The whole concept of integration testing depends upon this.
   Solution: Maintain a metadata file for each package which will have information about all the dependent files. So now if change is made in one package our repository can tell which all packages needs to be tested along with that package. It

Build Server

Concept

The reason for introducing a build server in the SCF is to save project from the condition where the developer comes and say "Its working on my system, I don't know why it is failing here". Sometimes it happens that we have some libraries or some utility present on our personal system that the build succeeds on our machine. But when we try to compile the same thing on other system it fails. Best example of this our project fail on other system because we have given absolute path in our code.

But it is not solely the reason we can have a build server similar to our production server where the code will actually run. By doing so we can eliminate most of the issues which we might face during production deployment.

In our project we have used the logs of the build server in preparing a production helper document as it will have most of the reason because of which our system can fail.

Uses, User and User Interface

Uses

The collaboration server uses the build server to get the compile time logs which it uses for preparation of the production planning document. Apart from that the test harness uses the build server to get the executable files of the code which it needs to run.

Interface

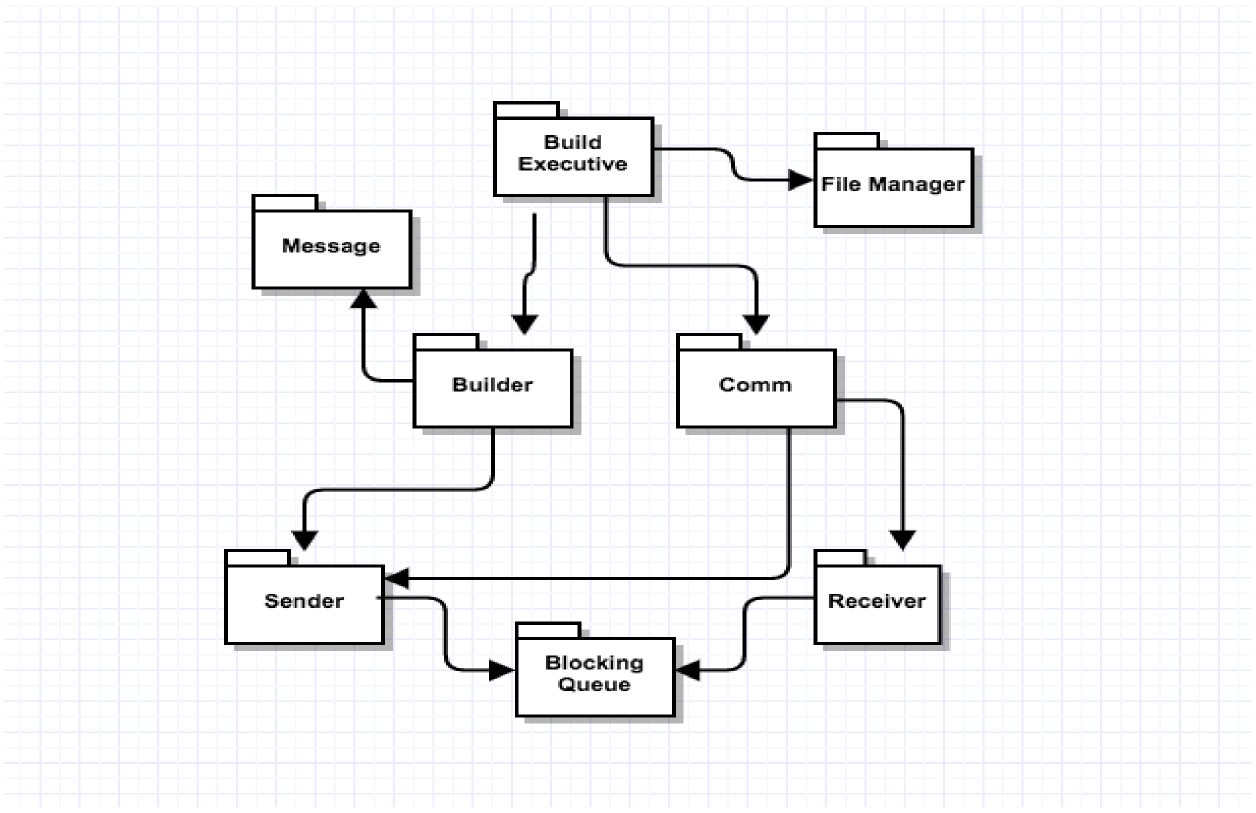
The build server has a console interface as for starting a build server it needs to know its own port number, so that build server can start a listener at that port.

Design impact

Any user does not have any direct impact on the design of the build server. But the facility of preparing the production planning document using build server has made an impact on the design. That document will be used by the "Deployment team"

STRUCTURE AND PARTITIONING

Packages



Build Executive

The Build Executive package is the controlling unit of the Build server system. It de-queues the incoming messages from the receiver package and gives control to the builder package in case it is a build request. In case it is the request for the build files of the baseline. The build executive sends those files to the test harness using the sender package. The Build executive also maintains a metadata of the dependency same as the repository. This information is passed to it by the Repository.

Builder

The builder package takes care of the task of preparing executable of the current application being developed. When a message that a new package has been checked in to the Repository. The builder package requests the repository for those files. Once it has all the files required for the built. It will compile them using a script. It will send the build status to the Client by create a message class object and putting it into the sender queue of sender package. It will also create a message with the build logs for the Collaboration server.

Receiver

The receiver package acts as the listener for the Build server system. This package creates a channel for the build server through which all other machines can send messages to it. This package has a receive message blocking queue which is used to store all incoming messages. It also provides a send message function which is used by other machines to put message into the receive queue. It also provides the facility to download a file in the build server system, this facility is used in the process of requesting files for preparing a build from Repository. The receiver package has the service class in it which is the implementation of the IService contract used for communication over WCF.

Sender

This package provides the medium for build server to send request to other system. The sender package helps in sending messages to other machines as well as upload file to other system. In order to send any request to other systems sender package create a proxy object of the machine to which the request has to be sent and then uses functions of other system service class to communicate. Whenever build server wants to send a message to other system, it puts a message in sender message queue of the sender. At the start of the build server one thread will be continuously running to send message from this sender queue.

Message

The Message package has been developed in order to provide a generic mode of communication between all the machines present in SCF. A message passes over WCF contains following information:

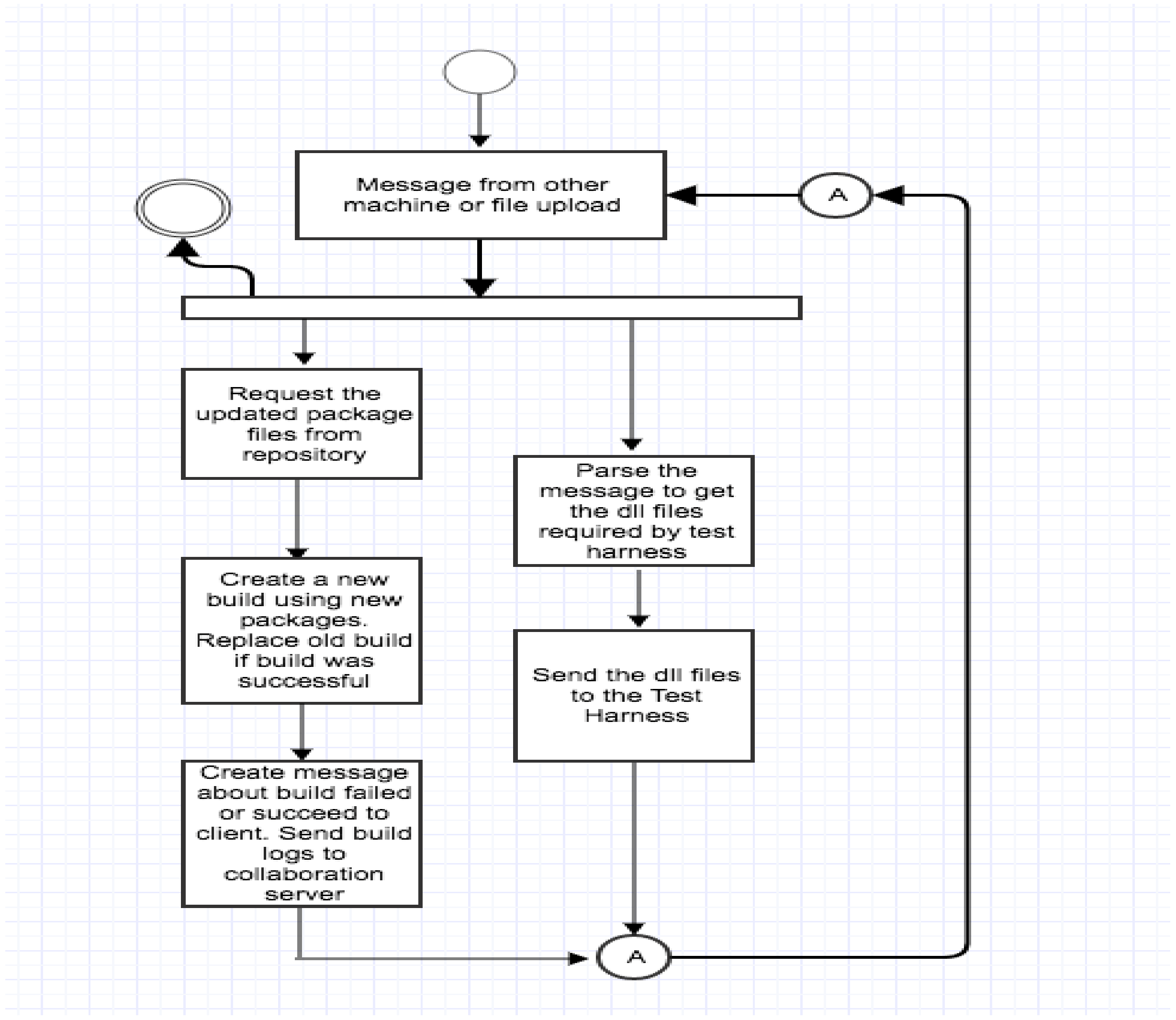
- To: URL of Receiver
- From: URL of Sender
- Author: Name of User
- Type: tells about the type of Message
- Timestamp: For holding date time
- XML Body: To hold operation oriented data.

Comm

The Comm package has been designed to remove the complexity from the executive package. The Comm package takes the responsibility to take care of the communication part of the system. It holds the sender and receiver of the system, so now the executive doesn’t need to hold them. It also keeps the track of the proxy object generated and helps in not creating the same proxy again and again by maintain a record of proxy and URL in a dictionary.

KEY APPLICATION ACTIVITIES

The major actives of the build server are to compile the code files present in current baseline to prepare a build for the baseline and provide it to the test harness to do integration testing. The flow of work done


Message from other machine or file upload
A
Request the updated package files from repository
Create a new build using new packages. Replace old build if build was successful
Create message about build failed or succeed to client. Send build logs to collaboration server
Parse the message to get the dll files required by test harness
Send the dll files to the Test Harness
A

The process of preparing a build happens in following steps:

1) **Parse Input Message from Repository:** The incoming message from the repository contains the name of the file which has been checked-in by the user, along with it's with the names of the files on which it depends. Build server creates a metadata using this information as it will help in sending dependent files to test harness for integration testing.
2) **Request Files from Repository**: Once the message has been received on the Receive queue of the Receiver package. The Build Executive pass the control to the Builder package, which parse the message body to get the names of the packages that have been updated or have been added new. The builder package uses the Sender package to send the request for required file from the Repository.
3) **Create Executable images**: Once the Build server has all the required files it will start the process of preparing the build with the help of a script. The script will have a statement wo compile all the new files. Once the script is executed the and successful the builder will replace the older version of the build with new one and will save the older version as archive and delete it after a fixed number of days like a month.
4) **Send update**: Once the building process is completed. The status of the build is send to the client and the build logs are send to the collaboration server using the sender package.

Sending executable images

The Test harness sends the request for the DLL files from the build server. The Build executive package dequeue the message from the receiver package queue. It parses the message body to get the names of the dll files required by the Test Harness and then sends those files one by one to the test harness using the send file function present in the sender package.

Critical Issues

- **Time consumption**: In our current scheme we are compiling the code files in sequence. This approach will take a lot of time if system has long queue for build.
  Solution: We can run the process of compiling in parallel instead of doing it in sequence.
- **Load**: If the load on the build server increases it won't be able to provide the test harness files at one go, which will bring the system performance down.
  Solution: We can have more than one build server in case of heavy load and use load sharing to reduce the burden

Test Harness

Concept

A Test Harness can be understood as a tool which provides facility for integration testing in an application. It provides the feature of automation in testing and generation of test reports for the user. One can formulate a test harness as

Test Harness= Test Environment + Test cases

It does not have any information regarding the test cases. In order to do its task, the test harness needs to have two type of information:

- Test Specification
- Test Cases

The test specification is the sequence of steps which needs to be carried out in order to achieve the task of testing i.e. will be taken care in our test driver. Let's take an example of testing the feature of sending email using Microsoft Outlook. The flow of testing will be "Opening the login page => Enter credential => Compose a mail=> Click send". If the test engine does not know about the sequence of these steps, then the test data is of no use. The sequence of these steps needs to be correct. Suppose the test engine tries to login to the page before even opening the login page. In that case we won't be able to test the application.

The test data helps in confirming the behavior of system for a valid input. It also helps in verification of the behavior in case of an invalid input. In the example above we can test the application once by giving the correct credentials, in that case the test engine will be able to login and send email. In order to verify the behavior for invalid input we must provide incorrect credentials. In our case the user needs to implement the test() function of ITest interface which will return true and false telling the harness if test passed or failed.

Advantages

The use of test harness by a team provides various benefits in the software development cycle:

- The facility to test complete system in case any component been modified; increases the quality of the software.
- By eliminating the need of manual testing too minimum, we are able to increases the productivity of the team.
- It makes the progress of a project more presentable with the feature of generation of reports.
- Eliminates the dependency on individual staff once the test case has been designed.

- Helps in testing an application in the conditions similar to the one for which it has been developed.

Uses, User and User Interface

Uses

The test harness provides the facility to do integration testing of an application in an environment for which it has been developed. This facility helps the development team in providing a better stable product to the user. An example of that can be the testing of a Banking application using test harness. It will help the developer to test the application thoroughly and fix the bugs. In case the developer doesn't have test harness facility for testing thoroughly then a bug might be left and the user will have pay very big amount for it as it is a banking application

Interface

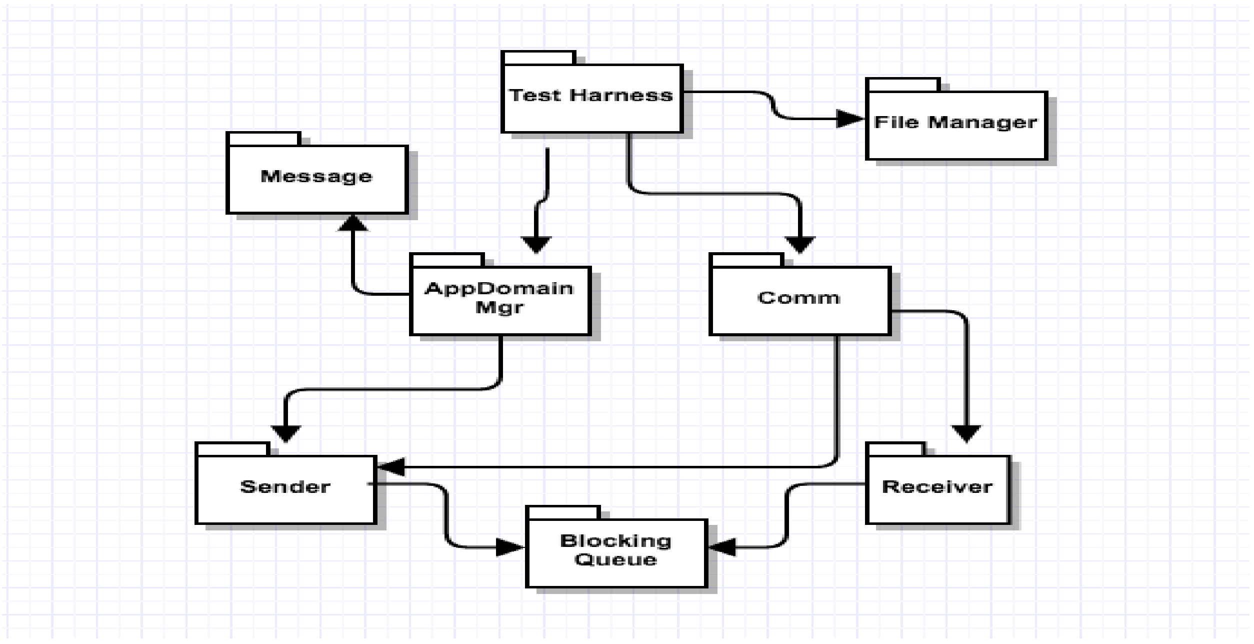
Test harness provides a console interface. As for working of a test harness, it needs to know its port number and the URL number of the Build Server from which it needs to take dll files. In the console interface the user can pass the port number of the test harness and build server as command line argument.

Users

We have discussed about how the users will use the Test harness in the federation discussion. Here we will discuss about the impact on design they made in test harness.

1) Developer's impact on design
   We designed the Test Harness in such a way to give flexibility to the developer of having as many test drivers. In order to provide this flexibility, we have used the dynamic polymorphism property of C# by creating ITest interface which allows the user to create as many test drivers that inherits ITest interface. In order to support the concept of multiple users we have used dynamic linked libraries so a single code can be used at multiple places.
2) Q.A Engineer's impact on design
   The Q.A engineer has to ensure that the standards asked by client are being maintained in the development phase. In order to do so the Q.A Engineer needs to maintain a record of old test results and logs. We have this burden from him, by sending the old test results and logs to the repository from where he can access them anytime
3) Project Manager's impact on design
   The project manager needs to know the status of the project. He should know about which all jobs are running fine and which all are failing. As we are transferring the results and logs of a test to the repository the project manager can access them anytime.

STRUCTURE AND PARTITIONING



Test Harness

The TestHarness package is the controlling package of the Test Harness system. In order to use the services of the Test Harness the user sends the message request to the TestHarness. TestHarness takes help of the Receiver package for the creation of the communication channel hosted by Harness. Once the channel has been established, the client can use the proxy object of test harness to calls the sendMessage function of the receiver package. The sendMessage function places the message into the receive message blocking queue (receiveQueue).

Once the Message is in the queue, the TestHarness uses will create a task with the help of appDomainMgr package and run it. The child thread will parse the message body to get required DLL information and use it to get files from the Repository. Once all the required files have been transferred to the TestHarness, the child thread will create object loader class in child app Domain and load the dll files. It will run the test drivers in child appDomain and save the results as well as logs in Message class object. Then it will send these results to the Client as well as the Repository with the help of sender package.

Blocking Queue

This package helps the TestHarness to have a thread safe storage for the incoming and outgoing messages. It uses monitor and locks to make it thread safe. If a thread tries to dequeue an empty blocking queue it will go into wait mode which helps in easy management of the threads.

Receiver

The receiver package acts as the listener for the Test Harness. This package creates a channel for the test harness through which all other machines can send messages to it. This package has a receive message blocking queue which is used to store all incoming messages. It also provides a send message function which is used by other machines to put message into the receive queue. It also provides the facility to download a file in the test harness, this facility is used to get the files required to run test driver.

Sender

This package provides the medium for test harness to send request to other system. The sender package helps in sending messages to other machines as well as download file from other system. In order to send any request to other systems sender package create a proxy object of the machine to which the request has to be sent and then uses functions of other system service class to communicate. Whenever test harness wants to send a message to other system, it puts a message in sender message queue of the sender. At the start of the Test Harness one thread will be continuously running to send message from this sender queue.

AppDomainMgr

The AppDomainMgr package takes care of creating the child app domains and the execution of the test drivers in it. It dequeue a message from receive blocking queue of the Test Harness, parses Message body and check if required files are present in cache, if not will request the Repository to give the files. Then create the object of loader class and load the assemblies. Once done it will execute the loader. After which it will take the result and pass it to TestHarness package for reply message creation.

The above whole process takes place in a child thread with the help of task utility C#.

FileManager

The FileManager package it used by the Test Harness for searching the code files which are needed to run the test driver in Test Harness cache. The main purpose of this package is to provide the file system access to the application.

Message

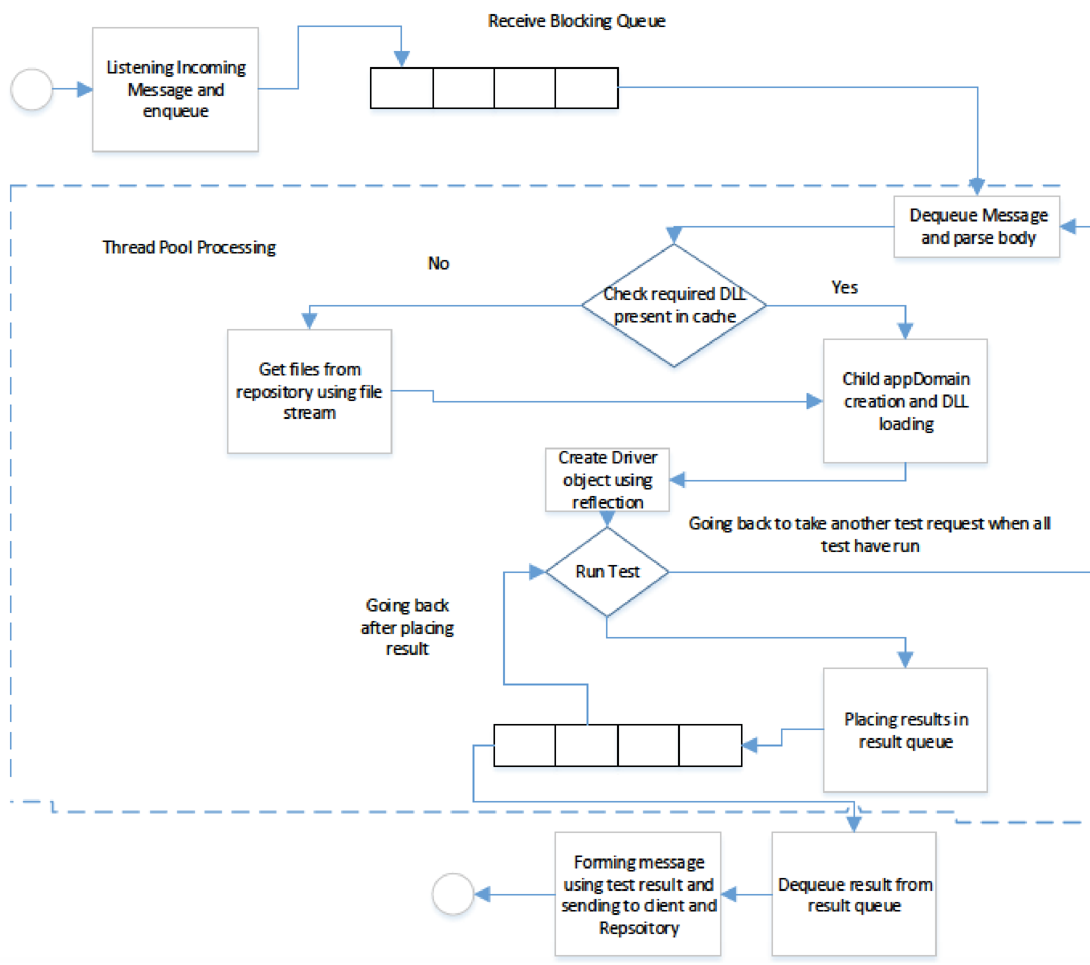
The Message package has been developed in order to provide a generic mode of communication between all the machines present in SCF. A message passes over WCF contains following information:

- To: URL of Receiver
- From: URL of Sender
- Author: Name of User
- Type: tells about the type of Message
- Timestamp: For holding date time
- XML Body: To hold operation oriented data.

Comm

The Comm package has been designed to remove the complexity from the executive package. The Comm package takes the responsibility to take care of the communication part of the system. It holds the sender and receiver of the system, so now the executive doesn't need to hold them. It also keeps the track of the proxy object generated and helps in not creating the same proxy again and again by maintain a record of proxy and URL in a dictionary.

KEY APPLICATION ACTIVITIES


Receive Blocking Queue
Listening Incoming Message and enqueue
Thread Pool Processing
Dequeue Message and parse body
No
Check required DLL present in cache
Yes
Get files from repository using file stream
Child appDomain creation and DLL loading
Create Driver object using reflection
Going back to take another test request when all test have run
Run Test
Going back after placing result
Placing results in result queue
Forming message using test result and sending to client and Repsoitory
Dequeue result from result queue

Listening incoming message and en-queue

At the time of TestHarness system creation the TestHarness package will create a TestHarness channel with the help of receiver package. TestHarness will have its own URL which will be used by other machines to send message to it. Once the channel has been created any incoming message will come through sendMessage function of the service class in receiver package. Other machine will use proxy object of the TestHarness to call sendMessage function of sendMessage function of receiver package. The sendMessage function will place the message passed in its argument into the blocking queue (ReceiveQueue) of the receiver package.

Task creation

At the time of creation of the test harness, one thread will be continuously running and will try to de-queue the ReceiveQueue of the receiver package. Once this thread gets a message it will create a task with the help of appDomainManager package execute function which will take this message as argument. The task will do below operations

- Checking DLL required in cache if not present get DLL from Repository
- Create child app domain and load dll in it.
- Creating object using Reflection and Execute the test driver in child app domain
- Save results in result sender queue of the sender package

The use of above concept will help test harness to take care of multiple test requests at same time i.e. make system capable of parallel processing

Checking for DLLs and Download from Repository

Our test harness will maintain a cache which will contain 50 recently used files to increase the efficiency of the Test Harness. Every task is a part of windows managed threadpool, it will check if the file is present in the cache, If so it will create a temporary folder with test driver name and timestamp. Then copy that file in that folder.

In case the file is not present in the Test harness cache. It will download the file from the Repository concrete service function download function in the cache directory. During the process of downloading in the cache it will also check the count of file in cache. If it is more than 50, it will delete old files to make the count 50. Once the file is in cache it will create a temporary folder with test driver name and timestamp. Then copy that file in that folder.

DLL loading and Test Execution

Once the Test Harness has all required file it will create an object of Loader class and load all the DLL files used load function of the loader class. After that it will call execute function the loader class which will execute the test driver. Inside the execute function we create object of test driver. It will use ITest interface reference variable to save the object of test driver. Using this reference variable child thread will call Test() function to test the code and then use getLog() function to retrieve logs.

Placing Results in Thread Pool Queue

After the execution of the test driver we have two type of information. One the test result and the other is the extra log part. The task saves them together as a string and place them in the sender blocking queue of the sender package. We place them in this queue as this is thread safe and many threads can place the results into it without conflict.

De-queue results, create message and send Message

Once the result is in the result queue of the sender package, sender package will dequeue it and use the data to create a message class object for the Client as well as the Repository. Once the message has been created it will send it to individual machines using the sendMessage function of the sender package.

Critical Issues

1. **File Transfer from Build server to Test Harness**: As we know that the test harness will be asking builder server for the DLL files. Suppose as a developer I am making change in a code, I will test it many times. So the test harness will need to get the file again and again from the Repository, which is doing work which uses resources for no good. Can we do it in a more efficient way?
   Solution: We can increase the efficiency of the system by just keeping the last used 50 files in the Test Harness as cache. Now whenever test harness has a test request it can first check in cache for the file and if not found then only request it from the Repository.
2. **Test Isolation**: If we run two test one test can affect the other one as they might be using same resources.
   Solution: Running all test in different child app domains will provide isolation to each test.
3. **Large testing loads**: Suppose we have a very large team and we are using test harness. If the testing load on the test harness is too much, it will bring the performance of the team down as it will take long for a test too run
   Solution: We can have multiple test harnesses. So now if the load on one harness is increasing a particular number it will start sending the overhead to other test harness.
4. **Security Issue**: We use dynamic link libraries in the working of test harness. It is a big security concern. A hacker can easily place a malicious dll file in the folder from where we are fetching the test driver files. If this file gets executed the hacker can get access to the repository.
   Solution: To solve this issue, we should keep a check if the class given in the dll file has inherited from the ITest interface. If it has not inherited from the ITest interface, then it should not be executed. We can also provide a login facility to verify the user accessing the Test harness and place the test harness on a server which is be accessible only through Virtual Private Network.

Collaboration Server

The process of software development is dependent upon a group of individual working together. The purpose of creating a collaboration server in the SCF to provide a platform in software development which can help everyone to up to date about current status of project and add management facility to the software development cycle.

Concept

The collaboration server developed as the part of SCF has following responsibilities:

- Manage Task
- Manage issues
- Generate Reports and Save Documents
- Help in production planning

By taking care of these responsibilities Collaboration server becomes of use of each and every user involved in the development cycle. Let us look into how collaboration server is taking care of these responsibilities:

1) Management of Tasks

A medium size application will involve thousands of tasks. It is very difficult to keep the track of each task at a manual level like maintaining an excel sheet. The collaboration server takes the responsibility of doing it for the user by providing a tool called task management tool.

Whenever a new task has to be assigned to an individual information about it is transferred to the collaboration server, which store this data in form of xml files and provides the facility to update these records with current status. Example:

```
<TaskDetails>
<TaskName>Login_addition_1001</ TaskName >
 <Assignor>Shishir</ Assignor >
<Assignee>Rahul</Assignee >
<TaskDescription>This task is to add login facility</ TaskDescription >
<TaskUpdate>
 <Timestamp>20140112180244</Timestamp >
 <UpdateDoneBy>Rahul< UpdateDoneBy >
  Created front End.
 </ TaskUpdate >
<TaskUpdate>
 <Timestamp>20140122180244</Timestamp >
 <UpdateDoneBy>Shishir< UpdateDoneBy >
  Rahul please prioritize working on the task.
 </ TaskUpdate>
 </ TaskDetails >
```

2) **Management of Issues**:

Knowledge sharing amongst the team member is very important. The collaboration server provides a tool which helps a team member in sharing any issue he has faced with other team members. This feature is particularly useful when the team members are sitting in different countries and can't talk to each other at personal level daily. This feature increases the productivity of team as after a short period of time most of the issues and errors are properly documented.

It can also be used to train fresher's in a project, as they can easily check the Issue Tracker to know how to solve issue. Lastly it could be used in some other project to avoid human errors made by studying the issues.

```
<IsssueRequest>
<IssueName>Login_addition_1001</ IssueName >
</ IsssueRequest >
```

3) Generate Reports & Save Documents

The collaboration server makes our project status, issues and other things more presentable. In this feature of the collaboration server helps in presenting information in terms of graph. As a customer it is very difficult to understand the status in terms of software terminologies. To help the customer to understand the work progress the collaboration server generates reports using the data provided by the Repository. These reports contain:

- the packages up and running information, Packages which are still to be developed, packages which have been developed but still having issues.
- Stability percentage of the package.
- Other package information's (author, date)

4) Production Planning

This feature of the collaboration server is different from rest of SCF features. All the other features talk about the software development process, But this feature talks about the deployment process. The collaboration server uses the logs provided by the build server prepare a delploymentHelper document. It makes an entry in this document any time the build fails due to some issue.

This document can be used by the production deployment team to foresee the situation that they might encounter in advance.

Advantages

- Acts like a white board to keep teams at different location up to date.
- Provides better management facility for task, which adds transparency to work process and helps in predicting timelines.
- Provides a Knowledge sharing point for all team members.
- Provides facility for deployment planning.

Uses, User and User Interface

Uses

Collaboration server adds the management utility to the SCF. It is used for following purposes:

- To generate reports like Project status report.
- To provide facility of task management by providing Task Mgmt. tool.
- It is user for knowledge sharing amongst team members at different location by tools like issue of defect tracker.
- Used in production deployment planning as it provides production planning document.
- Provides latest document to every user in team by providing document sharing tool.

Interface

The collaboration server provides a console based interface to get the port number over which it can start its listener. The user can pass this port number as the command line argument.

Users

We have discussed about how the users will use the Collaboration server in the federation. Here we will discuss about the impact on design they made in test harness.

1) Developer’s impact on design

   We have provided a tool Issue and defect tracker to share knowledge amongst the developer. Suppose a developer faces some issue and solves it in 2 days. Now if another person gets same issue he can check the issue tacker to know how the first developer solved the issue.

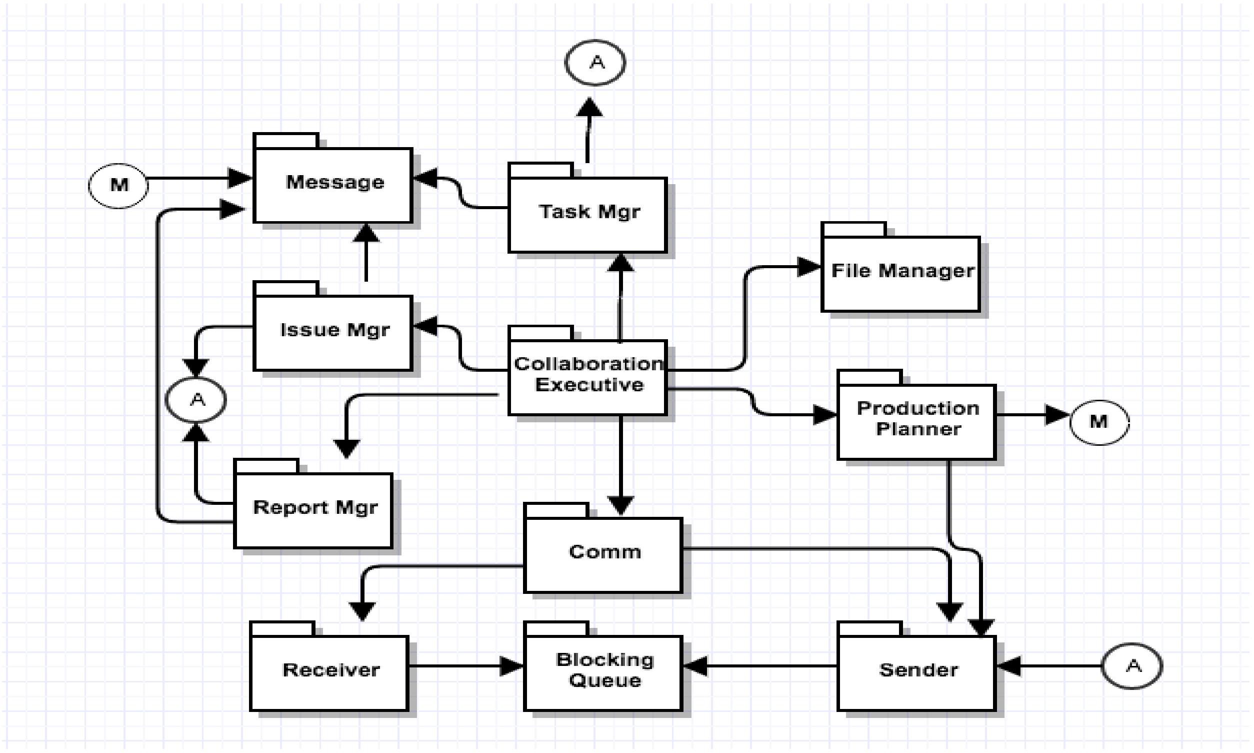
2) Project Manager impact on design
Project manager needed factual data to tell the customer about the project status and progress. The Collaboration server provides the facility of generating project status report which can be used by the project manager.

3) Social tracker impact on design
The social tracker has to keep the track of all the task assigned to the developers manually. The collaboration server task management tool provides the facility to monitor the work progress of each task without asking the developer about it.

4) Q.A engineer impact on design
Also needed a platform where he can tell the developer about the issues in his implementation like which boundary condition his code is not fulfilling. The task management tool provides that platform to the Q.A engineer. Now he can update the update section of the task management tool and the developer will take care of the issue.

STRUCTURE AND PARTITIONING

We have tried to segregate the responsibilities of the collaboration server based on the packages. Every package has been designed in such a way that it is for a particular type of task example for Task related request we have a task manager package.



Collaboration Executive

The collaboration executive package is center of the whole collaboration system. This package based on the type of incoming message received by the Collaboration system decides which package needs to take care of the job. It could be considered as the mind of the collaboration system. When a message is received at the Receive blocking queue of the receiver package the collaboration package dequeue that message and based on the type of the message call different packages like:

- Issue Mgr: for Issue related messages
- Task Mgr: for task related messages
- Report Mgr: For report and document related messages
- Production Planner: For maintaining logs passed by the build server.

Once the required operation has been completed the respective package which is taking care of the job will put a message into sender blocking queue. A sender thread will be continuously running on the sender blocking queue which will try to dequeue the queue and send message.

Task Manager

The task manager package has been designed to take care of the task related request sent by the client system. When the collaboration executive passes the message related to task to it, it takes care of the required task. The task manager will create a separate data file in xml format for each task. The task Manager will handle three type of situations:

- New task: When the task manager receives a message for new task. It will create a new xml file with below information:

  ```
   <TaskDetails>
  <TaskName>Login addition</ TaskName >
   <Assignor>Shishir</ Assignor >
  <Assignee>Rahul</Assignee >
   <Status>Open</Status>
  <TaskDescription>This task is to add login facility</ TaskDescription >
  </ TaskDetails >
  ```

  It will use the Task name and timestamp to maintain record for each task.
- Update Task: Once the task has been created the user can ask them to get updated will daily work, so that each task can be monitored by higher authorities when required. After getting the update from user the Task manager will update the existing task file by adding below information:

  ```
  <TaskUpdate>
  <Timestamp>20140122180244</Timestamp >
  <UpdateDoneBy>Shishir< UpdateDoneBy >
  ```

```
<update>Rahul please prioritize working on the task.</update>
</ TaskUpdate>
```

- Task List: User might need the list of task present in the collaboration server. To make the search process easier we provide the facility to search task based on key words. The message body will be created like below

```
<TaskList>
<Task>Login_shishir_20_may_15.xml</Task >
< Task > Admin_shishir_21_may_15.xml </Task >
</TaskList >
```

  Any task once created can’t be deleted, only its status can be changed from open to closed. Once the operation has completed it will put the message created into the sender package blocking queue.

Issue Mgr

The issue manager package has been designed to provide a tool through which the whole team can share the issues faced by them. The issue manager uses the xml form files to store the information about the issue. Once the collaboration package passes control to Issue Mgr by providing it the message class object. The issue manager package can do following:

- Create new Issue: In this case the Issue manager class uses the originator name and the issue name and timestamp to create a record of it inform of xml. A data inside the xml file will look like below

```
<IssueDetails>
<IssueName>user not found</ IssueName >
 <Originator>Shishir</ Originator >
<Solution>Database connection issue: Please use complete name</ Solution >
</ IssueDetails >
```

- Get List of issues: This operation is similar to the operation of giving the task list done by the task manager. It will also use the file manager facility to get the list of files related to a keyword.

```
<IssueList>
<Issue>Shishir_NullPtr_exception_20_may_17.xml</Issue >
< Issue > Shishir_divide_by_zero_exception_21_may_17.xml</</ Issue >
</ IssueList >
```

  Once the operation is completed by the issue manager it put a reply message in the sender package blocking queue.

Report Mgr

The Report Mgr package makes the project more presentable and reachable by all team members and Customer (using VDS). Once the collaboration comes to know the incoming message is related to the reports and document, it passes the message class object to the Report Mgr. The report Manager does two major task:

- Generating Report: In this part the report manager uses the information passes by the repository to generate reports. The repository provides the details of each package being updated and its current status to the Report Mgr. The report Mgr used this information to create xml files for each package. Example:

  ```
   <PackageStatus>
   <PackageName>Login addition</ PackageName >
    <Author>Shishir</ Author >
    <CurrenStatus>True</ CurrenStatus >
   <Update>
   <Status>true</Status>
  <Timestamp>12-13-12 </ Timestamp >
   </update>
   <Update>
   <Status>Fail</Status>
  <Timestamp>12-12-12 </ Timestamp >
   </Update>
   </ PackageStatus >
  ```

  To save the information the Report Mgr uses the author name and the file name as key combination.

- Uploading and Downloading document: The Report Manager needs to keep hold of all the important documents related to the project like Requirement document, OCD and others. The report manager gives the facility of saving these files with proper versioning and provide each user with the latest document. It helps to keep each and every member of the tam on same page.

Blocking Queue

This package helps the Collaboration server to have a thread safe storage for the incoming and outgoing messages. It uses monitor and locks to make it thread safe. If a thread tries to dequeue an empty blocking queue it will go into wait mode which helps in easy management of the threads.

Receiver

The receiver package acts as the listener for the collaboration system. This package creates a channel for the collaboration system through which all other machines can send messages to it. This package has a receive message blocking queue which is used to store all incoming messages. It also provides a send message function which is used by other machines to put message into the receive queue. It also provides the facility to download a file in the collaboration system, this facility is used by Report Mgr package in document management.

Sender

This package provides the medium for collaboration system to send request to other system. The sender package helps in sending messages to other machines as well as upload file to other system. In order to send any request to other systems sender package create a proxy object of the machine to which the request has to be sent and then uses functions of other system service class to communicate. Whenever collaboration system wants to send a message to other system, it puts a message in sender message queue of the sender. At the start of the collaboration system one thread will be continuously running to send message from this sender queue.

FileManager

The FileManager package it used by the collaboration system for searching the metadata files used for storing the task, issues and status reports. The main purpose of this package is to provide the file system access to the application.

Message

The Message package has been developed in order to provide a generic mode of communication between all the machines present in SCF. A message passes over WCF contains following information:

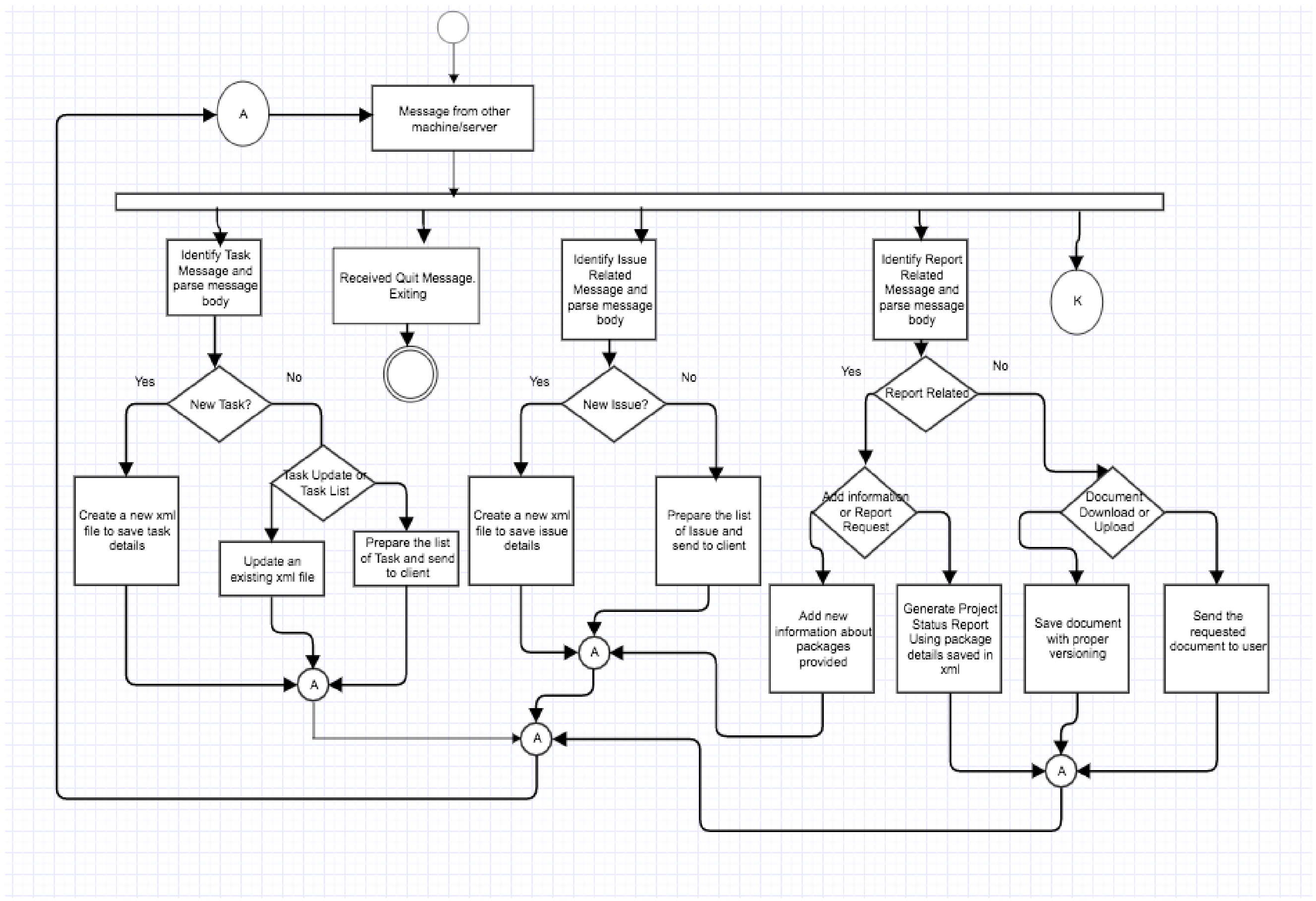
- To: URL of Receiver
- From: URL of Sender
- Author: Name of User
- Type: tells about the type of Message
- Timestamp: For holding date time
- XML Body: To hold operation oriented data.

Comm

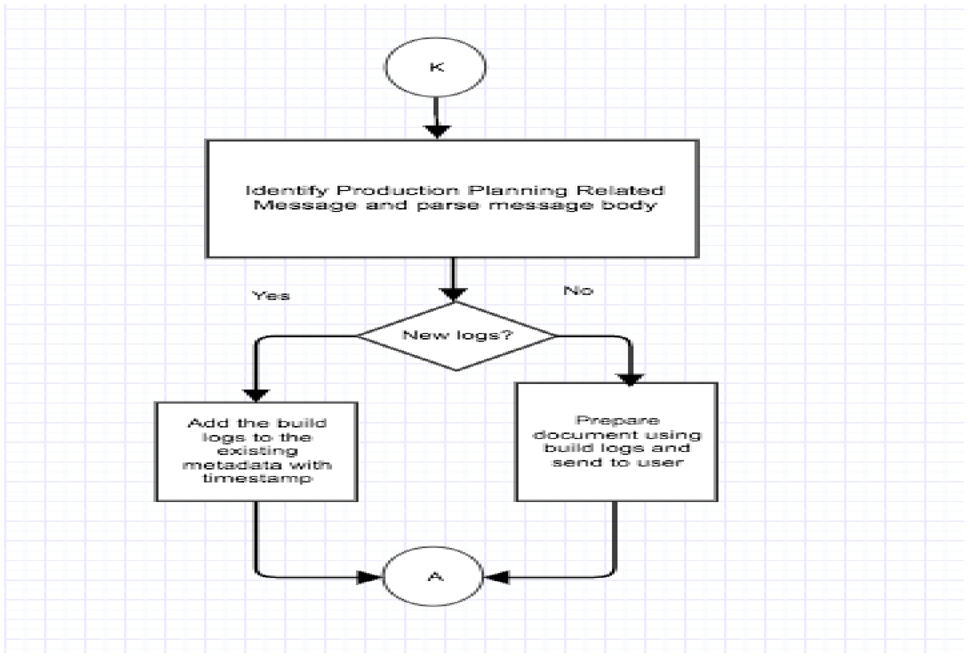
The Comm package has been designed to remove the complexity from the executive package. The Comm package takes the responsibility to take care of the communication part of the system. It holds the sender and receiver of the system, so now the executive doesn't need to hold them. It also keeps the track of the proxy object generated and helps in not creating the same proxy again and again by maintain a record of proxy and URL in a dictionary.

KEY APPLICATION ACTIVITIES

The Collaboration server is in wait state till and incoming message is received by it. Once a message is received by the collaboration server it could start processing that message. In the activity diagram below we have tried to show all the possible ways in which collaboration server can behave. Most of these steps are self-explanatory. So we have taken up the activities which require more explanation and explained.


A
Message from other machine/server
Identify Task Message and parse message body
Received Quit Message. Exiting
Identify Issue Related Message and parse message body
Identify Report Related Message and parse message body
K
Yes
New Task?
No
Yes
New Issue?
No
Yes
Report Related
No
Create a new xml file to save task details
Task Update or Task List
Update an existing xml file
Prepare the list of Task and send to client
Create a new xml file to save issue details
Prepare the list of Issue and send to client
Add information or Report Request
Document Download or Upload
Add new information about packages provided
Generate Project Status Report Using package details saved in xml
Save document with proper versioning
Send the requested document to user
A
A
A
A

As the above diagram was getting very big I have taken out a part of it and displayed separately.


K
Identify Production Planning Related Message and parse message body
Yes
No
New logs?
Add the build logs to the existing metadata with timestamp
Prepare document using build logs and send to user
A

Parsing Message Body

Once the message is received by the Collaboration system. It transfers the control to a specific package to take care of the operation required. In order to do the task required these package need to parse the xml body of the Message class. They use XDocument facility of C# to do this task. As we have discussed before different types of message, each package uses its own parser to extract information from the body.

Creating xml for Task

The collaboration server provides the facility to manage task for the user. Once it receives a request to create a new task. It will create a metadata file for it. It uses the task name and the time stamp to create the metadata file. This gives a unique name to each task. Example data in task.xml file:

```
 <TaskDetails>
<TaskName>Login addition</ TaskName >
 <Assignor>Shishir</ Assignor >
<Assignee>Rahul</Assignee >
 <Status>Open</Status>
<TaskDescription>This task is to add login facility</ TaskDescription >
```

```
</ TaskDetails >
```

Updating an existing Task

The collaboration server provides the facility to the user to update the work progress done in a day to be updated in the task metadata. The user provides the work done on a task and Task manager uses that information to be updated. Example updated files look like:

```
<TaskDetails>
<TaskName>Login_addition_1001</ TaskName >
 <Assignor>Shishir</ Assignor >
<Assignee>Rahul</Assignee >
<TaskDescription>This task is to add login facility</ TaskDescription >
<TaskUpdate>
 <Timestamp>20140112180244</Timestamp >
 <UpdateDoneBy>Rahul< UpdateDoneBy >
  Created front End
  </ TaskUpdate >
<TaskUpdate>
 <Timestamp>20140122180244</Timestamp >
 <UpdateDoneBy>Shishir< UpdateDoneBy >
  Rahul please prioritize working on the task
  </ TaskUpdate>
 </ TaskDetails >
```

Creating new issue

Whenever a user sends the information about some new issue or error the issue manager package uses that information to create an xml file. Every issue has its own xml file and we use the issue name and the timestamp as key combination to save it. A sample issue xml look like below:

```
<IssueDetails>
<IssueName>user not found</ IssueName >
 <Originator>Shishir</ Originator >
<Solution>Database connection issue: Please use complete name</ Solution >
</ IssueDetails >
```

Generating Deployment Helper

The Collaboration system constantly gets information about the build from the build server n form of logs. The collaboration server uses this information to be record maintained, so that this information can be used latter for production implementation and avoid situation that we faced on the build server.

```
 <BuildInformation>
 <Timestamp>22-May-2017<Timestamp>
 <Status>Failed</Status>
 <Issue>Database not found</Issue>
</BuildInformation>
```

The collaboration server saves this information with Application name and timestamp in form of xml document.

Generating Project Status Report

Whenever a file is checked in to the repository or a test driver is run. The Repository passes that information to the Collaboration server. Collaboration server maintains records at each package level. Each package has its own metadata file. An example xml looks like below:

```
<PackageStatus>
 <PackageName>Login addition</ PackageName >
  <Author>Shishir</ Author >
  <CurrenStatus>True</ CurrenStatus >
 <Update>
 <Status>true</Status>
<Timestamp>12-13-12 </ Timestamp >
 </update>
 <Update>
 <Status>Fail</Status>
<Timestamp>12-12-12 </ Timestamp >
 <Issue>File not found exception</Issue>
 </Update>
 </ PackageStatus >
```

Every time a test driver runs for a package and update is made to existing metadata file for that package. The Report Manager uses this information to generate following

- Percentage of package up and running.
- Stability graph of a package over time.
- Percentage of application complete.

Critical Issues

1. **Delay in communication between Multiple user:** Suppose V.D. server was not present in our federation. Then the part of communicating and acting as a common platform between all the virtual clients would have been done by collaboration server. This would have cause delay in reflecting the changes of common area in virtual area as collaboration server could not have taken that much load.
   Solution: To have a dedicated server for taking care of communication between the virtual clients.
2. **Concurrent file access:** Collaboration server will also face the same issue as repository of concurrent file access as many users can ask for the project status report at same it or any other document which is maintained by the Collaboration server.
   Solution: We can make the thread sleep when an exception is thrown while reading and try to read again. This can be done for fixed number of time say 4.

Client

The main idea for designing a client system was to provide access to the web services provided by the Test Harness, Repository and the Collaboration server. We have developed the GUI in such a way that a user can use services from all these three server by it.

Concept

During the designing of the client major weightage has been paid in the GUI. We will develop GUI in such a way that it becomes easy to use and requires minimum efforts from the side of the user. Clients major interaction with the three servers can be summed up as below

- Repository: The user uses the facility of uploading, downloading file and getting old logs from the repository. In order to use these three services of the Repository we have designed three different tab in the GUI. In order to use these services, the User needs to provide below detail:
    1. Uploading: The complete file name of file which has to be uploaded.
    2. Download: The file name of the file which has to be downloaded from repository.
    3. Log Request: The user needs to know the log file name to get details.

  We have designed our GUI so that minimum input is required for all these operations

- Test Harness: The main purpose for which the client uses the services provided by the Test Harness is to do integration testing. In order to do integration testing, the user needs to make sure first that all the files required to run the test have been uploaded to repository. For using the Test harness user needs to provide the name of the test driver and dependent code files to the GUI. The GUI will use this information to form a Test request.
- Collaboration Server: Collaboration server provides the management and information sharing platform to all the clients. We have designed the GUI so that user can take maximum advantage of the tools provided by the Collaboration server.

Advantage

- It will minimize the human errors while using the SCF.
- It provides ease for use as not much manual input is required.
- It makes the data more presentable and easy to understand.

Uses and Users

Uses

Different users use the client system for different purposes. The GUI of the client system provides a comfortable way to the user for using the web services provides by different servers in SCF. Let us look into various ways the users use the Client system:

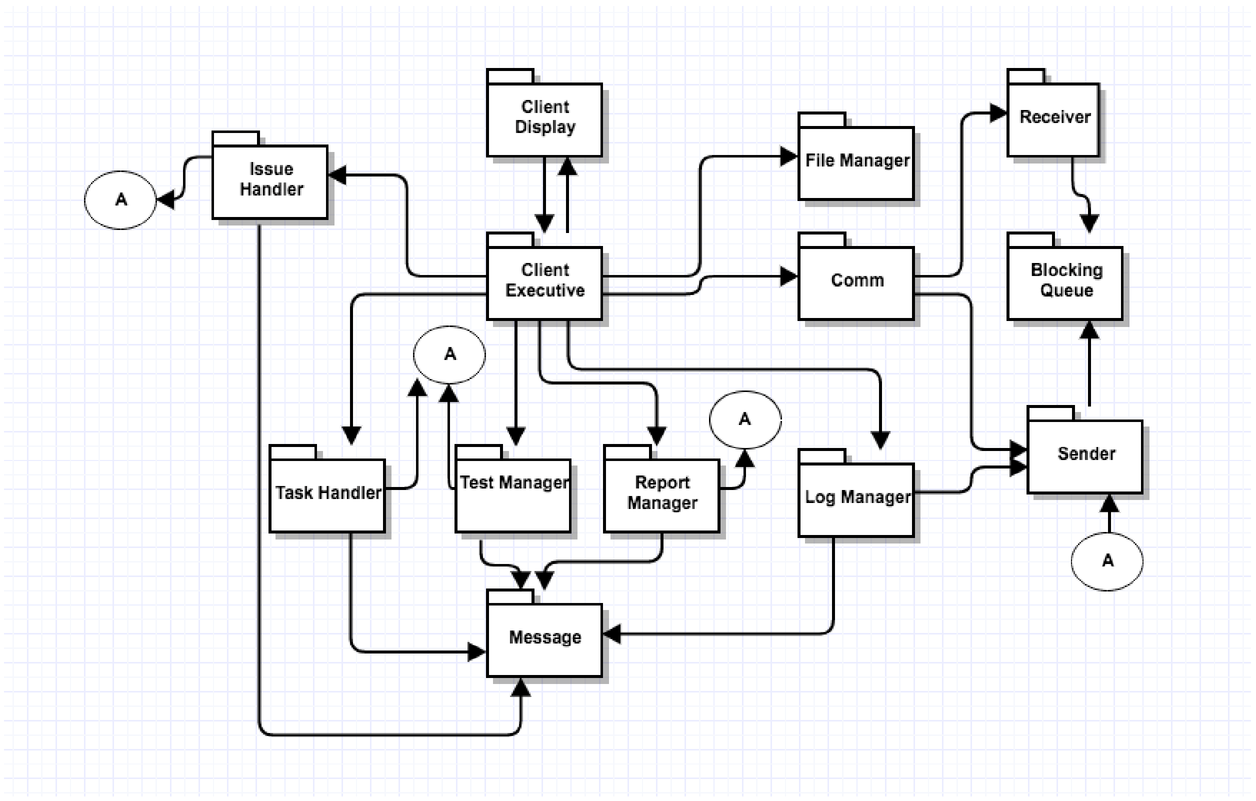
- Developer: Uses it to interact with the test harness, Repository and Collaboration server to use the services they provide like testing facility, code file access and Task management tool.
- Q.A Engineer: also uses it to interact with the test harness, Repository and Collaboration server, but the intent is different. The Q.A uses the collaboration server task management tool to update the issues in the current developed version, Repository to get the old logs rather than code files.
- Project Manager: Project Manager uses it to interact with the Repository to get old logs and for interacting with the collaboration server to use its management related utilities.
- Tech Architect: Use it for interacting with the collaboration server. The Tech. Arch. Might need the latest requirement document or he might need to update the design document in the collaboration server. In that case he uses the document management tab of the GUI
- SME and B.A.: Also uses the client system to interact with the collaboration server. They might use it to update the latest requirement document into the collaboration server.

Users

We have discussed about how the users will use the client in the federation. Here we will discuss about the impact on design they made in test harness.

1) Developer's impact on design
   It is very difficult to remember the exact names of the test driver and the dependent code files for the developer. So to save him from the trouble we have given the option to search files based on key words in GUI tab run test.
2) Q.A and Project Manager impact on design
   The Q.A. and Project Manager might not remember the exact date when the developer has run his code. So they won't be able to get the log file and results for a test. To make the job easier for them we have given the option to search log file with help of keywords.
3) Tech. Arch., SME and B.A impact on design
   We have provided the facility to upload the project related documents into the collaboration server. No other user apart from them will upload the project related documents like Requirement document and the design document.

STRUCTURE AND PARTITIONING



Client Display
This purpose of this package is to provide the graphical user interface (UI) to the user. The user interface will have eight tabs with individual functionality:

- To set up connection with Test Harness, Repository and Collaboration server and provide user input.
- For uploading files to repository.
- For downloading files from repository.
- For Running Test.
- For Requesting Test logs.
- Task Management tab.
- Issue Tracker tab

- Document and Report Downloading tab

Client Executive

Once Client Executive package has the user input and the instruction from the Client Display package, it will make use on of the manager or handler package to create a message and put it in the blocking queue (Sender Queue) for sending message. Client Executive package will send message to TestHarness Server, Collaboration Server or the Repository server, based on the type of the request using Sender package which provide the proxy object of the target server. The sendMessage function of sender package will dequeue request from Sender Queue and send it to host.

The client package will work in two phases. First has been discussed above i.e. sending message to the target server. The second phase deals with receiving of messages from TestHarness server, Collaboration Server or the Repository. The receiver package helps the client in creating a channel to listen all incoming request. Once the listener has started the Client Service class present in Receiver package helps the client in putting the incoming messages into the received message blocking queue (Receive Queue). After the request is in Receive Queue a thread will dequeue it and pass the control to the package based on the type of the Message. Then these manager package with use the information present in to display content on GUI.

Test Manager

The test manager package has been designed to take care all the request meant for the Test Harness. When user select the option of running a test. The Test Manager package get the user input in terms of file name from the user. The Test manager package then uses this information to create a message class object for Test Harness and put into the sender blocking queue of sender package.

Once the test has run the client get the message from the Test harness giving the details about the run. Then the test manager package uses the information from incoming message and display it on the screen.

Log Manager

The Log manager package has been designed to take care all the request meant for the Repository. When the user wants to upload or download a file. The log manager uses the file download or upload functionality of the sender package, which in turn uses the service package of the Repository. In case of a logs request the Log manager gets the name of log file whose data is required. The Log manager package then uses this information to create a message class object for Repository and put into the sender blocking queue of sender package.

In case of file download and upload it will be a synchronous communication and the log manager will come to know once the file has been uploaded or downloaded. In case of the log request the log manager will have to wait for the message from the Repository with test results in its message body. Once Log manager has this message it uses it do display content on GUI.

Report Manager

The report manager package has the functionality similar to that of the log manager, it has the functionality to upload the files and download the files to the collaboration server. Using the report manager package senior persons like the Tech Architect, Leads and Manager can upload documents like Requirement Document, OCD, Time lines document and others.

These documents can be downloaded by other people when required using the Report Manager package. Similar to the Log manager package the Report manager package uses the upload and download facility of the sender package to download and upload files into the collaboration server.

Task Hander

This package has been designed to help the user use the task management tool of the collaboration server. Using this package, the Tech Architect and Social tracker can create a task and assign it to a developer. To help the task manage the user provide the information about the task and the developers name to the GUI. Then the Task Handler package uses this information to create a message class object for Collaboration server and put into the sender blocking queue of sender package.

When the Client received a message related to the task then the task handler uses this information to display content on task management tab of GUI.

Issue Handler

This package has been designed to help the user use the Issue management tool of the collaboration server. Using this package, a developer can provide a description of the issue and the possible solution for it. Then the Task Handler package uses this information to create a message class object for Collaboration server and put into the sender blocking queue of sender package.

When the Client received a message related to the task then the task handler uses this information to display content about the issue and its possible solution on task management tab of GUI.

Message

The Message package has been developed in order to provide a generic mode of communication between all the machines present in SCF. A message passes over WCF contains following information:

- To: URL of Receiver
- From: URL of Sender
- Author: Name of User
- Type: tells about the type of Message
- Timestamp: For holding date time
- XML Body: To hold operation oriented data.

Comm

The Comm package has been designed to remove the complexity from the executive package. The Comm package takes the responsibility to take care of the communication part of the system. It holds the sender and receiver of the system, so now the executive doesn't need to hold them. It also keeps the track of the proxy object generated and helps in not creating the same proxy again and again by maintain a record of proxy and URL in a dictionary.

Receiver

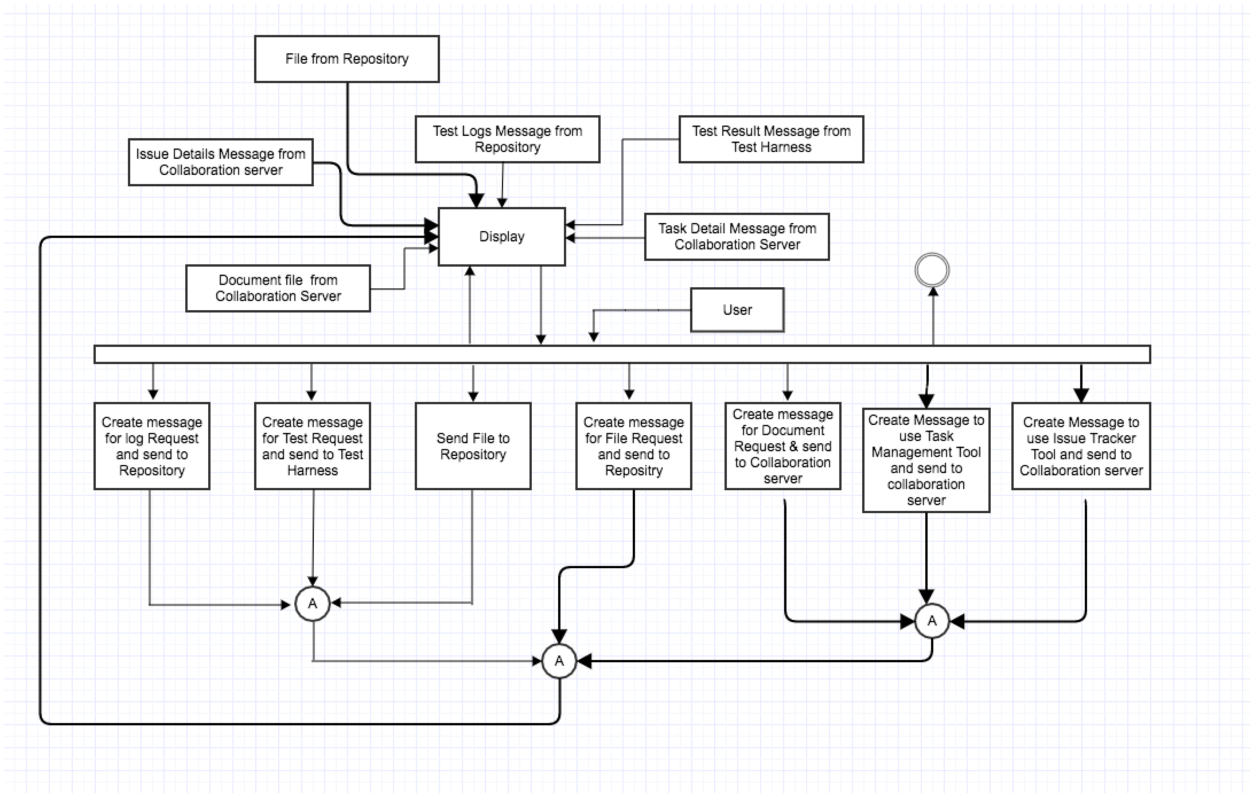
The receiver package acts as the listener for the Client. This package creates a channel for the client through which all other machines can send messages to it. This package has a receive message blocking queue which is used to store all incoming messages. It also provides a send message function which is used by other machines to put message into the receive queue. It also provides the facility to download a file in the client system, this facility is used to download code files as well as documents and reports.

Sender

This package provides the medium for client to send request to other system. The sender package helps in sending messages to other machines as well as download file to other system. In order to send any request to other systems sender package create a proxy object of the machine to which the request has to be sent and then uses functions of other system service class to communicate. Whenever client wants to send a message to other system, it puts a message in sender message queue of the sender. At the start of the client one thread will be continuously running to send message from this sender queue.

KEY APPLICATION ACTIVITIES

The key activities of the client system involve creation of messages to get the operation done by hosting servers and to get the information content from the message body display it on the GUI. Below is the activity diagram for the client system. We have taken each of these activities and described them in detail.


File from Repository
Test Logs Message from Repository
Test Result Message from Test Harness
Issue Details Message from Collaboration server
Display
Task Detail Message from Collaboration Server
Document file from Collaboration Server
User
Create message for log Request and send to Repository
Create message for Test Request and send to Test Harness
Send File to Repository
Create message for File Request and send to Repository
Create message for Document Request & send to Collaboration server
Create Message to use Task Management Tool and send to collaboration server
Create Message to use Issue Tracker Tool and send to Collaboration server
A
A
A

Test Request Message creation

Using the information provided by the user in the GUI the Test manager package create an object of message class. The message class object has following content:

To: <URL of Test Harness>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Test_Request

```
Body: <Test>
          <TestDriver>TestDriverOne.dll</ TestDriver >
          <DependentFiles> CodeToBeTested.dll</ DependentFiles >
          <DependentFiles> CodeToBeTested2.dll</ DependentFiles >
          </Test>
```

Once the message has been created it is send to the Test Harness by putting the sender message queue of sender package.

Create Message for log request

In case the user wants an old test run information from the repository, it provides the name of the log file. Using this name, the log manager package creates a message class object to get the details of the log file from the repository. The content of the message class is as below:

To: <URL of Repository>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: log_data
Body: <LogFile>
Shishir_12_May_2015.xml
</ LogFile >

Once the message has been created it is send to the Repository by putting the sender message queue of sender package.

Create Request for Task Details

When the user wants to use the Task Management tool of the Collaboration server. User passes information from the Task Manager tab of the GUI. In this tab he passes the information of the task he wants to get detail. Using this information, the Task Manager forms the message class object with similar details:

To: <URL of Collaboration server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Task_Request
Body: <TaskRequest>
<TaskName>Login_addition_1001</ TaskName >
</ TaskRequest >

Once the message has been created it is send to the Collaboration server by putting the sender message queue of sender package.

Create Issue message

When the user wants to get the details of an old log or wants to create a new issue he uses the issue manager tab of the GUI. Here the user can select the option to either create a new issue and provide information or to get details about an existing issue. The Issue manager uses this information and create messages to get the job done. Below are both type of messages:

```
To: <URL of Collaboration Server>
  From: <URL of Sender>
  Author: Shishir Bijalwan
  Timestamp: 12/14/2016
  Type: New_Issue
  Body: <IssueDetails>
          <IssueName>user not found</ IssueName >
           <Originator>Shishir</ Originator >
          <Solution>Database connection issue: Please use complete name</ Solution >
          </ IssueDetails >
```

Get Existing Issue details:

```
To: <URL of Collaboration Server >
  From: <URL of Sender>
  Author: Shishir Bijalwan
  Timestamp: 12/14/2016
  Type: Issue_Request
  Body: <IsssueRequest>
          <IssueName>Login_addition_1001</ IssueName >
          </ IsssueRequest >
```

Once the message has been created it is send to the Collaboration server by putting the sender message queue of sender package.

Task Detail Message from Collaboration server

Once the Collaboration server has operated on the request to give the Task details. It sends a message to the Client with task details. The details in the message body looks like below:

```
Body: <TaskDetails>
        <TaskName>Login_addition_1001</ TaskName >
         <Assignor>Shishir</ Assignor >
        <Assignee>Rahul</Assignee >
        <TaskDescription>This task is to add login facility</ TaskDescription >
        <TaskUpdate>
         <Timestamp>20140112180244</Timestamp >
         <UpdateDoneBy>Rahul< UpdateDoneBy >
          Created front End.
         </ TaskUpdate >
        <TaskUpdate>
         <Timestamp>20140122180244</Timestamp >
         <UpdateDoneBy>Shishir< UpdateDoneBy >
          Rahul please prioritize working on the task.
         </ TaskUpdate>
         </ TaskDetails >
```

The details from the above message body is parsed by the Task Manager class and display content one the GUI.

Test Result from the Test Harness

Once the test harness has run the test, it will send the results to the client as well as the Repository. The client receives a message from the test harness and the test manager class parse information from that message body and display it on the GUI. The message body with test result looks like below:

```
Body: <TestResult>
        <Author>Shishir Bijalwan</Author>
        <Timestamp>20140112180244 </ Timestamp >
        <TestDriver>TestDriverOne.dll</ TestDriver >
        <DependentFiles> CodeToBeTested.dll</ DependentFiles >
        <DependentFiles> CodeToBeTested2.dll</ DependentFiles >
        <Passed>True</Passed>
        <Logs>Null pointer exception</Logs>
        </ TestResult >
```

Issue Message from the Collaboration Server

The collaboration server provides the details of the issue requested by the user by sending it over using a message class object. The body of the message received at client is as below

```
Body: <IssueDetails>
      <IssueName>user not found</ IssueName >
       <Originator>Shishir</ Originator >
      <Solution>Database connection issue: Please use complete name</ Solution >
      </ IssueDetails >
```

The issue manager package uses this information and update the GUI with issue details.

File, Document or Report request:

In case of any file download request the Report manager uses the sender package download file function to call the service class method of the Repository and the collaboration server file upload method. The sender class uses the proxy object of the service class to use it functionality. The file transfer process is a synchronous communication and could freeze the GUI. So special care must be take while calling these methods as the size of the file can be very large and the GUI can become unresponsive for that duration of time. It is a good practice to start thread for these type of request.

Critical Issues

1. Communication: In case the client was using synchronous mode of communication then client has to wait for the reply from the server. In that case the client won't get free and won't be able to do anything else.
   Solution: This issue can be resolved by using asynchronous mode of communication where client sends a request and has not to wait for a reply.

Virtual display System

Concept

The virtual display system is a platform provided for team member at different location to interact with each to design and plan software development process.

Uses and Users

Uses

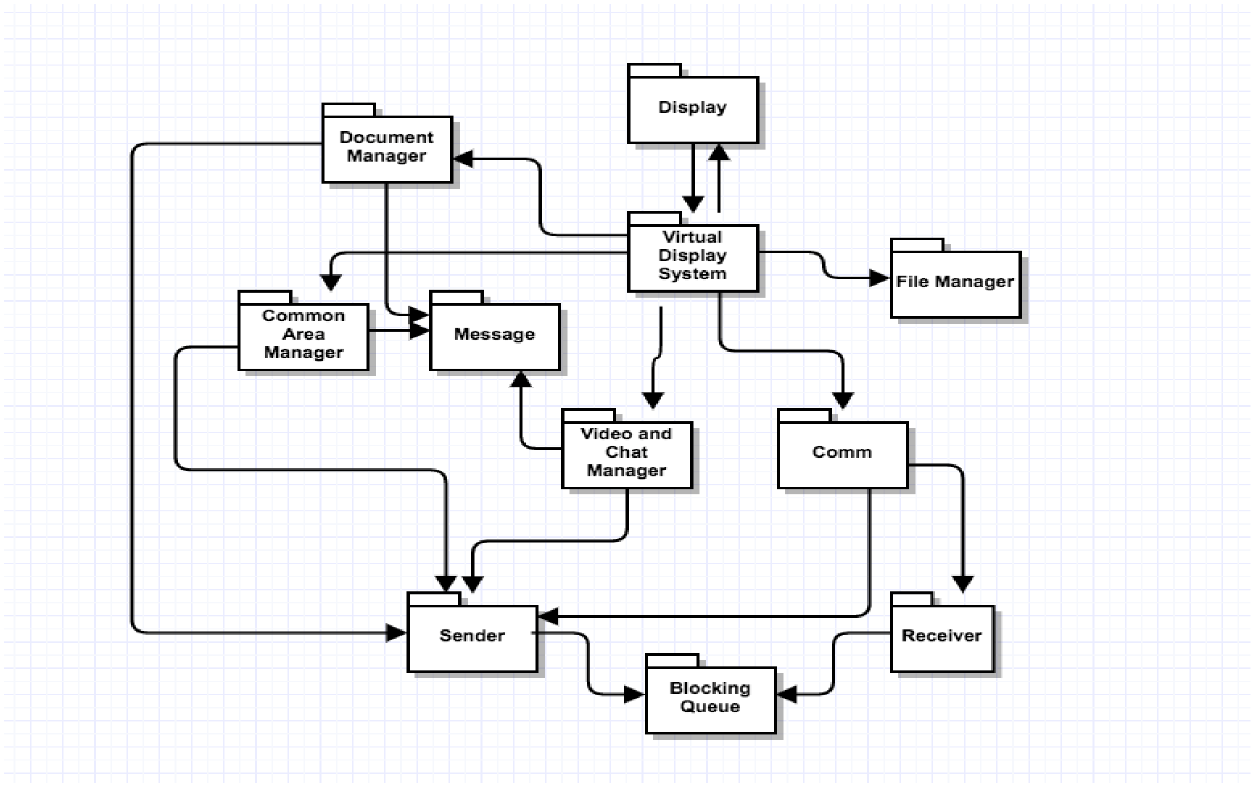
The virtual display system provides a touch sensitive screen which is used by the user to pass instruction to the V.D system. The user uses this system as a discussion platform to discuss things with the customer or the team members present at different locations.

Users

We have discussed about how the users will use the Virtual Display system in the federation. Here we will discuss about the impact on design they made in test harness.

1) Technical Architect's impact on design
   The job of the Tech. Arch. Is to design the software architecture which requires constant writing and drawing facility during discussion. So here we have a touch sensitive screen which can be used to design and sketch things on the screen.
2) Project Manager impact on design
   Project manager has to regularly update the client with current status of the project. So we had to provide the facility to display the reports and documents on the screen.

STRUCTURE AND PARTITIONING



Display

This purpose of this package is to provide the Touch sensitive user interface. The user interface will have eight block with individual functionality:

- Chat window: To provide the facility to chat at group and personal level.
- Video Streaming: To have live face to face communication with other teams.
- Common Area: Area where team member can draw and discuss things. Needs authorization.
- Code Display: To display code from repository.
- Report Display: Area to display Reports such as project status report.
- Personal Notes: This area is for discussion for team at particular location. Not visible to others
- Calendar: To set task for the day and set up meetings
- Project Requirements: This displays the requirements for the project and can be interchanged with any document to display.

All these blocks can be magnified by clicking on them.

Virtual Display System (V.D. System)

This package interacts with the display package and transfer control to one the package discussed below based on the type of input passed:

- Document Manager package in case the request is related to Report or document or code Files.
- Chat and Video: At the start of the Virtual client the V.D. System package uses the facility of Video and chat manager to starts sending the live feeds from the client location to the Virtual display server. The V.D. System package also uses the video and chat manager to send message updates passed from the display.
- Common Area Manager: Whenever an authorized user makes changes in common area the V.D. System package informs the common area manager package which take of sending the update to the Virtual display server.

Document Manager

The Document manager package has two jobs:

1) To send request for Report, Document or code file: For this Document manager creates the object of Message class with file information in the body and puts it into the sender’s package blocking queue.
2) Incoming Report, Document or code File: The incoming files are copied into a temporary location on the V.D system. From there the Document manager extracts the data from the file and displays it on the screen.

Common Area Manager

The common area manager packages have been designed to take care of three types of task:

1) Request Authorization: When the user request to make changes in the common area. The common area manager sends a request to Virtual display server requesting authorization. Once given the user can draw and write in common area.
2) Drawing and writing: Once the user has been authorized any changes made in the common area should be sent to the Virtual display server. This task is taken care by the Common area manager with the help of the sender package.

3) Receiving update: This type of update will come for user who are not making the changes and wants to see the changes that authorized clients are making. This again will be stream of data with images with the rate of 30 frames per second.

Video and Chat Manager

This package will take care of the of all the functionality related to chats and video session being done by the user. Let us understand the working of this package in two parts:

1) Chat facility: Whenever user enters any data in chat window and press enters. The Video and chat manager package will create a message class object and attach the chat to message body. Once the message has been created it will put it into the sender blocking queue.
   For incoming messages for chat update (which will be sent other users). The video and chat manager will get the update from the body of the message and use it to update the content in chat window.

2) Video Facility: For a normal human eye if we move 30 images in a second our eye take it as a video. So at the start of the virtual client the Video and Chat manager package starts sending the current feeds (video) of meeting room to the Virtual display server. It uses the send Livestream function of the sender package to send video stream
   The virtual display server gets live video from all location and sends it to all the virtual clients. The video and chat manager again collects these video that are images at virtual client side and displays them on screen.

Receiver

The receiver package acts as the listener for the V.D System. This package creates a channel for the V.D System through which all other machines can send messages to it. This package has a receive message blocking queue which is used to store all incoming messages. It also provides a send message function which is used by other machines to put message into the receive queue. It also provides the facility to download a file in the V.D System, this facility is used in the process of display code files, documents and report on screen. The receiver package has the service class in it which is the implementation of the IService contract used for communication over WCF.

Sender

This package provides the medium for V.D System to send request to other system. In order to send any request to other systems sender package create a proxy object of the machine to which the request has to be sent and then uses functions of other system service class to communicate. Whenever V.D System wants to send a message to other system, it puts a message in sender message queue of the sender. At the start of the V.D System one thread will be continuously running to send message from this sender queue.

Message

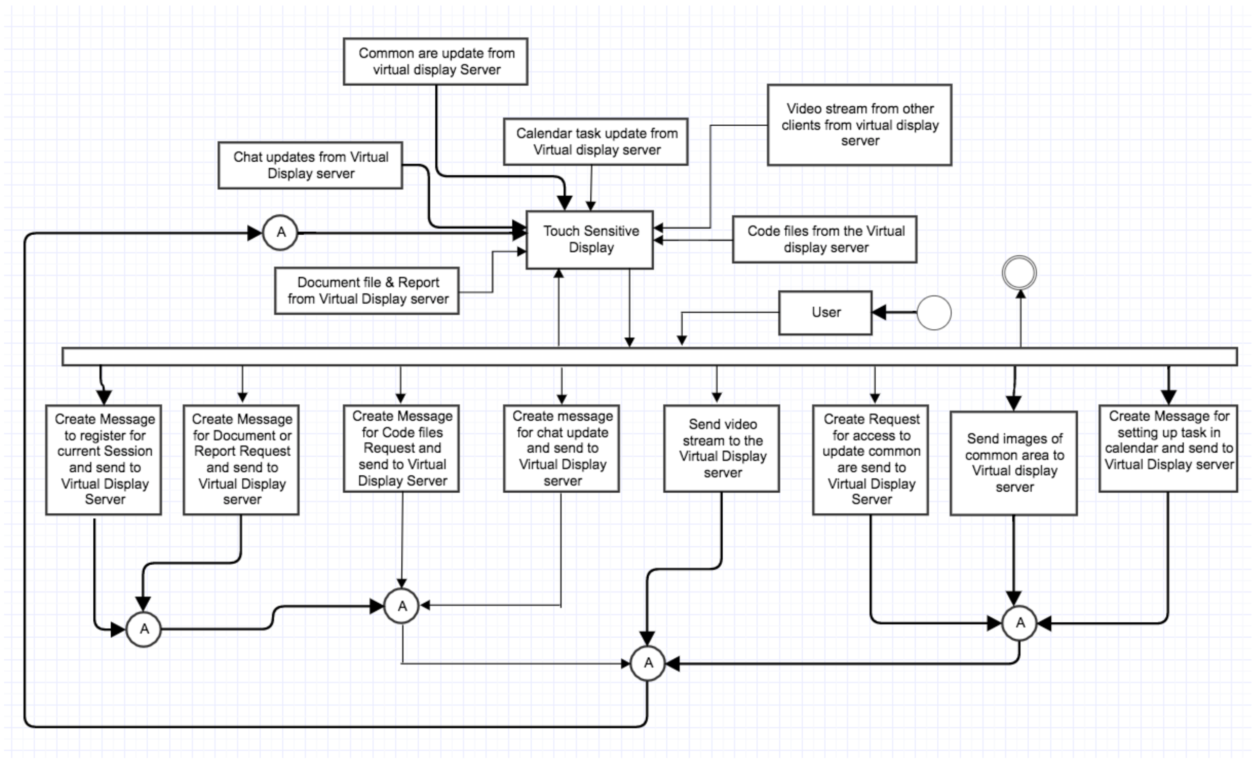
The Message package has been developed in order to provide a generic mode of communication between all the machines present in SCF. A message passes over WCF contains following information:

- To: URL of Receiver
- From: URL of Sender
- Author: Name of User
- Type: tells about the type of Message
- Timestamp: For holding date time
- XML Body: To hold operation oriented data.

Comm

The Comm package has been designed to remove the complexity from the executive package. The Comm package takes the responsibility to take care of the communication part of the system. It holds the sender and receiver of the system, so now the executive doesn’t need to hold them. It also keeps the track of the proxy object generated and helps in not creating the same proxy again and again by maintain a record of proxy and URL in a dictionary.

KEY APPLICATION ACTIVITIES


Common are update from virtual display Server
Video stream from other clients from virtual display server
Calendar task update from Virtual display server
Chat updates from Virtual Display server
A
Touch Sensitive Display
Code files from the Virtual display server
Document file & Report from Virtual Display server
User
Create Message to register for current Session and send to Virtual Display Server
Create Message for Document or Report Request and send to Virtual Display server
Create Message for Code files Request and send to Virtual Display Server
Create message for chat update and send to Virtual Display server
Send video stream to the Virtual Display server
Create Request for access to update common are send to Virtual Display Server
Send images of common area to Virtual display server
Create Message for setting up task in calendar and send to Virtual Display server
A
A
A
A

Register for a session

In order to be the part of any meeting the virtual client has to send a request to the virtual display server to join it. The message body will have the name of the team and type will be Registration Request.

Create Message for file request

User has the facility to request files to be displayed on the screen. For this the document manager package will create a message with the name of the file and the type of file requested example code file, report or document. Once the message has been created it will be put into the sender queue of the sender package will send it to virtual display server using its proxy.

Create chat update message

When the user types message in the chat window and press enter. The chat and video manager package get the information about the message content and the user or group for which the message was sent. Then it creates a message with type chat update and attach a body with the chat information. A sample message body for chat update message will look like below:

<ChatUpdate>

```
<to>Group</to >
 <from>shishir</from>
 <Timestamp>12-12-16</ Timestamp >
<Message>Database connection issue: Please use complete name</Message >
</ ChatUpdate >
```

Once the message has been created it will be put into the blocking queue of the sender package.

Creating message for common area Access

In order to draw things in the common area the user need authorization from the virtual display server. The common area in the display will show maximum number of user that can be authorized and their images. If any image block is empty the user can click on it to send the request.

Sending files and common area images

For any of the above task the virtual display system needs to transfer data in bytes to the virtual display server. This facility is provided by the sender package. The sender package will have a function which can break files into chunks of bytes and send it over the WCF channel to the Virtual display server.

Receiving chat update message

The chat update message is send by the virtual display server to all the active clients in case of a group message and to the individual client in case of personal message. Once the message has been received by the chat and video manager package it will extract the chat information from the body of the chat and use it to update chat window on the display.

Receiving Document, report or code file

The virtual server creates the proxy object of each and every client present in current session and send all the files to every user. When the receiver package receives a file from the virtual display server. It creates a copy of this file in the temporary folder. Once the file has been copies the virtual display system will use the content from these files and display it on the screen.

Calendar update

The virtual client display provides the facility to set reminder for the meeting task or to block time for the next meeting. Again it will create a message with type calendar_update and body with the information about the update made in calendar. Once the message has been created it is send to the virtual display server with the help of sender package.

Once the calendar_update message reaches the server it sends to all the active clients in the session. Once received at other virtual clients the virtual display system package makes the changes in the calendar based on the update sent in the message body.

Critical Issues

1. Camera direction setting: The virtual systems will be used in a conference room and can involve many people attending it. When someone speaks it will be difficult manually setting up the camera over that person. If it is not proper the people sitting at different location won't be able to have proper communication
   Solution: If we can develop a software which turns the direction of the camera based on audio waves direction then we don't have to do the job manually.

Virtual Display Server

Concept

The virtual display server is the connecting thread between the n virtual display systems active in a session. The virtual display server has been put in our federation to full fill two jobs:

1 The virtual display server takes away the extra responsibility from each of the virtual client to know all the other client present in a session. Every client knows only one other machine that is the virtual display server. Any communication they want to do, that will go through the virtual display server. The good thing about this type of model is that by sending the information to the server and the server distributing it is that we have the surety that every virtual client is looking at the same thing.

2 The job that is being done by the Virtual display server of connecting the virtual display users together could have been done by the collaboration server also. The reason for developing a dedicated server for this job was to avoid any type of communication delay that can happen if the server is busy working on something else and does not have enough thread to communicate a message from one virtual client to all the other client. Just imagine a situation where Tech Architect has made a design in the "common design area" but other team members are waiting for the changes to reflect on their screen.

Uses and Users

Uses

The virtual display server package provides the common ground to the virtual client to interact with each other. It is used by the virtual display system for:

- Chat and video conferencing feature
- Common writing and drawing area
- Providing the requested files

Users

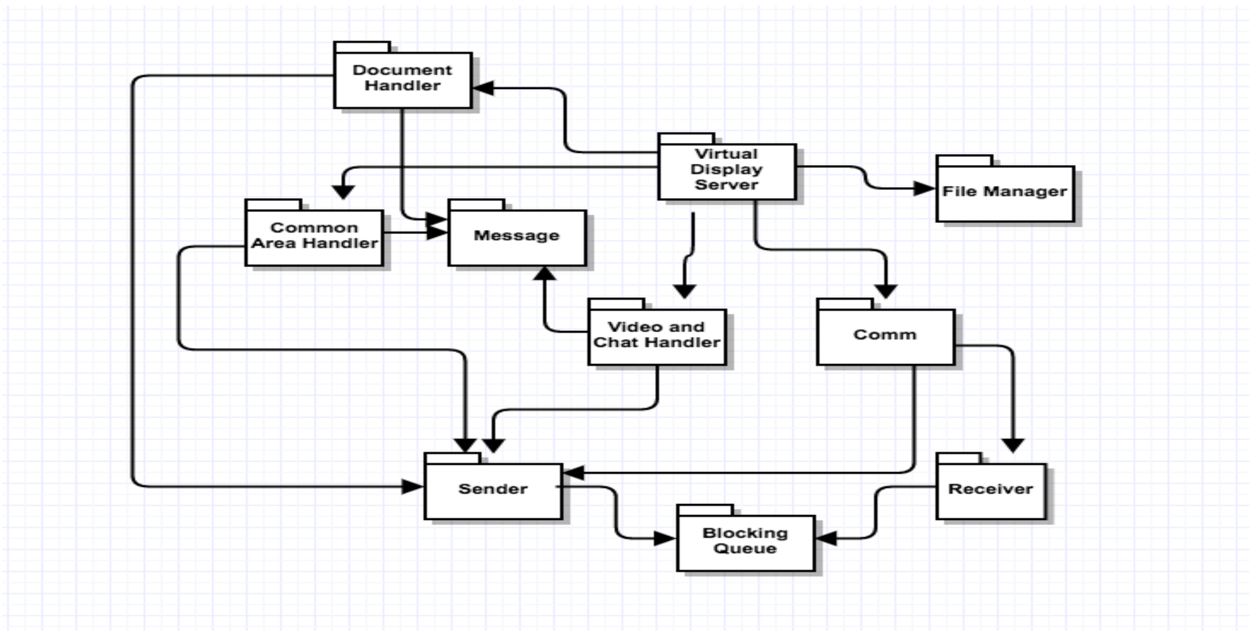
We have discussed about how the users will use the Virtual Display Server in the federation. Here we will discuss about the impact on design they made in test harness.

1) Technical Architect's impact on design
   The job of the Tech. Arch. Is to design the software architecture which requires constant writing and drawing facility during discussion. So we have provided a common writing and drawing area which is common for all users. The tech arch can draw in that area and the rest of the people will be able to see those changes.
2) Project Manager impact on design

Project manager has to constantly update the client with current status of the project. So we had to provide the facility to display the reports on the virtual display. For this we need to provide the facility at the server end to provide these reports

STRUCTURE AND PARTITIONING

We have tried to do the partitioning of the virtual display server based on the responsibility that package will take. Below is the package diagram for the Virtual Display server:



Virtual Display Server

It is the executive package of the Virtual display server. At the start of server, this package creates a channel to listen to incoming messages with the help of the receiver package. Once the listener has started the server waits for incoming messages. Based on the type of the incoming message the virtual display server package passes the control of that job to other packages like document or file related jobs are handled by document handler package, Common area request are handled by the common area manager and chat & video request is handled by its handler package.

Document Handler

The document handler package has been designed to entertain all the file related request of the Virtual display system. When any file related request comes to document handler it takes the message body and creates new message for collaboration server or repository depending upon the type of file and put it the sender package queue.

When incoming file comes, the document handler package sends that file to each active user in that session again with the help of sendFile utility of the sender package.

Common area Handler

The working of this package is what the functionality of google doc. This package receives message with type Request authorization and common area update. This package will maintain a specific number of people who will be authorize to update the common area, and it will maintain that list in a dictionary or a list. The common area either use the concept of image comparison to figure out the changes made by multiple users at same time and then superimpose the image and sent to all the users present in an active session with the help of sender package. It will use the message class object to create message authorizing the user to make updates in common area.

Chat and Video handler

This package has been designed to take care of any chat and video communication happening between the virtual display clients. The package responsibility could be better understood if we see both of these responsibilities individually:

- Chat: The responsibility of managing chat is quite simple. The chat and video handler package will get a message class object with information about the chat example for, to and the chat message. The chat handler package will send the message to an individual client in case it is a personal message or send it to every user apart from the own who has sent in case of the group message. For this it will again create message or messages and put them in the queue of the sender package.
- Video: Handling video input from different users and sending each user the video of all other user is the responsibility of the video handler package. All virtual users will send their live video data using the send Livestream function of the service class of the receiver package. Once the server gets the stream of data it will send it to all other users apart from the open who has send the feed with the help of chat and video handler package. The chat and video handler package will use the utility send video stream of sender package to do this task.

Receiver

The receiver package acts as the listener for the V.D server. This package creates a channel for the V.D server through which all other machines can send messages to it. This package has a receive message blocking queue which is used to store all incoming messages. It also provides a send message function in its service class which is used by other machines to put message into the receive queue. It also provides the facility to download a file in the V.D server, this facility is used in the process of get code files, documents and report from repository or collaboration server. The receiver package has the service class in it which is the implementation of the IService contract used for communication over WCF.

Sender

This package provides the medium for V.D server to send request to other system. In order to send any request to other systems sender package create a proxy object of the machine to which the request has to be sent and then uses functions of other system service class to communicate. Whenever V.D server wants to send a message to other system, it puts a message in sender message queue of the sender. At the start of the V.D server one thread will be continuously running to send message from this sender queue.

Message

The Message package has been developed in order to provide a generic mode of communication between all the machines present in SCF. A message passes over WCF contains following information:

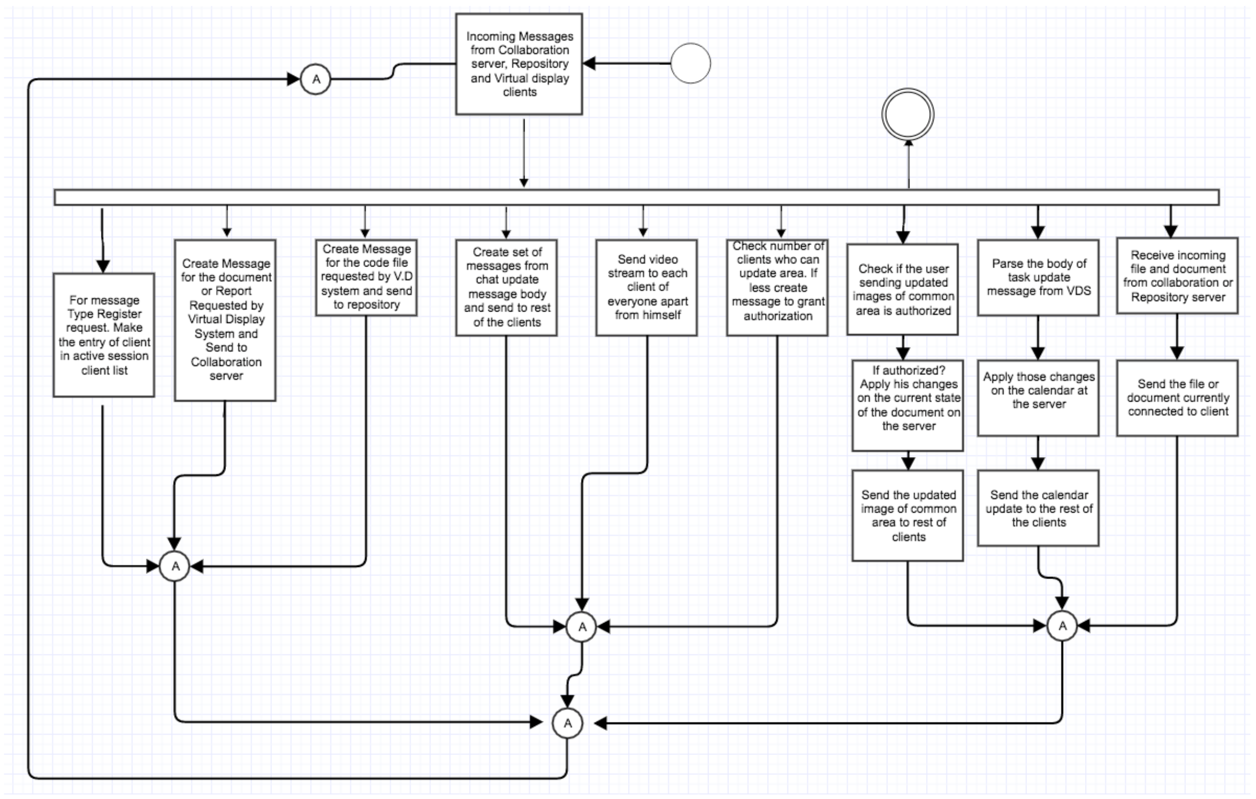
- To: URL of Receiver
- From: URL of Sender
- Author: Name of User
- Type: tells about the type of Message
- Timestamp: For holding date time
- XML Body: To hold operation oriented data.

Comm

The Comm package has been designed to remove the complexity from the executive package. The Comm package takes the responsibility to take care of the communication part of the system. It holds the sender and receiver of the system, so now the executive doesn’t need to hold them. It also keeps the track of the proxy object generated and helps in not creating the same proxy again and again by maintain a record of proxy and URL in a dictionary.

KEY APPLICATION ACTIVITIES

The key activities of the Virtual display system are handling the document request, chat and video request & the common area request. Document request is same as the any file request by any other system in SCF. So we are not going to discuss them in detail. In this part we are going to talk about three activities chat, video and common area management which are done by Virtual display server only. Below is the activity diagram for Virtual display server


Incoming Messages from Collaboration server, Repository and Virtual display clients
A
For message Type Register request. Make the entry of client in active session client list
Create Message for the document or Report Requested by Virtual Display System and Send to Collaboration server
Create Message for the code file requested by V.D system and send to repository
Create set of messages from chat update message body and send to rest of the clients
Send video stream to each client of everyone apart from himself
Check number of clients who can update area. If less create message to grant authorization
Check if the user sending updated images of common area is authorized
If authorized? Apply his changes on the current state of the document on the server
Send the updated image of common area to rest of clients
Parse the body of task update message from VDS
Apply those changes on the calendar at the server
Send the calendar update to the rest of the clients
Receive incoming file and document from collaboration or Repository server
Send the file or document currently connected to client

Chat Service

All the virtual users will have a chat window which will give them the option to chat at group level as well as individual level. On group level the message will go to each and every user present in current session. When the user sends a message with body below to the server:

```
 <ChatUpdate>
 <to>Group</to >
 <from>shishir</from>
 <Timestamp>12-12-16</ Timestamp >
<Message>Database connection issue: Please use complete name</Message >
</ ChatUpdate >
```

The chat handler package parses the body to know whom the message has been sent for. Once it has the information about the user or the group for which the message has been sent. Chat handler package will take the same message and change the “To” class variable and send it a single user or multiple user by changing “To” every time.

Video Service

When each virtual user starts they send a registration request to the server. The server makes the list of active user for the session from those registration request. The Video handler package uses this list to know which all users he needs to send the videos. Whenever the virtual display server receives a video stream it takes that stream and divert it to all the other users present in the current session apart from the one from whom it has been sent. This way every user will have live video feeds from every other user in the session attending the meeting. The send Livestream function of the sender class will help the video handler package to do the task.

Common Area Management

It will be very difficult to allow every user in the meeting the authority to write and design things in the common area. As it will take time to reflect the changes made by so many users at the same time. So the common area handler will allow limited number of users to make changes in the common area. A message will come to server for authorization and it will authorize the user.

The other task of the Common area manager is to how to display the changes made by different user at same time. The easiest solution to this problem is superimpose the images sent by different users and then sent the image to all the users present in the active session.

Calendar task Management

The virtual display server package will handle the task of managing the calendar updates sent by the user. A calendar message for update will look like below

```
Type: calendar_update
Body:<CalendarUpdate>
      <Date>12-12-16</Date >
      <FromTime>10:00 am</ FromTime >
      <ToTime>12:00 pm</ ToTime >
     <Task>Discuss requirement 16th</Task >
     </ CalendarUpdate >
```

Once the server receives a message for calendar update it changes the “To” of the message class object and sends to every user apart from the one who has sent it.

Critical Issues

1. **Delay in reflecting common area updates**: Suppose we have 20 virtual clients in a session and all of them try to draw stuff in the common area. In that case as server is getting so many updates at same time it won't be able to send the updates images to other users simultaneously.
   Solution: Fix the maximum number of people who can make changes in the common area to a number such as 4. So server won't get loaded and there is no delay in updating other users

Messages

The Messages have been developed in order to provide a generic mode of communication between all the machines present in SCF. Below we have talked of various types of messages that are being used in SCF for getting the task done:

1) **File Transfer Message**
   The file transfer messages are used for getting the code files or documents over the WCF channel. Based on the type i.e. File_Tranfer_Request the receiving machine will come to know that this is a file transfer request and it will use the message body to get the file names.

   To: <URL of Repository>
   From: <URL of Sender>
   Author: Shishir Bijalwan
   Timestamp: 12/14/2016
   Type: File_Transfer_Request
   Body: <Files>
   <File>CodeToBeTested.cs</File>
   <File>Game.cs</File>
   </File>

2) **Test Request**

   In order to use the services of the test harness the client sends these type of messages over the WCF channel. Once the Test harness receives this message it come to know it has to run test. It uses the body of message class to get information about the files required to run the test.

To: <URL of TestHarness>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Test_Request
Body: <Test>
 <TestDriver>TestDriverOne.dll</ TestDriver >
 <DependentFiles> CodeToBeTested.dll</ DependentFiles >
 <DependentFiles> CodeToBeTested2.dll</ DependentFiles >
 </Test>

3) Log, Task, File, Report, Issue or document List
In order to request the logs from the repository the client needs to know the log file name. To make the task easier we have designed a message called log_list which helps the user to get the names of log file related to key words. These key words can be passed in the GUI while requesting for list. Similarly, we can request Repository for file list using **file_list** and request task list or document or report list from collaboration server using type **task_list, document_list**, **Issue_list** and **report_list.**

To: <URL of Receiver>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: log_list
Body: <KeyWords>
 <KeyWord>TestDriver </ KeyWord >
 < KeyWord > CodeToBeTested </ KeyWord >
 </ KeyWords >

4) Get log data
Once the user knows the name of the log file he can select that and request the repository to send the log data to him/her.

To: <URL of Repository>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: log_data
Body: <LogFile>
 Shishir_12_May_2015.xml
 </ LogFile >

5) **Test Result**

This type of message is used by the Test Harness as well as the Repository to send the result and the log for a particular test. In our case we send the complete log plus results to the user and give the liberty to the GUI to display just results or logs too.

```
To: <URL of Client>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Test_Result
Body: <TestResult>
        <Author>Shishir Bijalwan</Author>
        <Timestamp>20140112180244 </ Timestamp >
        <TestDriver>TestDriverOne.dll</ TestDriver >
        <DependentFiles> CodeToBeTested.dll</ DependentFiles >
        <DependentFiles> CodeToBeTested2.dll</ DependentFiles >
        <Passed>True</Passed>
        <Logs>Null pointer exception</Logs>
        </ TestResult >
```

6) **Task Message**

The collaboration tool provides the facility to monitor updates of each task. The user can get the details of work done on any task using this message from the collaboration server. Every task will have an id associated with it to make the name.

```
To: <URL of Collaboration server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Task_Request
Body: <TaskRequest>
        <TaskName>Login_addition_1001</ TaskName >
        </ TaskRequest >
```

7) **Task information**

Once the client has requested the collaboration server for the information or updates regarding a particular task. The collaboration server gives back the information in form of below message.

To: <URL of Client>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Task_Details
Body: <TaskDetails>

```
          <TaskName>Login_addition_1001</ TaskName >
           <Assignor>Shishir</ Assignor >
          <Assignee>Rahul</Assignee >
          <TaskDescription>This task is to add login facility</ TaskDescription >
          <TaskUpdate>
           <Timestamp>20140112180244</Timestamp >
           <UpdateDoneBy>Rahul< UpdateDoneBy >
            Created front End.
           </ TaskUpdate >
          <TaskUpdate>
           <Timestamp>20140122180244</Timestamp >
           <UpdateDoneBy>Shishir< UpdateDoneBy >
            Rahul please prioritize working on the task.
           </ TaskUpdate>
           </ TaskDetails >
```

8) Issue detail Request

The collaboration server keeps the record of all the issues faced by the developer and help other developer to get the solution for that issue. The user gets the list of issues from collaboration server using the Issue_list type of messages. Once the user has the list he/she can use it to request for the issue details
To: <URL of Collaboration server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Issue_Request
Body: <IsssueRequest>

```
          <IssueName>Login_addition_1001</ IssueName >
          </ IsssueRequest >
```

9) Issue Details

Once the collaboration server gets the request to give the details of an issue. It uses the Issue_Details message to provide the information regarding that issue.

To: <URL of Client>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Issue_Details
Body: <IssueDetails>
    <IssueName>user not found</ IssueName >
    <Originator>Shishir</ Originator >
    <Solution>Database connection issue: Please use complete name</ Solution >
    </ IssueDetails >

10) Creating new issue

Whenever the developer gets a new issue it needs to inform the collaboration server about it. It uses the below message to inform the collaboration server about the new issue.

To: <URL of Collaboration server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: New_Issue
Body: <IssueDetails>
    <IssueName>user not found</ IssueName >
    <Originator>Shishir</ Originator >
    <Solution>Database connection issue: Please use complete name</ Solution >
    </ IssueDetails >

11) Creating new Task

The Tech Architect/Social Tracker/Lead will need to create task for the developer. This type of message is used to create a new Task in the collaboration server.
To: <URL of Collaboration server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: New_Task
Body: <TaskDetails>
    <TaskName>Login addition</ TaskName >
    <Assignor>Shishir</ Assignor >
    <Assignee>Rahul</Assignee >
    <TaskDescription>This task is to add login facility</ TaskDescription >

</ TaskDetails >

12) Chat update Message

This type of message is used by the virtual display system to communicate the chat message with other user present in the meeting. The V.D. system sends this message to the V.D. server.

To: <URL of V.D. server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Chat_Update
Body: <ChatUpdate>
<to>Group</to >
<from>shishir</from>
<Timestamp>12-12-16</ Timestamp >
<Message>Database connection issue: Please use complete name</Message >
</ ChatUpdate >

13) Common Area Authorization request

The Virtual display server can allow finite number of people to have access to write in common area. So the user must request the Virtual display server informing that he will write and draw in the common area. A user won't be able to send request until a spot is free in our current design so virtual client doesn't need to wait for the reply.

To: <URL of V.D. server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: common_area_authorization
Body: <commonAreaAuthorization>
<requester>URL</requester >
<name>shishir</name>
</ commonAreaAuthorization >

14) Registration Request

The Virtual Display server must know all the users who are attending a meeting. So every user who wants to connect to the Virtual display server first needs to send a registration request.

To: <URL of V.D. server>

From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: Registeration_Request

```
Body:   <Register>
            <requester>URL</requester >
             <name>shishir</name>
           </ Register >
```

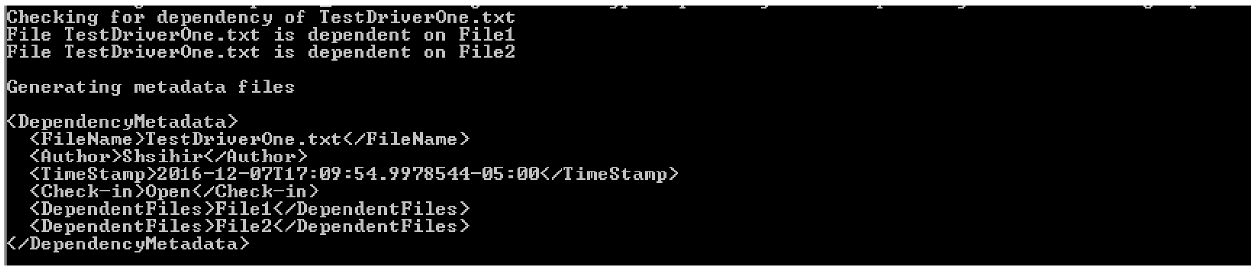
15) Calendar update

In the virtual display screen, we have given the option to set the task for a meeting or to book time slot for a task. The user can click on a date and set the task in the popup. Once user set the task on pop up. Below message will go to the server who will circulate this information to other users.

To: <URL of V.D. server>
From: <URL of Sender>
Author: Shishir Bijalwan
Timestamp: 12/14/2016
Type: calendar_update

```
  Body:<CalendarUpdate>
            <Date>12-12-16</Date >
            <FromTime>10:00 am</ FromTime >
            <ToTime>12:00 pm</ ToTime >
           <Task>Discuss requirement 16th</Task >
           </ CalendarUpdate >
```

Prototype 1

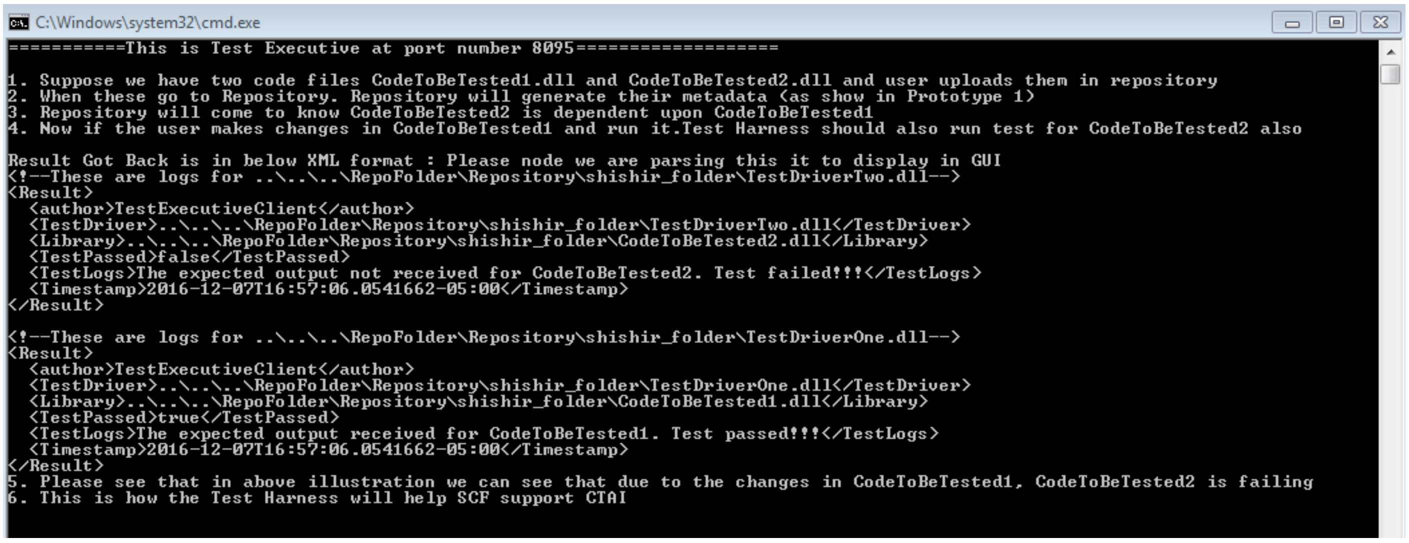


```
Checking for dependency of TestDriverOne.txt
File TestDriverOne.txt is dependent on File1
File TestDriverOne.txt is dependent on File2

Generating metadata files

<DependencyMetadata>
  <FileName>TestDriverOne.txt</FileName>
  <Author>Shsihir</Author>
  <TimeStamp>2016-12-07T17:09:54.9978544-05:00</TimeStamp>
  <Check-in>Open</Check-in>
  <DependentFiles>File1</DependentFiles>
  <DependentFiles>File2</DependentFiles>
</DependencyMetadata>
```

In this prototype we have demonstrated how the repository will get the dependency list and use it to create a metadata file. Every time a code is uploaded into the Repository. The repository will make type table entries using it. A type table entry will have type name and the file name in which that type is found.

Now any time any file is uploaded. The repository will check the file against this type table and come to know on which all files it depends. Hence using that information it will generate the metadata file as displayed in the image.

Prototype 2



C:\Windows\system32\cmd.exe

```
===========This is Test Executive at port number 8095===================

1. Suppose we have two code files CodeToBeTested1.dll and CodeToBeTested2.dll and user uploads them in repository
2. When these go to Repository. Repository will generate their metadata (as show in Prototype 1)
3. Repository will come to know CodeToBeTested2 is dependent upon CodeToBeTested1
4. Now if the user makes changes in CodeToBeTested1 and run it.Test Harness should also run test for CodeToBeTested2 also

Result Got Back is in below XML format : Please node we are parsing this it to display in GUI
<!--These are logs for ..\..\..\RepoFolder\Repository\shishir_folder\TestDriverTwo.dll-->
<Result>
  <author>TestExecutiveClient</author>
  <TestDriver>..\..\..\RepoFolder\Repository\shishir_folder\TestDriverTwo.dll</TestDriver>
  <Library>..\..\..\RepoFolder\Repository\shishir_folder\CodeToBeTested2.dll</Library>
  <TestPassed>false</TestPassed>
  <TestLogs>The expected output not received for CodeToBeTested2. Test failed!!!</TestLogs>
  <Timestamp>2016-12-07T16:57:06.0541662-05:00</Timestamp>
</Result>

<!--These are logs for ..\..\..\RepoFolder\Repository\shishir_folder\TestDriverOne.dll-->
<Result>
  <author>TestExecutiveClient</author>
  <TestDriver>..\..\..\RepoFolder\Repository\shishir_folder\TestDriverOne.dll</TestDriver>
  <Library>..\..\..\RepoFolder\Repository\shishir_folder\CodeToBeTested1.dll</Library>
  <TestPassed>true</TestPassed>
  <TestLogs>The expected output received for CodeToBeTested1. Test passed!!!</TestLogs>
  <Timestamp>2016-12-07T16:57:06.0541662-05:00</Timestamp>
</Result>
5. Please see that in above illustration we can see that due to the changes in CodeToBeTested1, CodeToBeTested2 is failing
6. This is how the Test Harness will help SCF support CTAI
```

In prototype two I have demonstrated how the test harness will help the SCF in doing continuous testing and integration. I have used sequence of steps to explain it and used my project 4 of SMA to illustrate it.

References

1. Wikipedia https://www.wikipedia.org/

2. Test Automation: A Project Management Perspective http://www.uploads.pnsqc.org/2013/papers/t-035_Pulla_paper.pdf

3. TESTING FRAMEWORKS http://www.cs.colorado.edu/~kena/classes/5828/s12/presentations/testing-frameworks-by-gayat.html

4. Class Notes SMA CSE-681.