CSE 681: SOFTWARE MODELING & ANALYSIS
PROJECT # 3

"REMOTE TEST CONFIGURE TOOL"

OPERATIONAL CONCEPT DOCUMENT

VERSION 1.0

July 4, 2007
Syracuse University

-- Aniruddha Gore
SUID#909876627

Table of Contents

Table of Figures

1. Executive Summary

‘Remote Test Configure Tool’ is a very useful tool in the development process of a large software project spread over multiple locations in an organization. It facilitates remote evaluation of a large project (or a part thereof) by configuring tests on a remote test server and retrieving test results after test execution is completed.

Remote Test Configure Tool will work in an environment with a few members viz. *User* (for input/output), *Remote Terminal File System* Services, *Build Processor*, *Test Harness* and *Network* Resources (for communication). Modular architecture of the tool is designed with concept of layers in mind: *Presentation, Data Storage, Communications* and *Processing*. The tool is developed as ‘Client’ Module and ‘Server’ Module; major part of both these modules is actually a reusable structure composed of:

- XML Encoder: encodes message/command in XML format using custom XML tags
- XML Decoder: extracts information from a XML based message string
- Message Mailbox: ensures safe transmission of each message to receiver
- Command/Message Handler: interprets extracted information and act upon it
- Send Queue: stores a small set of messages to be sent
- Communications: transmits data from sender to receiver

‘Message Mailbox’ and ‘Send Queue’ modules are main members of ‘Data Storage’ layer and plays very important role as working of tool involves extensive message passing.

Principal users of *Test Harness* will be software professionals but not excluding non-technical end users and users from academia. All possible interactions between the user and application are facilitated by the proposed interface, of which two cases namely ‘Add to Suite’ and ‘Abort Tests’ may be difficult to implement as they may be requested while application is already running.

Ten important issues are identified and potential solutions are suggested. Of these ten issues, three are found to be *critical*; implementation decisions are concluded for each issue which will help make *Remote Test Configure Tool* an efficient application. The three critical issues are:

- Tool Performance under heavy loads
- Ensuring safety and integrity of messages being transferred across network
- Overwriting of system files on test server

Summary of conclusions of all important aspects bolsters successful implementation of *Remote Test Configure Tool* within a considerably short period of time and with available resources.

2. Introduction

'Testing' phase is considered to be an important phase in Software Development Life Cycle due to following reasons:

a) It is the phase where value delivery to customer is ensured
b) Getting involved into testing at earlier stages of development reduces the risk of getting into critical problems at the end, thereby cost benefiting since cost of fixing a defect is more at the later stages

Applications like *'Test Harness'* facilitates such testing needs but with increasing growth of (software) project sizes, customer requirements and expected application accuracy, large software projects are being developed by different groups of developers and architects. Depending on availability of skills & resources and time constraints these groups are often alienated and thus have to coordinate with each other remotely to meet with customer specifications. *'Remote Test Configure Tool'* comes to rescue at the *testing* front in such situations so that software tests can be configured, scheduled and performed remotely independent of from where they are being invoked and where the test files/libraries are located.

Figure 1 below describes basic activities performed by the *Remote Test Configure Tool*. The purpose of this activity diagram is to give the reader an abstract idea of what activities the tool will perform at the client's end.

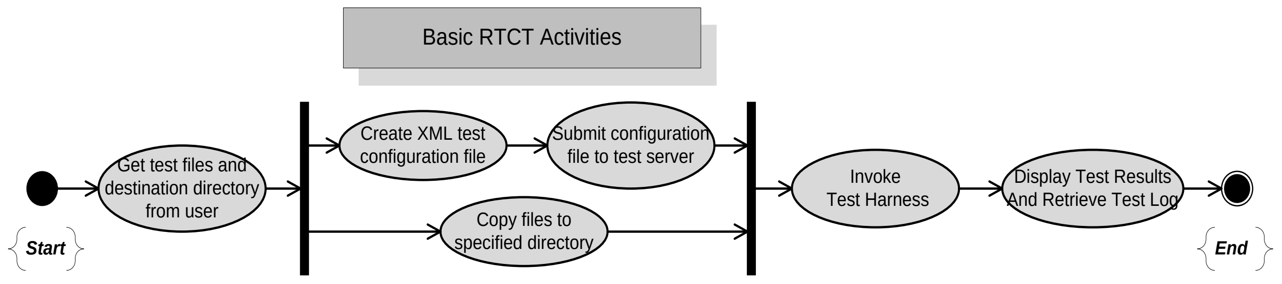

Basic RTCT Activities
Start
Get test files and destination directory from user
Create XML test configuration file
Submit configuration file to test server
Copy files to specified directory
Invoke Test Harness
Display Test Results And Retrieve Test Log
End

Figure 1: Basic RTCT Activities

As shown in the figure, tool starts with invoking a user interface prompting user to select test files. It is to be noted here that user will be able to select the test files by browsing his/her local client terminal, a remote client terminal or the test server. *Remote Test Configure Tool* then copies all the specified files to the directory and creates an XML file with these details (path to test directory and list of test libraries) which is then submitted to the test server. Creation and submission of XML file is shown between same pair of synchronizing bars as copying of test files to server, to indicate that these two activities may take place in any order or in parallel. Test server selects one test configuration at a time (from many of them submitted by the clients on the network) based on a specific priority mechanism and invokes *Test Harness* to perform tests on that selected configuration. *Test* Harness is invoked only after the configuration file has been submitted to test server and all the test libraries have been copied to destination directory on server. After test server has performed all the tests, test results are displayed to user and test result log is retrieved as its last activity before terminating.

3. Project Obligations

'*Remote Test Configure Tool'* will be cultivated as an efficient and user friendly tool to support remote configuration of tests for modules (which are part of a large project being developed) concomitantly as they are developed.

The main specifications laid for this project are to:

- Establish communication between CONFIGURE program running on all terminals of the local network using either .NET Sockets or .NET Remoting
- Provide a *Graphical User Interface* to browse test files on local and remote terminals
- Build any selected source code into libraries
- Create a XML based test configuration files and submit it to the test server
- Retrieve test result log from server

4. Remote Test Configuration Tool (RTCT) Deployment

Section 2 (Introduction) discussed basic activities of RTCT. This section describes how RTCT will be deployed on machines so that it may be used in an actual network setting for test configuration purposes. Figure 2 below depicts a simple block diagram showing deployment of RTCT in a typical local network containing three clients viz. *Terminal # 1, Terminal # 2, Terminal # 3* and a test server viz. *Server*. All clients will be connected to the test server through a communication channel which may be implemented using .NET Sockets or .NET Remoting. The three terminals will also be connected to each other but it's not shown here for the sake of clarity.

RTCT will be installed on each of the four machines with an additional mechanism installed on the test server along with RTCT. This additional add-on is required at the test server because eventually it is going to be the test server which will start *Test Harness* and log results. As mentioned earlier, clients may also communicate with each other and thus reflect a client-server behavior while in communication. It is always convenient to express such interactions with the client-server terminology but in this case the term 'server' is very likely to be confused with test server on which *Test Harness* resides. To avoid such situations of ambiguity, a terminology is adopted which will be used in the rest of the document:

Terminology: - This terminology is adopted from 'Digital Electronics' where 'Master-Slave' flip-flops are so called because of the difference in direction of control/data flow.

- **Server RTCT:** RTCT installed at the test server where tests will be performed
- **Master RTCT:** RTCT installed at the client which sends a message/command to other client or the *Server RTCT*
- **Slave RTCT:** RTCT installed at the client which receives message/command from either a *Master RTCT* or the *Server RTCT*

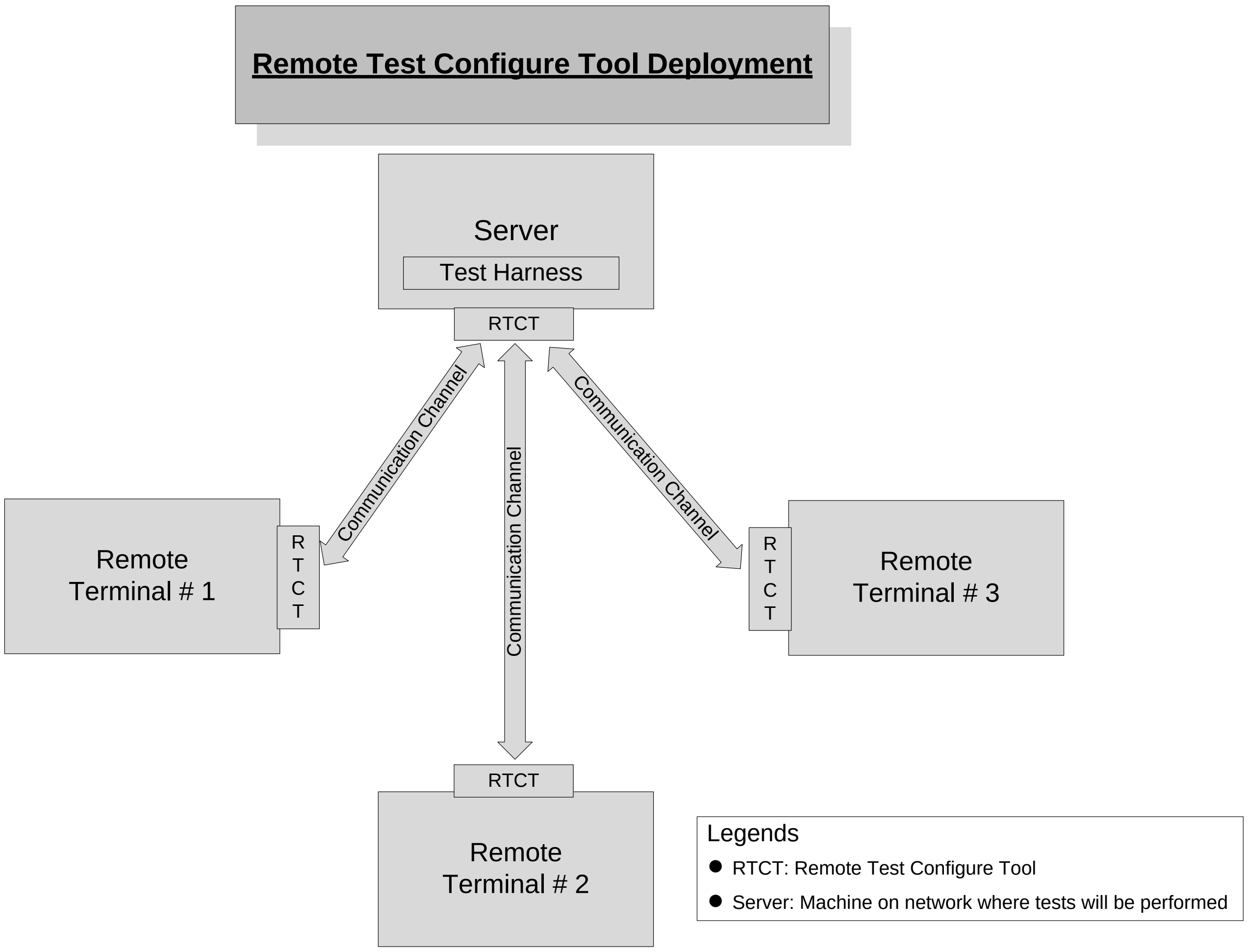


Figure 2: Block Diagram Showing RTCT Deployment

What follows explains how a user can perform tests using RTCT. Suppose a user is on *Terminal # 1* and wants to perform tests which involve files on *Terminal # 2.* Now the user will start the RTCT installed at *Terminal # 1* and will browse local test files and files at some remote terminal say *Terminal # 2*. After selecting test files, RTCT at *Terminal # 1* may find some source code on *Terminal # 2* which is selected and needs to be built into library. Thus it instructs *Terminal # 2* to do so. In this case *Terminal # 1* act as *Master RTCT* and *Terminal # 2* acts as *Slave RTCT*. After all files are built into libraries, *Master RTCT* will create a XML file with all test configuration details required by the *Server RTCT*. *Master* and *Slave RTCTs* will copy files to the test directory on server. Test server now picks up this XML configuration file when it finds it appropriate to execute tests. *Server RTCT* then starts *Test Harness* service with the add-on provided to it (as mentioned above) and initiates tests. After all tests complete, results are displayed to *Master RTCT* GUI and *Master RTCT* can also retrieve the test result log. It should be noted that *Server RTCT* can very well become a *Master RTCT* for a test when Server Administrator (*user* in this case) starts this tool at the test server.

Conclusion: *RTCT* must be installed on all machines which are to be the part of testing system. GUI should provide local and remote browsing facilities to user.

5. Top Level Architecture

Context of *Remote Test Configure Tool (RTCT)* will depend a lot on whether the *RTCT* is a *Server RTCT*, *Master RTCT* or a *Slave RTCT;* however the discussion here revolves around context in which it is used by a user. Figure 3 below depicts context[¥] of *RTCT* in a typical setting wherein user is using the *RTCT* from a *Master RTCT* and configures test files located at both *Master RTCT* and a *Slave RTCT,* to be tested at the *Server RTCT*.

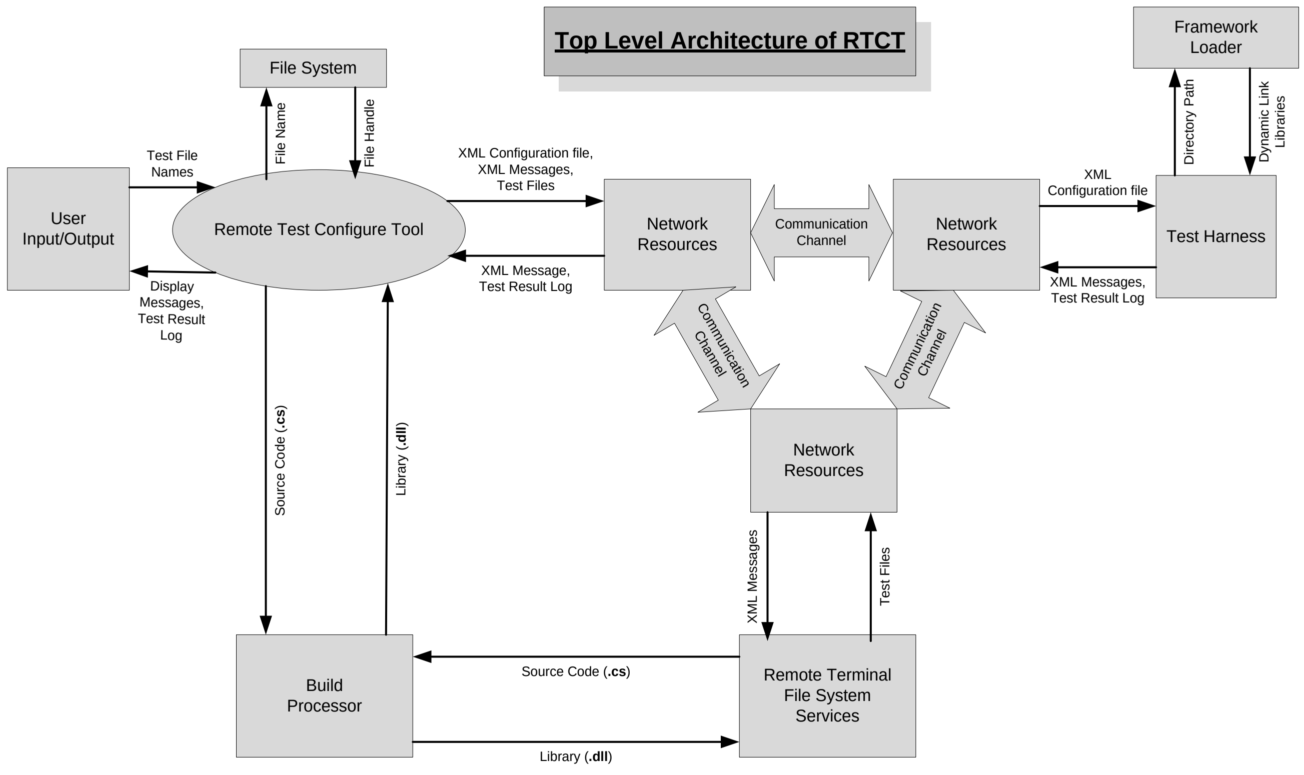


Figure 3: Context Diagram of Remote Test Configure Tool

The context comprises of mainly seven elements which are described below:

5.1 User Input/Output

User initiates test configuration by interacting with a *Graphical User Interface* provided by *RTCT.* User browses test files located across the network and confirm this files list before submitting it to *RTCT*. The *RTCT* keeps the user informed of status of tests he/she submitted and display progress/error messages during the tests and test result summary at the end.

¥: Although a *Context Diagram* has to do with interaction of code being developed with resources in its environment, 'Test Harness' and 'Remote Terminal File System Services' are actually processes but have been included here to convey very clearly how *RTCT* interacts with its surroundings. Sticking to the UML conventions, these two processes are enclosed within a block (and not in a 'bubble') to focus the discussion around *Remote Test Configure Tool*.

5.2 Network Resources

Being capable of configuring tests remotely, *Network Resources* are vital to *RTCT* in the process of test configuration. *Network Resources* plays the main role of facilitating communication between *Master RTCT, Slave RTCT and the Server RTCT*. When the user at a *Master RTCT* intends to browse a *Slave RTCT* (i.e. a remote terminal), the *Master RTCT* injects a message into *Network Resources* which it conveys to requested *Slave*. It then conveys remote browsing information to requesting *Master RTCT*. *Network Resources* are also responsible for copying/transfer of XML Test Configuration/XML Message files across the network; these file transactions mainly include:

- Submitting XML Test Configuration file created at *Master RTCT* to Test Server
- Copying test files from various terminals across network to Test Server
- Retrieving Test Result Logs from Test Server and copying it to *Master RTCT*
- Transfer XML Message files between *Master, Slave* and *Server*

5.3 Build Processor

Not all modules are built as libraries but *Test Harness* residing at the test server can load only *Dynamic Link Libraries (DLLs)* and thus the test files submitted to the server must all be DLLs. To relieve code developers (and other users) of this overhead of keeping in mind of compiling each module in as a library, *Remote Test Configure Tool* makes use of *Build Processor.* Whenever a source code is found among the test files in a configuration, it is submitted to *Build Processor* which builds it into library and returns back to the test file bank.

5.4 Remote Terminal File System Services

Remote Terminal File System Services are responsible for communicating with *Master RTCT* and *Server RTCT* and take appropriate action. A user is very likely, in a large software environment, to browse some remote terminal for test files. As a consequence, files located at this remote machine will be copied to test server and might require being built into libraries. All such communications are done by *Master RTCT* and *Server RTCT* through XML Message passing to this particular *Slave RTCT. Remote Terminal File System Services* are responsible for this communication as they will be required to receive these XML Message files, decode them and take actions which were requested in the messages.

5.5 Test Harness

Eventually it's *Test Harness* which utilizes work done by *Remote Test Configure Tool,* performs tests and returns the results back to *Master RTCT* through *Network Resources. Test Harness* gets directory path where all test files are stored from the XML Test Configuration file submitted by the many *Master RTCTs.* Which configuration to load next depends on *Server RTCT,* which decides this based on some priority mechanism. *Test Harness* utilizes services provided by *Framework Loader* (not discussed here as *RTCT* does not directly communicate with it) to load test libraries. After all test executions complete, results are logged to a text file.

5.6 File System

File System facilitates services for dealing with files, which is an obligatory task of *Remote Test Configure Tool.* A *Master RTCT* uses File System to create a XML Test Configuration File with all test configuration details in it to be submitted to test server. Similarly it may receive XML Messages from Server or *Slave RTCT* which it will have to decode and act upon.

5.7 Remote Test Configure Tool

Remote Test Configure Tool (RTCT) represents all the code being developed. It is the main workstation which configures the tests on test server. *RTCT* presents the user with a *Graphical User Interface (GUI)* when the tool is started. It communicates with *Remote Terminal File System Services* through *Network Resources* by XML Message passing if a user happens to browse remote machines. Also *RTCT* sends XML Configuration File to test server and receives, decodes and acts upon messages received from server (if any). While in progress for configuring a test on server *RTCT* is also (conditionally) required to interact with *Build Processor* for purposes described earlier.

<u>Conclusion:</u> *Remote Test Configure Tool* can be implemented as a physical unit which accomplishes test configuration by simple data transfers with its environment consisting of seven members. The *RTCT* receives required inputs from *User Input/Output* so it must provide a *GUI* which can be easily used by all of its users*;* builds XML Test Configuration file with the help of *File System, Build Processor and Remote Terminal File System Services* and submits test configuration to the server through *Network Resources*. *RTCT* is required to communicate with other members from time-to-time so the communication mechanism built into it must have a simple and flexible structure capable of performing efficient data transfers.

6. Data Flow in RTCT

This section attempts to explain, at a high abstraction level, as to how the data flows inside *Remote Test Configure Tool*. In accomplishing its task of remotely configuring tests on a test server, the *RTCT* has to exchange data with various other members in its surroundings (as explained in previous section) but how this data is processed inside the structure of *RTCT* can be best depicted by means of a top level *Data Flow Diagram*, as shown below in Figure 4.

As shown in diagram, *RTCT* can be thought of producing desired results by co-operation of six processes. Each process either takes some data from an outer resource or from another process and produces some data which may be used by yet another process or is the output of *RTCT*. In the paragraphs to follow each of these processes is briefly described.

6.1 Process 1: Build Libraries

After user browses and selects all test files of interest, the first process *Build Libraries* starts. In this process, each selected file is checked for whether it is a source code (and needed to be built into library) or it is a Dynamic Linked Library. In case of former, all the source codes are built into one or more DLLs with proper display messages shown on *GUI* to the user. After this process is completed, a data structure *LibStore* is created to store names of all test libraries.

6.2 Process 2: Create XML Configuration File

Create XML Configuration File process involves creating a XML file listing names of all test libraries stored in *LibStore* data structure. This process also decides the name of destination directory on server to copy all libraries into and appends the XML file with this information. In order to do so, this process needs to make use of the *File System* services so that it gets hold of the *file handle* for manipulation of file with that *file name*.

6.3 Process 3: Copy Files to Server

After creation of configuration file, libraries listed therein are to be copied to the test server. This process starts with sending of XML based messages to test server of getting ready for receiving test files. Test files (libraries) are then transferred one by one and corresponding display messages are shown to the user.

6.4 Process 4: Submit Configuration File to Test Server

After all test files are copied on server, the XML configuration file created in process 2 (TestConfig.xml) will be submitted to server to schedule the tests. This process ends with sending out a message to server that a configure file has been added to its pool.

6.5 Process 5: Invoke Test Harness

After *RTCT* has completed its task of configuring tests and submitting details to the server, it will have to wait for the *Test Harness* process to execute all tests. Thus now *Test Harness* is invoked and as it progresses, proper test messages are display to user through GUI.

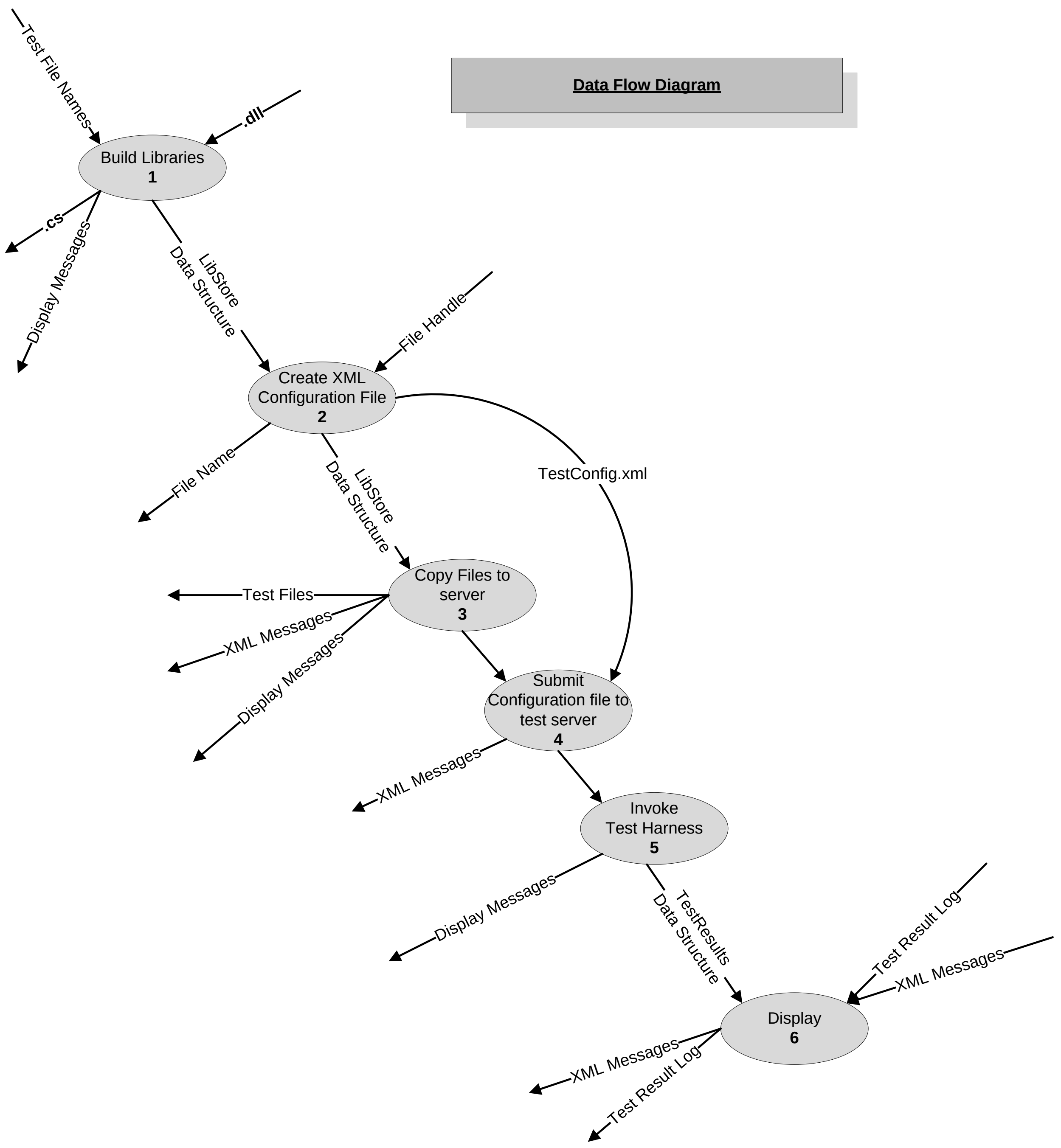


Figure 4: Top Level Data Flow Diagram for Remote Test Configure Tool

6.6 Process 6: Display

This is the last process as tests culminate at this stage. *Test Harness* sends a message to *RTCT* intimating test completion. Along with this *Test Result Log* is also sent to the *Master RTCT* (i.e. *RTCT* from where user started tool). Summarized test results are displayed to user and the *Test Result Log* is also supplied to the user.

Conclusion: As shown in Figure 4 and described in previous sub-sections, *Remote Test Configure Tool* mainly involves transferring files from and to *Master RTCT*. Thus considering pivotal role of XML Configuration file and undoubtedly the test files in testing process, *RTCT* must implement a secure and efficient communication and message passing mechanism for reliable data transfers. Among constituting processes of *RTCT* the main data flow is of data structures which are used to store test file/library names (TestLib DS) and results of individual tests (TstResults DS).

7. Use Cases

This section attempts to identify principle users of the *Remote Test Configure Tool*, ways in which they will interact with the tool and what features will be incorporated to suffice needs of all users.

7.1 Principal Users

Main users of this tool can be categorized into three groups:

- Software professionals
- Non-technical end users
- Users from academia

In the sections to follow is described who all belong to each of above category of users and what they expect from *Remote Test Configure Tool.*

7.1.1 Software Professionals

This category comprises of most of the users of this tool and it is this category which will use the tool most frequently than any other category.

- **Software Architects**

A *Software Architect* is a person, who is responsible for gathering user requirements to lay down project specifications and divide project into simpler modules keeping in mind available infrastructural, economic and human resources. Often implementation of logic deviates from what was set in its concept document due to technical constraints. It is the responsibility of a *Software Architect* to verify whether the progress is in a direction meeting specifications or not. When, in case of large projects, different modules of the projects are being developed at distant places, *Remote Test Configure Tool* will prove to be very efficient for such purposes.

<u>Need:</u> *Remote Test Configure Tool* will prove to be beneficial to an *Architect*:

a) Continuous track of whether project specifications are met
b) Identification of probable architectural faults if a module fails
c) Immediate communication with *Program Manager/Team Leader* on a failure and prompt action for an alternative architecture

<u>Conclusion:</u> With the progress of a project an *Architect* very frequently hops (from his terminal) between testing modules developed by different groups thus *RTCT* should provide easy browsing of remote terminals and convenient features to add/edit test suites so that an *Architect* may frequently configure tests located at remote locations. Also a history of recently tested files should be maintained.

- **Program Manager/Team Leader**

A Team Leader is a person who leads a group of software professional to achieve a certain goal in Software Development Life Cycle. Many such *Team Leaders* work in co-ordination to meet specifications of a project and are in turn directed by a *Project Manager*. Thus it comes on the shoulders of *Program Managers* and *Team Leaders* to complete the project with in stipulated time and with an effort to deliver utmost accuracy and quality to the customer. In an effort to do so, tests are frequently performed on different modules being worked on. A *Remote Test Configure Tool* will be very helpful for these purposes and in case of *Project Managers* this becomes even more useful because they have to assess the work going on in a number of different teams under him/her. Again for this class of users frequent testing is a must to ensure proper progress of the project.

<u>Need:</u> *Project Manager* and *Team Leader* play very crucial role in software development and use of *Remote Test Configure Tool* will have following impact in the process:

a) Immediate discussions with teams/members if a module fails
b) Assessment of accuracy and speed of teams
c) Prompt changes in design/architecture/functionalities if so required

<u>Conclusion:</u> *Remote Test Configure Tool* again needs to provide a user friendly interface so that different remote test files may be browsed easily and in addition now the interface should show a brief detail of result so that *Program Manager* know which *Team Leader* to contact and a *Team Leader* knows which group to contact.

- **Code Developer**

Eventually it comes to a *Code Developer* to transform code logic into reality. Writing an efficient code under numerous time and resource constraints needs continuous testing of code being written so that codes depending on it will not be affected due to it. Dependent codes may possibly be developed by another developer sitting at a remote terminal and this is where *Remote Test Configure Tool* comes into play.

<u>Need:</u> *Remote Test Configure Tool* when utilized for code development process will definitely speed up the process as it will help in:

a) Successful integration of modules being developed with already existing modules
b) Discussion with *Project Manager, Software Architect* or *Team Leader* to change design/architecture if present concept is practically not viable
c) Modifying module designs to keep code complexity within allowed levels
d) Avoiding failures at later stages

Conclusion: A *Code Developer* works at the finest details of the project and thus needs to know all possible details of a failure, if one happens, and thus the interface will furnish details of tests if a *Developer* chooses to view. Moreover *RTCT* should keep a brief history of recent files/modules/suites tested so that this user will save time testing same set repeatedly.

- **Quality Assurance/Testing**

Quality Assurance team is responsible for maintaining industry standards of the code being developed whereas *Testing teams* are responsible for verifying whether a code is capable of performing the satisfactory under all circumstances and in all environments. A person can never completely test/verify a code developed by him/her and this is the reason why these teams are almost always an imperative part of the industry.

Need: *Quality Assurance* and *Testing* teams perform some of the final tasks before product deliverance to the customer and thus *Remote Test Configure Tool* will be helpful in valuable returns in terms of time:

a) While modules are under development, recursive testing of modules is performed to check that modules being developed and already existing modules are compatible with each other
b) Immediate notification to concerned teams of any failures
c) Discussion with *Team Leaders* of maintaining industry standards
d) Intimate *Architects* and *Code Developers* if desired performance is not obtained

Conclusion: *Testing/QA* teams should concentrate more on which types of tests to perform than how to configure and perform the tests. Thus *Remote Test Configure Tool* will provide very simple *user interface* so that test configuration and execution now becomes responsibility of *Remote Test Configure Tool*. Keeping track of recently tested files/modules/suites is even most for this class of users as they are only concerned with exhaustive testing of code being written.

7.1.2 Non-Technical End Users

The *Remote Test Configure Tool* will mostly service the users described in above section but there exists a category of *Non-Technical End Users* which includes all those clients of software industry that do not deal in software but intend to utilize a software tool/product to aid profit, promptness and convenience in their business. When such a customer places an order with a software company it becomes necessary to keep track of what's the progress of his order to ensure his investments has gone to useful sinks.

Often vendors keep their clients informed (sometimes they are obligated to) and meet up to showcase, if a part of the project partially satisfying customer's requirements is completed. For such purposes *Test Harness* may be used as a demonstration tool. In such situations a client can judge how the product/tool under development is going to behave when made to work in actual environment. The 'environment' here may be 'to work in conjunction with other software tools already in use at client's site'. *Remote Test Configure Tool* will prove to be beneficial in this case as it will save client's time in aggregating all test files.

Need: Use of *Remote Test Configure Tool* by this category helps improve co-ordination between software industry and its clients by constructive feedbacks. The impacts are:

a) Immediate intimation to vendor if product development process is lacking in considering some requirements or if client changes some requirements
b) Prompt action and design changes if product is not working as expected in its destined environment

Conclusion: A fairly simple user interface and summarized test result display is desired for this user. As he/she does not directly relate to software usage so a tool which generates desired results just on a 'button's' click is the most sought-after in this case.

7.1.3 Users from Academia

Although *Remote Test Configure Tool* is more suited for large commercial projects but academic work places are the platform where almost everything can be put to some useful work. This tool may be put to a very efficient use in some on-the-spot code development competition where codes developed can be instantly checked on-the-spot only. Also in a university, *Remote Test Configure Tool* can be utilized by an instructor to test numerous project submissions quickly if he/she has developed some test code to test those submissions.

Overall Impact: Based on the discussions in previous sub-sections, following conclusions can be drawn:

- **I.** A common conclusion about *Graphical User Interface* of *Remote Test Configure Tool* is that it should be fairly simple and user friendly so that all categories of users do not have to make efforts to figure out how to configure tests and rather concentrate on tests.

- **II.** *Team Leaders, Code Developers* and *Testing Teams* will need to repeatedly run tests on same set of test files as changes are made to them. So *Remote Test Configure Tool* will keep a history of recently tested projects so that these class of users need not browse all the way down to the files they have already used in testing many times.

- **III.** Certain classes of users for example those belonging to *Non-Technical End Users* and *Testing/QA* teams, due to specific reasons as quoted above, would appreciate a tool which can run on just a button's click and generate required results without requiring any further user interference.

7.2 Use Case Diagram

This section points-out all possible interactions which may take place between a user and *Remote Test Configure Tool.* These interactions are depicted here with the aid of a *Use Case Diagram:*

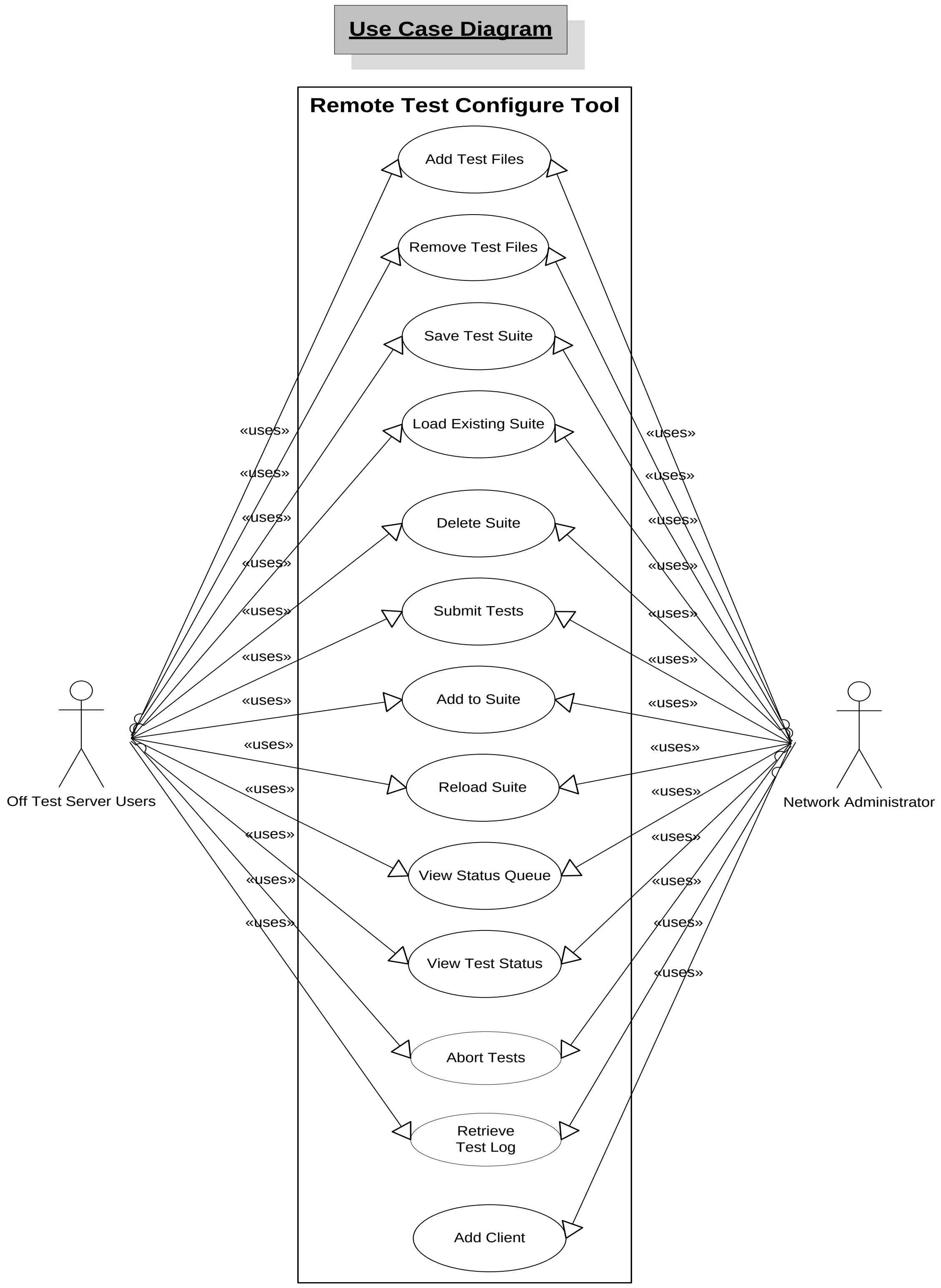


Figure 5: Use Cases for Remote Test Configure Tool

7.3 Use Cases

The *Use Case Diagram* shown in last section represented all possible interactions of the tool with its users; this section discusses each of those in brief.

7.3.1 Add Test Files

User will probably be first interested in performing this command. The user will be capable of selecting whether to browse in local terminal or a remote terminal and select multiple files to add to test suite.

7.3.2 Remove File/s

If user thinks to execute lesser tests at a time or due to some other reasons, he/she may do so by removing (multiple) files from test suite.

7.3.3 Save Test Suite

User will have an option to name and save a test suite he/she will be performing tests on very frequently so that next time tests can be performed by loading this suite which already exists.

7.3.4 Load Existing Suite

A user will be able to load an existing suite if he/she saved one in past so that every time same files will not have to be selected if same test suite has to be tested.

7.3.5 Delete Test Suite

User will be able to delete an existing suite.

7.3.6 Submit Tests

After user has added all desired test files to test suite, he can submit the suite to test server.

7.3.7 Add to Suite

After submitting suite to server, if a user intends to add some more files to the suite he/she may do so using this command. This will result in starting the tests again with new files in consideration this time.

7.3.8 Reload Suite

A user will be able to reload the suite (already loaded) to perform tests again after certain modifications to one or more files of suite. This will be the case when a set of modules are continuously concurrently tested and developed.

7.3.9 View Status Queue

After submitting a suite to test server, if test does not start immediately that means submitted suite is in a queue and will be executed in a priority order. In this case user will be able to view status of his/her suite in the queue.

7.3.10 View Test Status

User will always be able to view status of tests being performed at that time at server. The status bar will show how many tests are still to be performed.

7.3.11 Abort Tests

If due to some reasons user decides to cancel tests he submitted earlier and not yet executed, he/she may choose to do so by aborting the test suite submitted.

7.3.12 Retrieve Result Log

After tests complete, user will be able to retrieve and save test result log from server.

7.3.13 Add Client

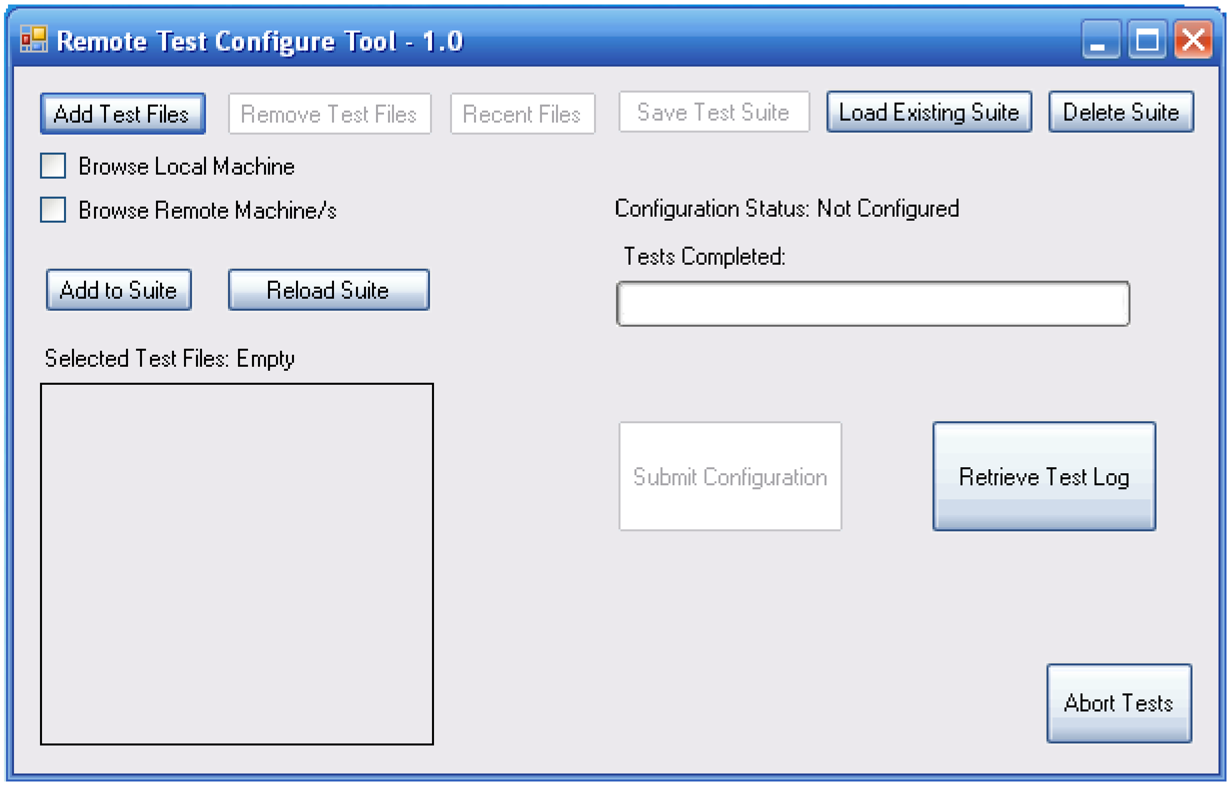
With expansion of local network, it may be required after some time to add some new clients. For the new clients to facilitate *Remote Test Configure Tool*, the *Network Administrator* will be required to configure new clients on all existing terminals supporting the tool.

Conclusion: Above use cases cover a wide range of interactions probable between users and *Remote Test Configure Tool* considering all of its users. The *Graphical User Interface,* if incorporates all of above features, will provide all classes of users with a very convenient and friendly interface to configure tests. '*Add to Suite'* and '*Abort Tests'* may be difficult activities to handle as they will be requested when test have already been submitted to test server.

8. User Interface Views

This section describes the graphical interface between the user and *Remote Test Configure Tool*. *RTCT* is intended to be used mainly by the software professionals where it will most likely be put to repetitive use and thus the interface must be very simple and quick to use so that not much efforts and time should be invested in using the tool rather than performing tests and analyzing the results. As explained above, the users may also be from academia that will use such a tool for the first time and also a *Customer* need not necessarily be having much software experience and thus an effective and easily comprehensible user interface must be provided by the system. Following are screenshots of possible interactions a user may have with the tool are given along with their description.

Screenshot # 1: User will be presented with screen shown in *Screenshot # 1* below. As seen from figure, buttons 'Submit Configuration', 'Remove Test Files', 'Recent File/s' and 'Save Test Suite' are disabled because no files have been added to test suite yet.



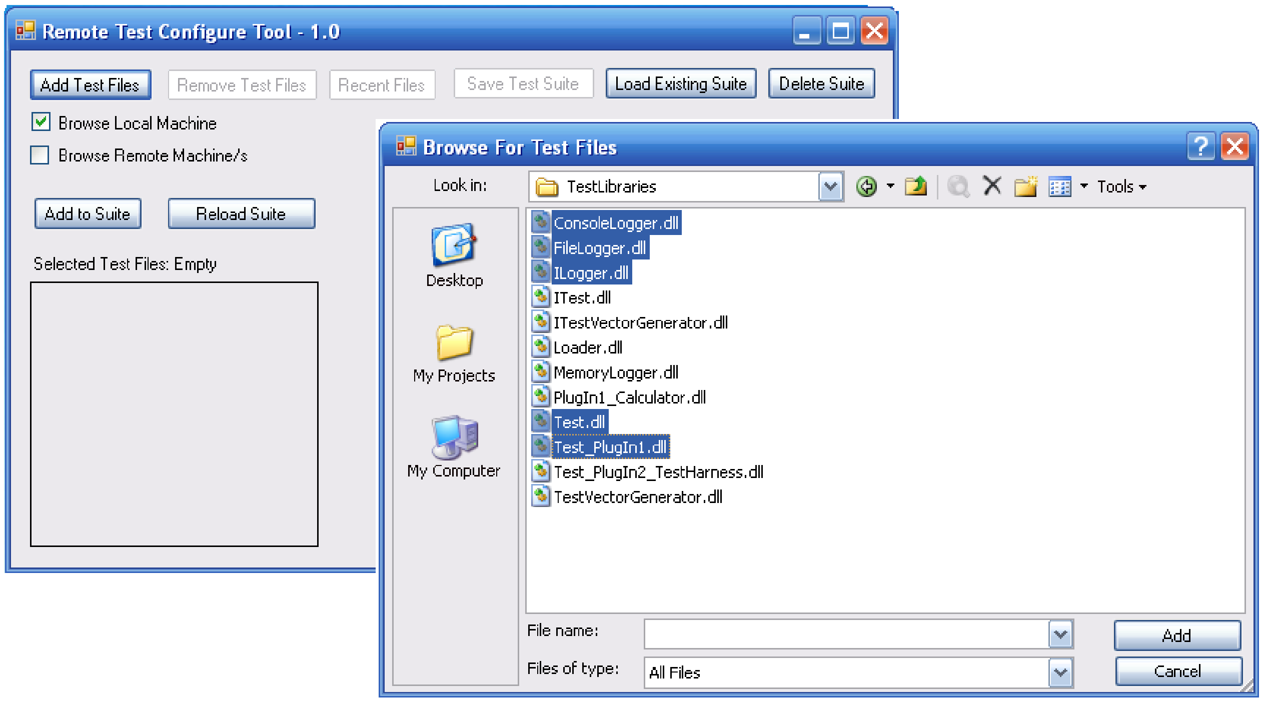
Configuration Status: Not Configured

Tests Completed:

Selected Test Files: Empty

Screenshot # 1

Screenshot # 2: On pressing 'Add Test Files' button, a file browser will be opened as depicted in *Screenshot # 2*. Two checkboxes are provided to select whether to browse local or remote terminal/s (in this screenshot local machine is browsed). The user will browse to particular directory and will be able to add one or more files. It is to be noted (although not shown here) that source code can also be added as a test file.



Selected Test Files: Empty

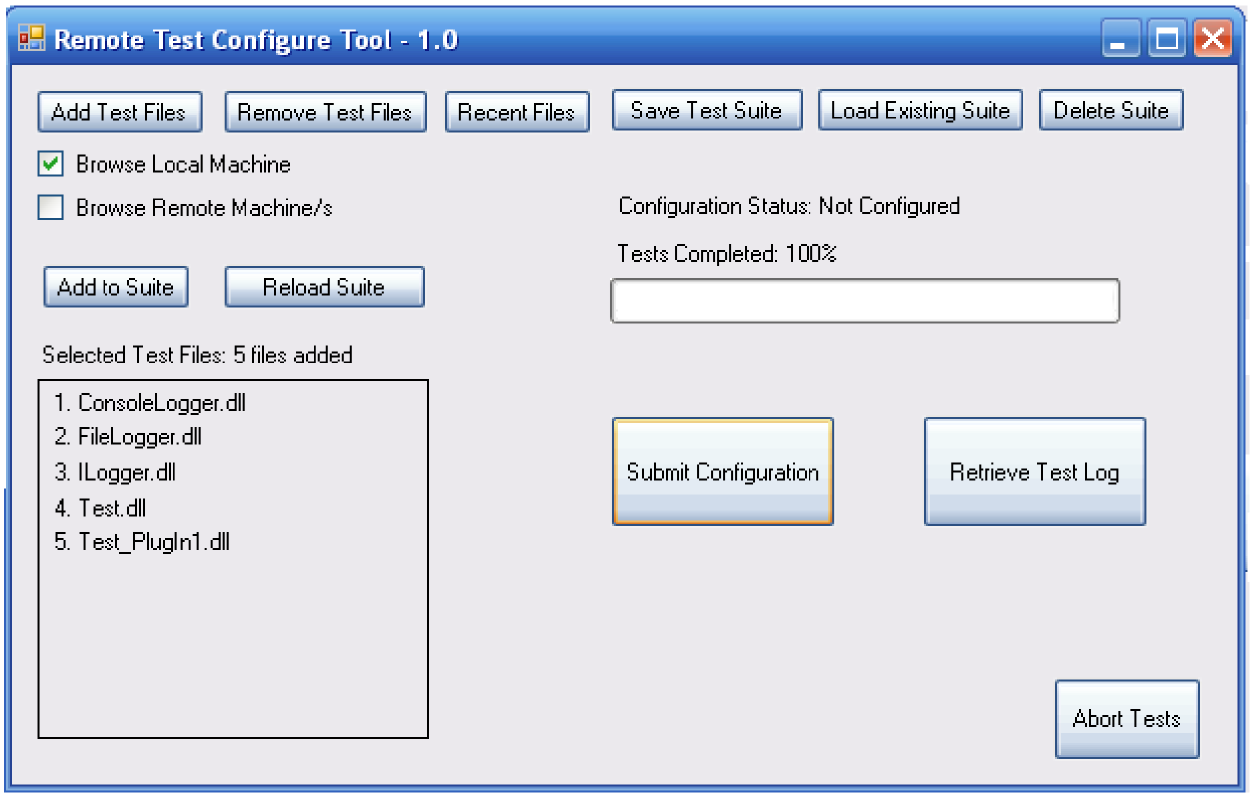
Look in: TestLibraries

ConsoleLogger.dll
FileLogger.dll
ILogger.dll
ITest.dll
ITestVectorGenerator.dll
Loader.dll
MemoryLogger.dll
PlugIn1_Calculator.dll
Test.dll
Test_PlugIn1.dll
Test_PlugIn2_TestHarness.dll
TestVectorGenerator.dll

Files of type: All Files

Screenshot # 2

Screenshot # 3: As shown in *Screenshot # 3*, the browser window is closed on clicking 'Add' button and selected test files are added to 'Panel' with title 'Selected Test Files' which also shows how many files are there in the suite. As soon as more than one file are added, the initially disabled buttons become enabled as seen here because now test suite is not empty and thus these commands can be executed.



[✓] Browse Local Machine

[ ] Browse Remote Machine/s

Configuration Status: Not Configured

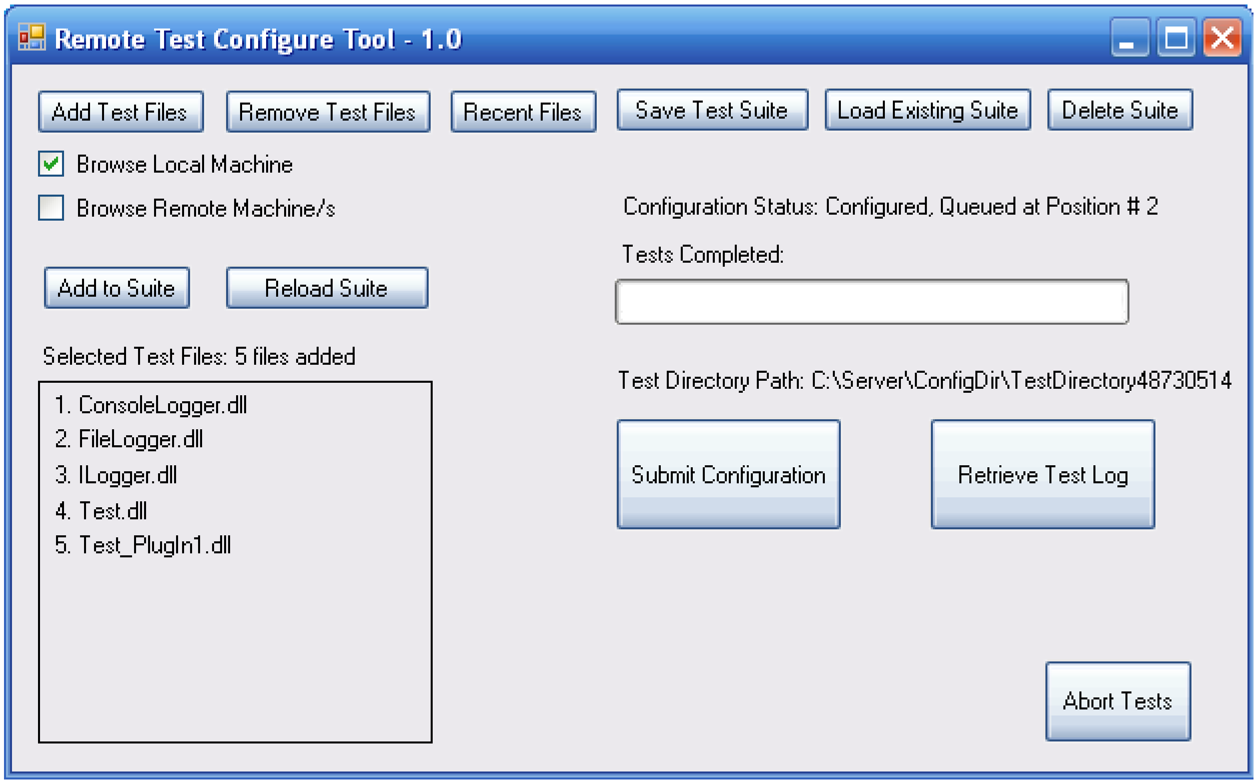
Tests Completed: 100%

Selected Test Files: 5 files added

1. ConsoleLogger.dll
2. FileLogger.dll
3. ILogger.dll
4. Test.dll
5. Test_PlugIn1.dll

Screenshot # 3

Screenshot # 4: Now if 'Submit Configuration' button is clicked window displays some more information as shown in *Screenshot # 4*. Complete path to test directory on server where all test libraries to be loaded are copied is shown next to 'Test Directory Path'. 'Configuration Status' shows status of submitted test/s from this terminal. In this case the configuration will be processed after another configuration. Note the name of test directory created on server, this is a specific nomenclature for avoiding certain issues as discussed in section 9.2 (Test Server Issues).



Configuration Status: Configured, Queued at Position # 2

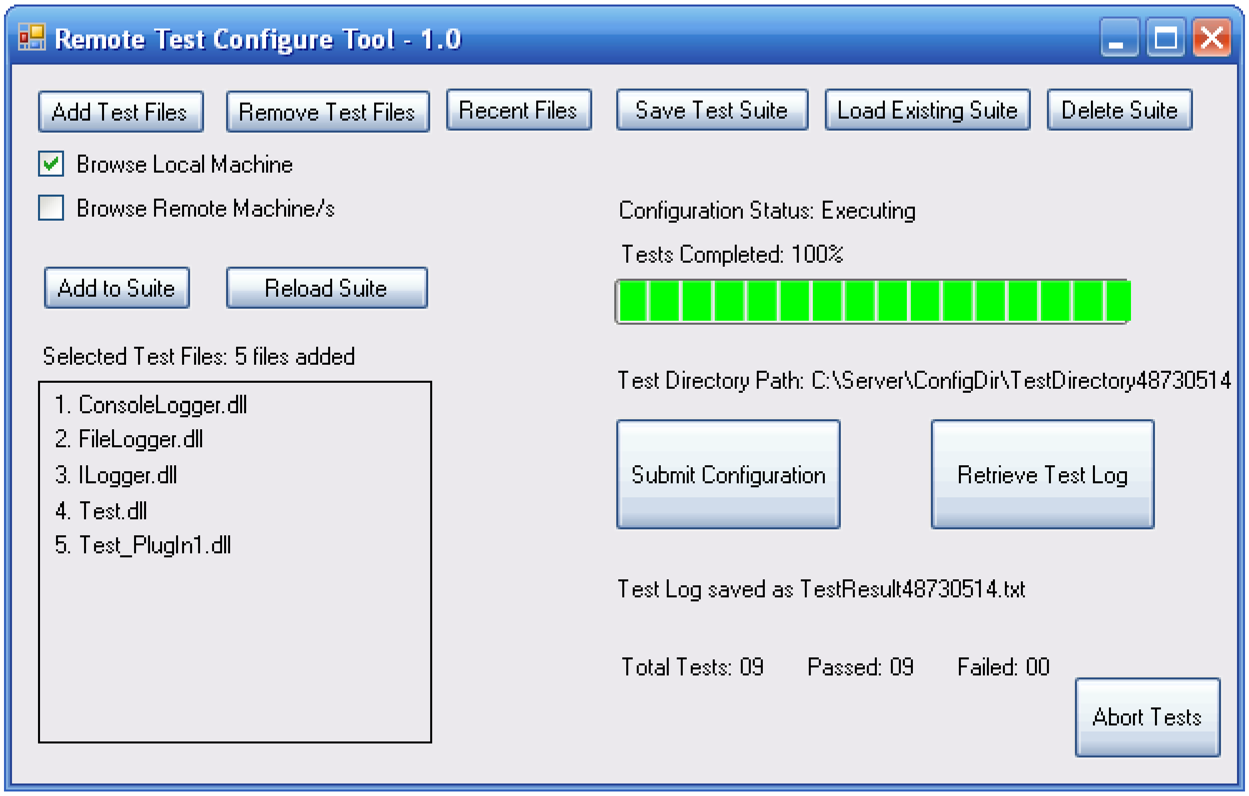
Selected Test Files: 5 files added

1. ConsoleLogger.dll
2. FileLogger.dll
3. ILogger.dll
4. Test.dll
5. Test_PlugIn1.dll

Test Directory Path: C:\Server\ConfigDir\TestDirectory48730514

Screenshot # 4

Screenshot # 5 & 6: Once *RTCT* starts test configuration submitted by this *Master RTCT*, 'Configuration Status' changes to 'Executing' as shown below in *Screenshot # 5*. On completion 'Tests Completion' turns 100% and a brief summary of tests is displayed on the interface. Again notice the name of test result log (discussed in next section) a part of which is identical to test directory name. A screenshot is shown here of the console logger's output (*Screenshot # 6)*.



Configuration Status: Executing

Tests Completed: 100%

Selected Test Files: 5 files added

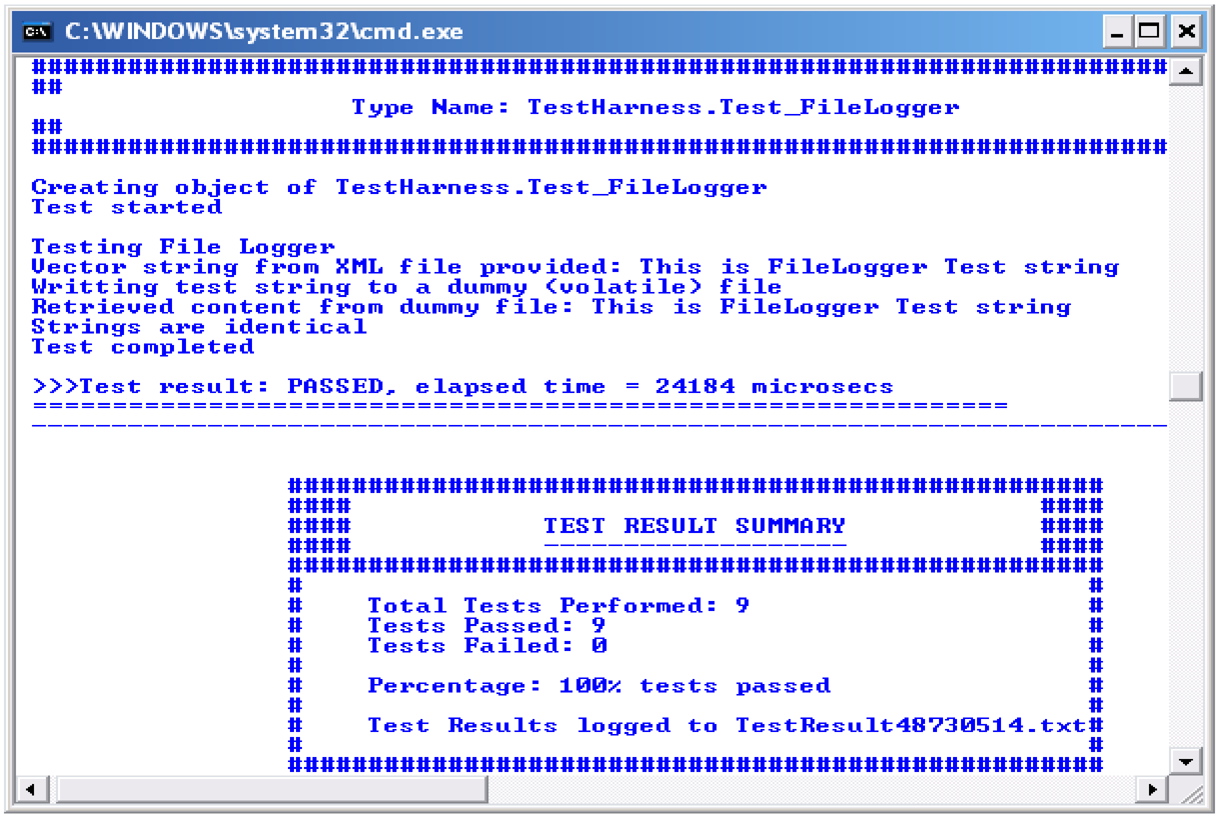
1. ConsoleLogger.dll
2. FileLogger.dll
3. ILogger.dll
4. Test.dll
5. Test_PlugIn1.dll

Test Directory Path: C:\Server\ConfigDir\TestDirectory48730514

Test Log saved as TestResult48730514.txt

Total Tests: 09 Passed: 09 Failed: 00

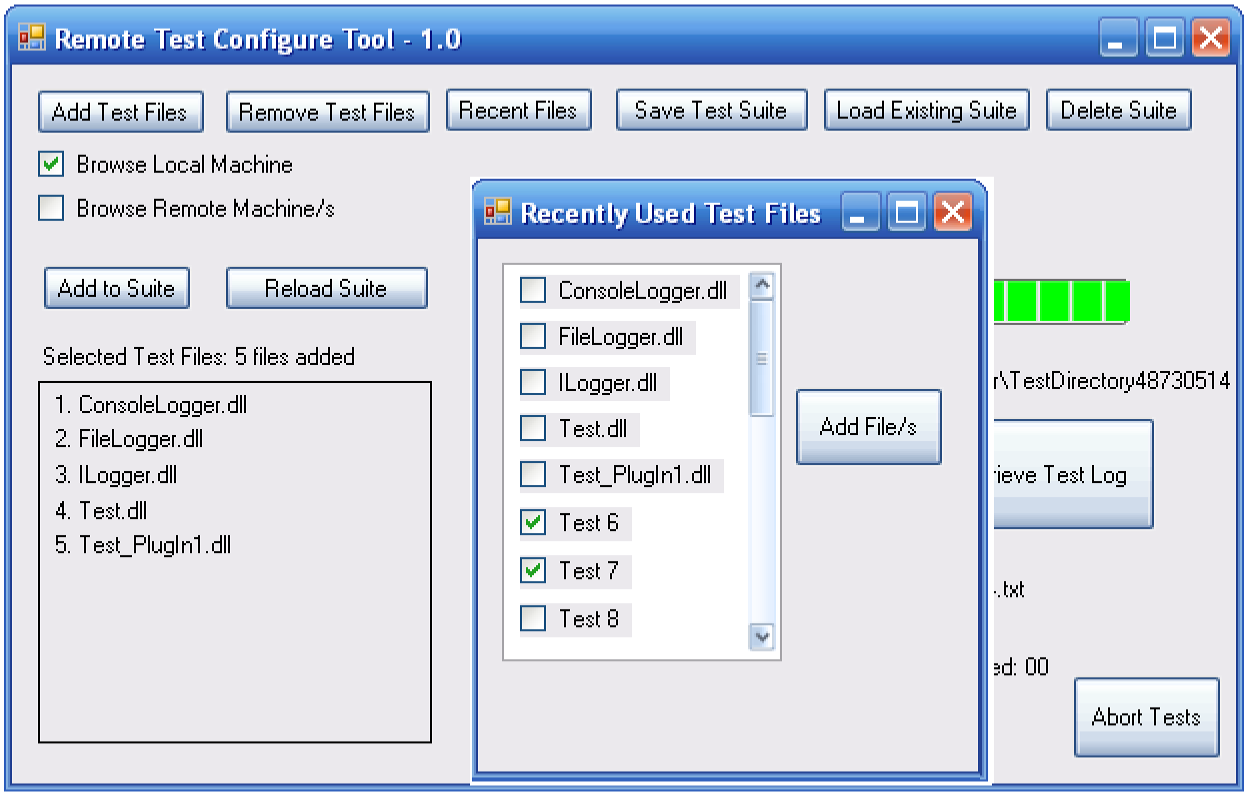
Screenshot # 5



```
C:\WINDOWS\system32\cmd.exe
##########################################################################
##
                    Type Name: TestHarness.Test_FileLogger
##
##########################################################################

Creating object of TestHarness.Test_FileLogger
Test started

Testing File Logger
Vector string from XML file provided: This is FileLogger Test string
Writting test string to a dummy (volatile) file
Retrieved content from dummy file: This is FileLogger Test string
Strings are identical
Test completed

>>>Test result: PASSED, elapsed time = 24184 microsecs
================================================================
--------------------------------------------------------------------------


                ####################################################
                ####                                            ####
                ####            TEST RESULT SUMMARY             ####
                ####            -------------------             ####
                ####################################################
                #                                                  #
                #    Total Tests Performed: 9                      #
                #    Tests Passed: 9                               #
                #    Tests Failed: 0                               #
                #                                                  #
                #    Percentage: 100% tests passed                 #
                #                                                  #
                #    Test Results logged to TestResult48730514.txt#
                #                                                  #
                ####################################################
```

Screenshot # 6

Screenshot # 7: Now that a test has been performed, *RTCT* has some history of recently tested files. Now if user clicks 'Recent Files' button, a window pops up as shown below in Screenshot # 7 which populates recently tested files in the history of tool. At this stage one/more files can be added to the test suite by checking the proper check boxes (Test 6 and Test 7 here) and clicking 'Add File/s; button.



- [✓] Browse Local Machine
- [ ] Browse Remote Machine/s

Selected Test Files: 5 files added

1. ConsoleLogger.dll
2. FileLogger.dll
3. ILogger.dll
4. Test.dll
5. Test_PlugIn1.dll

- [ ] ConsoleLogger.dll
- [ ] FileLogger.dll
- [ ] ILogger.dll
- [ ] Test.dll
- [ ] Test_PlugIn1.dll
- [✓] Test 6
- [✓] Test 7
- [ ] Test 8

r\TestDirectory48730514

.txt

ed: 00

Screenshot # 7

Conclusion: The *User Interface* designed is very convenient to use and results are displayed in a simple & easy to understand and analyze. In addition, buttons for all possible interactions automate the process to a large extent. Many features are added to make the tool interface convenient and easy to test a same set of files again and again.

9. Issue Analyses

'*Issues*' are actually like corner-cases when we talk of developing software, which must be given proper thought before starting with the project or else a vendor may have to face discontented client reviews. This section discusses some issues which are pivotal in *RTCT's* sound functioning. For each issue, first the possible problem/s is/are discussed in brief, then a solution is suggested which if incorporated into design may, at least, moderate the situation; at last each issue has been identified to be either as a '*critical'* or '*not critical'* issue.

9.1 Performance Issues

These issues deal with the load and memory usage of system.

Issue # 1: What should be done if data in the network transfers increase to some uncontrolled levels at peaks hours? What if in a large project, tests keep on getting configured at the server and data traffic increases to a limit where performance starts deteriorating?

A XML Test Configuration File will only be sent to server after all test files have been copied on the server. Suppose at a given time a server is heavily loaded performing tests and sending test result display messages and test result logs to respective *Master RTCT*. At this time when already a server is so loaded, if it keeps getting test files and configured tests, this will result in delay in tests which were being performed and even more delay for newly configured tests to be executed hampering overall performance of the tool due to high loads.

A **Load Analysis** is done below which better explains this issue in direction of *Principal Users* discussed in previous sections. As these figures closely represent a real† large project development scenario, the importance of this issue may be truly assessed.

† Mr. Navit Saxena (E-Mail: nsaxena@syr.edu) suggested me these figures as I have no work experience in software industry. Mr. Rohit Missar's OCD for 'Remote TestBed' (Version 1.0) has been referred for analysis purposes.

Consider the following project details:

Project Team Size = 100
Total number of teams = 10
Total *Code Developers* = 80
Total *Team Leaders* = 10
Total *Managers* = 5
Total *Architects* = 5

This leads to a team of '*8' Code Developers* which is a fairly real world situation for a large project. *Test Harness* (Version 1.0) developed in 'Project 2' generates test log of approximately 11 KB; if only details pertinent to tests are logged then this size may be say some 30 KB (average figure) for a large project involving many more tests. Thus,

Size of test log = 30 KB
Size of XML Message file = 500B

Consider a large project as:

Total Line of code = 1000000
Number of modules = 250

Take the case of a request for retrieving Test Result Log files for complete project from server. Considering only the one way traffic when log files are being sent to *Master RTCT* by *Server RTCT* (i.e. XML Message file is not considered), size of all log files comes to be:

Number of modules * Size of one test log file = (250 * 30) KB
= 7500 KB or 7.5 MB

Log files would be requested mostly at the start of a day to review previous day's scheduled test result logs and secondly at the end of the day to review the day's work. If we take 85% of team members to be at work at these peak hours then server would face 85 such requests which will push into network total bytes given by:

= (85 * 7.5) MB = 637.5 MB

Now for any given local network bandwidth, this is a heavy size and really a gigantic load for server to service.

Possible Solution: There are two possible solutions to tackle this situation:

1) Some 'Caching' mechanism may be used for holding test result logs. Whenever a *Master* RTCT requests the log, test log on the cache is checked for being the latest copy of the log. If so, cached copy is returned. This will definitely moderate load on server.

2) When server can not support any more requests, a message may be sent to all *Master RTCT* to be displayed to user that server is overloaded and if still files are being sent then a queue will be established at the respective *Master RTCT* which will hold these new blocks of data. When server is relieved of its load, it fetches data from these queues until they exhaust.

Implementation Decisions: *RTCT* will be designed keeping in mind both of above solutions as a combination of these will take care of heavy test log requests and heavy traffic due to copying of test files. But there will be a limit to size of queues established at *Masters* so adding files to these queues will create problems after some level which makes it **a critical issue.**

Issue # 2: Issues with crashes due to resource contentions

Whenever *Test Harness* is started at the server, all configured tests are served on different threads. When these requests go high, memory contention may begin leading to one process using up the memory allocated to another process resulting in unstable environments that will eventually crash the server and whole application.

Possible Solution: A solution for this issue may be to run each test configuration in a different AppDomain. This will facilitate unloading the child AppDomain once tests configured to run in it are exhausted. This way now very fewer processes will have a chance to interfere with each other's resources.

Implementation Suggestions: *RTCT* will create a new child AppDomain each time a XML Configuration file is read and tests will be executed in this newly created domain. As this issue can be easily taken care of so it **is not a critical issue.**

9.2 Test Server Issues

These issues are concerned about the processing done on test server.

Issue # 3: Issue with naming of Test Directory on server

To save time and service a *Master RTCT* at the earliest possible, each test configuration will be run on a separate thread. When tests are performed *Test Harness* will load test libraries from a specified test directory. Directory name can not be same for all tests as then *Test Harness* will load libraries for all tests from same directory as all test files will be copied to same directory and respective *Master RTCTs* will not get test results fir tests they configured. This will create great confusion for *Test Harness* and it may eventually crash.

Possible Suggestion: Root cause for this issue is naming of test directory on server so this issue can be handled if some naming mechanism can be figured out so that every time a test directory is created it has a unique name which is taken for the first time. There are two possible solutions for this issue:

1) When user adds test files to test suite in the initial steps, at that time user may be prompted to give a name for test directory to be created. If a directory with that name already exists user will be prompted to enter a different name. This is a feasible solution but then tool will no more be a 'single push-button click tool' as required by certain classes of users.
2) Another solution exists which will handle this issue and will not let the user even know about any such problem/activity. This solution is to implement a mechanism which randomly names test directory to be created at server when creating XML configuration file at *Master RTCT*. One such mechanism is explained below:

Test Directory Naming Mechanism: Every time a *Master RTCT* is started, it contacts *Server RTCT* and gets itself registered. This way every *Master RTCT* on network is assigned a random *Terminal ID* which has not yet been assigned to any other terminal. If we observe carefully, the *Server RTCT* will select a configuration file from a bunch of them based on some priority criterion and starts running specified tests on a separate thread. Thus the server creates only one configuration file at a given time or put simply, time of creation is unique for every configuration file. Thus directory name will be created as:

Directory Name = "TestDirectory" + Terminal ID + Current Time

For example, if two users submit two test configurations from two different terminals at the same time (considering worst case scenario), say 18:34 hours and let terminal IDs are 1234 and 9876 respectively.

Directory Name 1 = TestDirectory12341834
Directory Name 2 = TestDirectory98761834

NOTE: It is assumed that system time on all machines on the network is synchronized.

__**Implementation Suggestions:**__ Solution # 2 will be incorporated in design and will be used for naming of test directory. This naming mechanism can be easily implemented in C# and thus it **is not a critical issue.**

Issue # 4: Issue with order of selecting XML configuration files.

When numerous users configure a number of tests on the server, it depends on *Server RTCT* as to which XML configuration to select and start performing tests mentioned in it, but then what order should the server acquire or else if to be selected in some priority mechanism, how should the priority of XML configuration files be decided?

Possible Solution: This issue is important to maintain fair and logical test execution. There are two possible options for this:

1) Randomly select a configuration file from available set of files and start executing it. This is a simple solution but not at all fair (on many occasions). In this manner a configuration submitted last may be selected and started first which will be unfair to other users who submitted their tests quite early but had to wait without any reason.

2) Another solution is to keep a record of number of libraries to load for each configuration and the time stamp when XML file was submitted. The XML file which lists maximum number of libraries to be loaded will be selected first. After this configuration is operated on, other configurations will be checked for time stamps of their arrival on server. Now configurations will start loading in order of their arrival in parallel. Once configuration with maximum libraries to be loaded exhausts its tests, remaining configurations will again be checked for maximum number of libraries to be loaded and the process will continue until there's only one configuration file at server.

__**Implementation Suggestions:**__ Later solution will be implemented into design. Although this involves a higher logic level that the former one, but it will maintain fairness and keep the execution sequence in a logical order. As discussed above, this issue can be properly handled and thus it **is not a critical issue.**

Issue # 5: Issue of overwriting system files on server.

Server is the central processing element in a *Remote Test Configure Tool* set-up. While files are copied from *Master RTCT* and various *Slave RTCTs,* there is a high risk of important system files to be overwritten in this process. This may bring the whole network communication down.

Possible Solution: Apart from only allowing *Network Administrator* to manipulate setting on test server, it must be made an obligation during installation process to mention critical drive (if more than one drives are present) from important file point of view. One more, even better, solutions is to enlist names of important system files which are present on every machine and store this list in a data structure. Whenever a file is being copied to server, name of file must be checked against names in this list; if a match is found user is prompted to rename the file/s.

Implementation Suggestions: As this is a security issue for server, both of above measures will be incorporated. Although later suggestion holds weight, *Network Administrator* setting will prove to be a second line of defense. By no means can *RTCT* always find out which are important files; *Administrator's* negligence/ignorance/mistakes (administrator being a human) can not always be relied upon. This makes this issue **a critical issue.**

Issue # 6: Issue of deletion of XML configuration file.

Although this is really a rare case, but due to many possible reasons (for example inadvertently by *Network Administrator*) one or more XML configuration file/s may be deleted from server. In that case, user will wait after submitting the tests and will never see the results because test libraries are at server but configuration file corresponding to those libraries is deleted.

Possible Solution: A simple and feasible solution may be to keep a track of how many configuration files have been received by the server. This can easily be done with a counter. Now every time *Server RTCT* comes to configuration file pool, it should verify that number of files is same as indicated by counter. If so proper configuration is opened to read and processed and counter is decreased by one.

Implementation Suggestions: Using counters for tracking configuration files, this issue can be easily handled and thus it **is not a critical issue.**

9.3 Data Transfer Security Issues

These issues deal with theft of data which are transferred over network to configure tests at server.

Issue # 7: Issue of data hacking during transmission.

With increasing competition in every field, it is very possible that a firm's competitor may try to hack the data being transferred during testing. This may result in leak of important information about a project under development or even tampering of important code within interiors of the organization.

Possible Solution: Two potential solutions for this issue are as below:

1) Data may be encrypted before transferring to other end. At destination data is again decrypted and interpreted. This is a reliable method but involves overhead in terms of extra CPU cycles for encoding and extra financial resources.

2) Another remedy may be to use a coding which can be decoded by only terminals on the network. Also rules like CRC checks may be incorporated.

Implementation Suggestions: As quoted above, former is not an economical solution whereas later does not ensure complete data security. Thus, as such, no reliable solution is found for securing data against hacking and this **is a critical issue.**

Issue # 8: Issue of communication line disconnections.

Local networks are usually always laid through workplaces of employees in an organization. Due to personal intentions, accidents or insects, communication channels may face physical breakages. These breakdowns are very critical as in this case, message sent from *Master/Server RTCT* will never be received by the *Slave RTCT* and thus system may enter an undefined state (possibly a deadlock).

Possible Solution: Buffers may be used at terminals so that every outgoing message will be stored in the buffers until an acknowledgement is received from receiver. Thus in case of channel breakages, buffers will contain the recently sent message and as soon as connection is resumed unacknowledged message is sent again.

Implementation Suggestions: This is a reliable solution for the problem but may increase intricacy and complexity of the code. As this solution is feasible so this is **not a critical issue.**

9.4 Source Code Build Issues

Issue # 9: Issue of source code builds

Test Harness loads only Dynamic Link Libraries built from the source code to be tested. If a source code (i.e. *.cs file) it will not even load that file. A user can fairly select a source code file as one of its test files and it's a project requirement to build that source code to a DLL. Now this can be done at the *Master/Slave RTCT* where the source code resides or else it can be first copied to test server and then built into library at the server. Which one is better?

Possible Solution: It is always better to build source codes into libraries at their host machines (and not at test servers) because of two reasons:

1. Building libraries at server will adversely affect its performance at times of peak loads when in addition to service client requests and performs tests it will have to do this addition processing while clients will be comparatively much less loaded.

2. If source code is built into library at their hosts, XML Configuration files will be created with final listing of test libraries in it; whereas if built at server, first server will read for any source code in test directory, if one exists it will built it and then update XML configuration file so that when it starts loading libraries it gets all of them.

Implementation Suggestions: Source codes, if any selected as test files will be built into libraries (DLLs) on machines where they reside. The logic to incorporate this decision can be easily implemented and thus it is **not a critical issue.**

9.5 Test Result Log Issues

These issues are related to processing of test result logs.

Issue # 10: Issue of naming of Test Result Log.

Dependent on user, he/she may test logs from server. Now if name of test logs is fixed then every time a log is retrieved, a single file is either replaced with the new one thereby losing old logs or appending old logs with new test results. How can test logs be made unique for each test performed?

Possible Solution: A very simple solution is to use the same file name as the directory name of test directory as discussed in Issue # 3 under section 9.2 with some modifications. For example test result logs for those two cases will be:

Test Directory: TestDirectory12341834	Result Log: TestResult12341834
Test Directory: TestDirectory98761834	Result Log: TestResult98761834

Implementation Suggestions: As shown above this issue can be easily taken care of, this **is not a critical issue.**

Conclusion: Some *critical issues* are identified and a potential solution to problems caused by these issues has been suggested for each one. Decisions needed to be taken while implementations are also mentioned. After proper implementation of these solutions, Remote *Test Configure Tool* will be implemented as a very efficient application from the view point of memory space and execution time.

10. Modular Architecture

Remote Test Configure Tool will be developed as an integration of two modules namely 'Client Module' and 'Server Module'. This is due to slight difference in functional requirements of *Master/Slave RTCT* and *Server RTCT*.

A Client Module should provide following facilities to user:

- Browsing local machine
- Browsing remote machine/s
- Sending messages/commands to server and slaves
- Receiving messages/commands and execute them

Similarly a Server Module should be capable of:

- Selecting a configuration file from a pool and interpret it
- Sending messages/commands to clients
- Receiving messages/commands and execute them
- Running Test Harness

In an abstract way, both modules have most of functions common except for a few. Thus Client Module and Server Module will be developed as consisting of same set of modules (except a few) so that they are reusable. It should be noted that however the (modular) structure in which these modules interact is different in the two cases.

10.1 Layers

Modular architecture of a *RTCT* can be thought of as composed of four main layers:

- Presentation Layer
- Processing Layer
- Data Storage Layer
- Communications Layer

Presentation, Processing and *Communications* layers, as the names suggest, are responsible for user interactions, data processing and communication. *Data Storage* layer however deals with 'messaging'. It should be realized by this point, as it will be explained in following sections, that interaction of *Master RTCT, Slave RTCT* and *Server RTCT* forms the very basis for proper functioning of *RTCT* and these interactions are implemented using message passing. Thus 'messaging' and so *Data Storage* layer are very vital for *RTCT*. Also from the discussion preceding this section it is clear that *Data Storage* and *Communications* layers will be similar for both modules however *Processing* and *Presentation* layers will slightly differ in two cases.

In the paragraphs to follow, all modules are described in brief and then working of 'Client Module' and 'Server Module' is explained with the aid of modular structures.

Note: Modules are described before explaining Modular Architecture diagrams because Client and Server modules are constituted by mostly the same modules and thus it will be easy to understand differences in interactions of same set of modules in the two architectures after we know functions of each module.

10.2 XML Encoder Module

XML Encoder belongs to 'Data Storage' layer. It gets its operand in form of strings which are appended with proper XML tags to form legal XML based messages to be used for communication. These XML based messages are then passed on to *Message Mailbox* module.

10.3 Message Mailbox Module

Another member of 'Data Storage' layer, *Message Mailbox* stores XML messages generated by *XML Encoder* module. It is an entity independent of Communications layer as it goes on storing messages irrespective of at what rate the messages are being sent. This module works in conjunction with *Send Queue* module (explained next) and plays pivotal role in case of communication channel disconnections; *Message Mailbox* keeps a message stored in it until *Send Queue* is acknowledged by *Communications* module of receiving last sent message. Thus in case of receiver disconnections, *Message Mailbox* stores all messages which were not sent due to breakage and makes these messages available to *Send Queue* once connection is re-established. This module forms the core of 'Data Storage' layer as it is the one which takes care of proper storage of data until it is safely transmitted to its receiver.

10.4 Send Queue Module

A member of 'Data Storage' layer, *Send Queue* module is responsible for fetching messages from *Message Mailbox* module and making them available to *Communications* module. As explained above, it aids *Message Mailbox* module in avoiding malfunctions during disconnections.

10.5 Communications Module

Communications module represents the 'Communications' layer. It is responsible for all interactions between different members of *RTCT* which interact across the communication channel. It will be implemented .NET Sockets. This module receives messages to be sent from *Send Queue* module and communicates with the *RTCT* whose IP address is mentioned in the message.

10.6 XML Decoder Module

As is clear from its name, *XML Decoder* module does exactly opposite to what *XML Encoder* module is responsible for. This module receives XML based messages from *Communications* module and parses the message to extract messages/commands sent in message. This module constitutes 'Processing' layer and passes extracted information from XML message to *Command/Message Handler* module (explained next).

10.7 Command/Message Handler Module

Another member of 'Processing' layer, *Command/Message Handler* module is responsible for interpreting the information (which must be a message or a command to be executed) received from *XML Decoder* module and taking proper actions. This module forms the core of 'Processing' layer as it comprises the intelligence which eventually process information being passed around in network.

Additional 'Client' Module members

10.8 Client Executive Module

This is the executive of 'Client' Module and is responsible for initiating Graphical user Interface and Communications services. This module belongs to 'Presentation' layer of Client Module.

10.9 GUI Module

Another member of Client Presentation layer, *GUI* module is responsible for interacting with the user. This module supplies *XML Encoder* and *Command/Message Handler* modules with necessary user inputs.

10.10 Build Module

A member of Client Processing layer, *Build* module is responsible for building any source code into library.

<u>Additional 'Server' Module members</u>

10.11 Server Executive/GUI Module

A Server Presentation member, *Server Executive/GUI* module serves as the executive of 'Server' module. As *Server RTCT* will rarely be used to configure tests by a user, thus GUI will not always be available at Server module. GUI will only be started when a user (often the Network Administrator) intends to configure tests from the test server.

10.12 Configuration Loader Module

This module belongs to Server Processing layer and is responsible for deciding and reading XML Test Configuration files in a logical order.

10.13 Test Harness Module

The central processing unit responsible for performing all tests, *Test Harness* module constitutes Server Processing layer. For more details on this module please refer to "Test Harness – Operational Concept Document Version – 1.0".

10.14 Working of 'Client' Module

This section explains how will the basic functions of 'Client' module will be performed with help of the proposed modular architecture for 'Client' shown below:

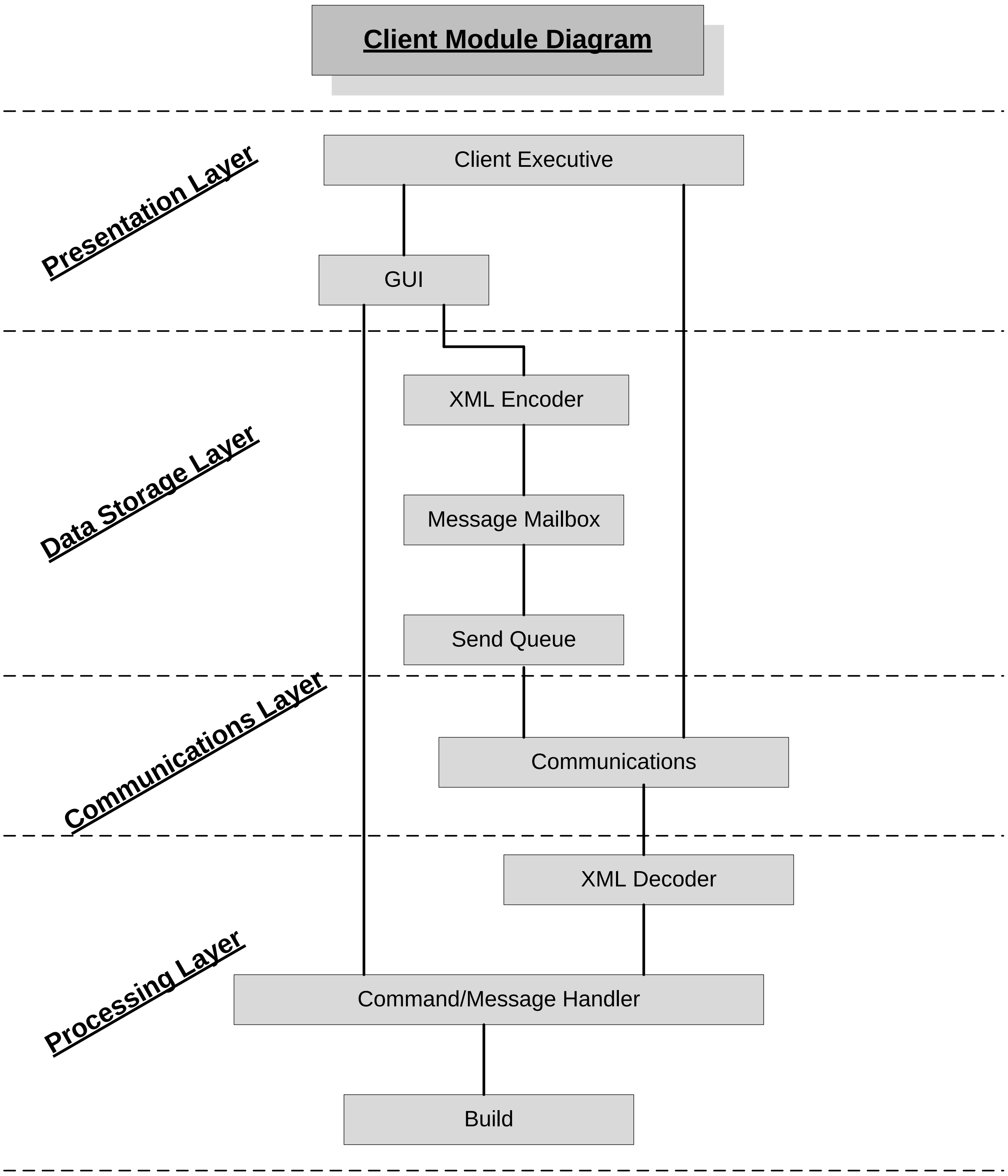


Figure 6: Module Diagram for 'Client'

a) Browsing Local Machine

This is a very simplified operation with reference to the proposed architecture. When user clicks 'Add Files' on the GUI with a check in 'Browse Local Machine' checkbox, GUI invokes a *Command/Message Handler* module service which opens up the browser to browse local machine.

b) Browsing Remote Machine/s

If a user clicks 'Add Files' on the GUI with a check in 'Browse Remote Machine/s' checkbox, GUI Client module sends a XML based message to all machines on network (as explained in part c). This message is then received by *Communications* module at all machines and proper action is taken (as explained in part d).

c) Sending Messages/Commands to Servers and Slaves

In cases when Client module needs to send out a message to a slave or the server, it invokes a *XML Encode* service which creates a XML based message. This message is then passed to *Message Mailbox* module to be stored. *Message* Mailbox module forwards this message to *Send* Queue module. Next time when *Communication* module fetches this message, it is sent to all machines on network.

d) Receiving Messages/Commands and Execute

Often a Client will receive message/command from server. This information is obtained at the *Communications* module which is then passed on to *XML Decoder* module. This module decodes the XML strings based on its knowledge of tag interpretations and pass on extracted information to *Command/Message Handler* module. Based on command/message in extracted information, proper action is taken.

10.15 Working of 'Server' Module

This section explains how will the basic functions of 'Server' module will be performed with help of the proposed modular architecture for 'Server' shown below:

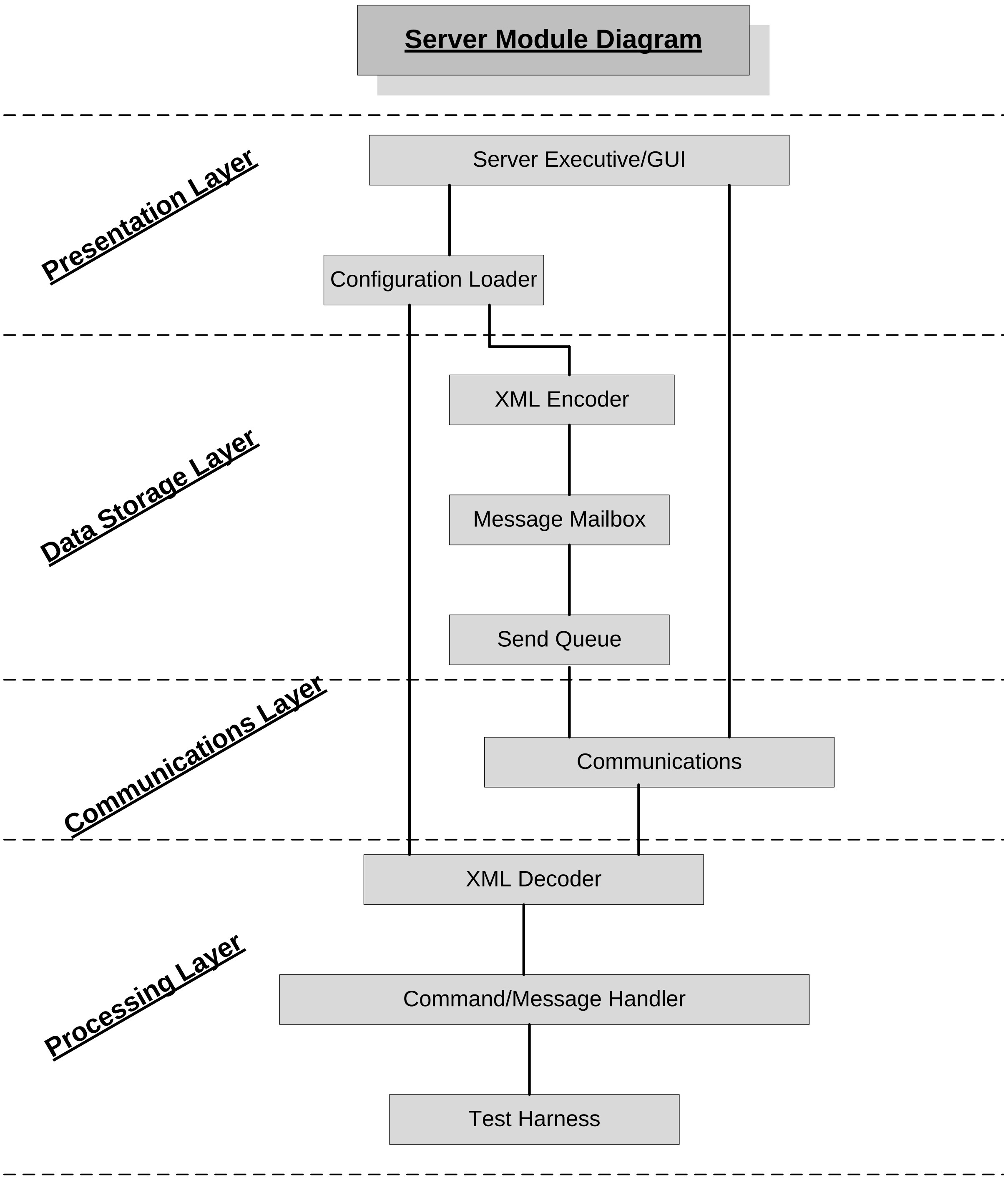


Figure 7: Module Diagram for 'Server'

a) **Selecting Configuration File and Interpreting it**

With the instantiation of *Server RTCT* the *Server Executive* module starts a service in *Configuration Loader* module. This module continuously looks for any files in its configuration pool and if finds one, the file is passed on to *XML Decoder* module which extracts test directory path and pass it to *Command/Message Handler* module to take proper action.

b) **Sending messages/commands to clients**
As explained in previous section.

c) **Receiving messages/commands and execute them**
As explained in previous section.

d) **Running Test Harness**

Whenever *Command/Message Handler* module interprets received command to execute *Test Harness*, it simply invokes the runTests() service of *Test Harness* module which spawns tests in specified test directory on a new thread.

Conclusion: *Remote Test Harness Tool* will be developed as two modules viz. 'Client' module and 'Server' module. Modular composition of the two is so designed that it results in a reusable modular architecture. Considering importance of messaging in this tool, 'Data Storage' layer is identified as the most important layer.

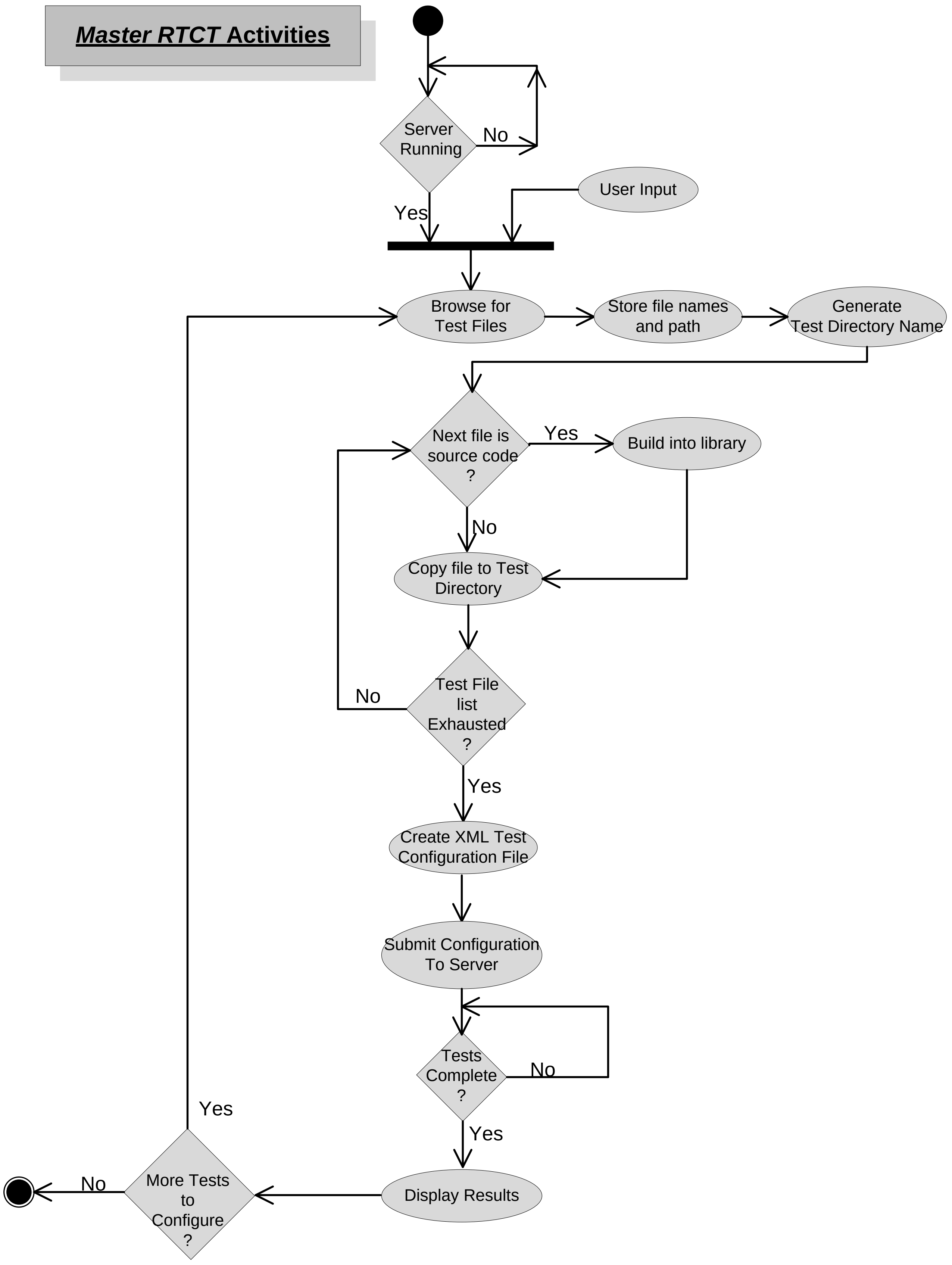
11. Activities

11.1 Master RTCT Activities

This section discusses important activities of a typical *Master RTCT*. All the main activities are represented in the form of an activity diagram below.

A *Master RTCT* starts with listening to the test server to find out whether server is running so that tests can be performed. If the server is running and user commands to add test files, browsing for test files starts depending on whether user intends to browse local machine or a remote machine. The names and complete paths of the files selected by user are stored in a data structure and name of Test Directory is generated. Now one by one an entry is selected from the data structure to check whether any file is a source code and needs to be built into a library; if one is found, it is built into library. Each file is copied to test server in the test directory.

After all files are copied like this on server, a XML Test Configuration File is created which contains test configuration information such as complete path to Test Directory and a list of all test libraries. This configuration file is submitted to server and the *Master RTCT* now waits for server to complete its tests. Once tests are completed, results are shown to the user and depending upon whether user has more tests to configure in server or not, the tool again shows the browsing options or else terminates.



Master RTCT Activities
Server Running
No
Yes
User Input
Browse for Test Files
Store file names and path
Generate Test Directory Name
Next file is source code ?
Yes
Build into library
No
Copy file to Test Directory
Test File list Exhausted ?
No
Yes
Create XML Test Configuration File
Submit Configuration To Server
Tests Complete ?
No
Yes
Display Results
More Tests to Configure ?
Yes
No

Figure 8: Master RTCT Activity Diagram

11.2 Server RTCT Activities

This section discusses important activities of a typical *Server RTCT*. All the main activities are represented in the form of an activity diagram below.

Server RTCT is the entity which should always be running so that anytime a client wants to configure tests it should be able to do so. The server can only be shutdown by a user with administrator permissions (for example a Network Administrator) say for maintenance purposes, etc. This is the reason why the server is always shown to be checking whether it is commanded to shutdown by a user in which case server activities terminate; otherwise server continues to run. As mentioned earlier, most of the times a user will configure test from client terminals of network and thus user activities/inputs are not considered here.

If server is not commanded to shutdown, it enlists all the configuration files submitted by various users. It then selects a configuration and interprets the XML tags to find out the Test Directory path. If this path is found to be invalid, an error message is displayed to the user and next configuration file, if available, is selected. If directory path is found to be valid, *Test Harness* is invoked on a new thread which then loads all test libraries from this path. After all tests are completed for this configuration, results are displayed and this sequence keeps on repeating itself.

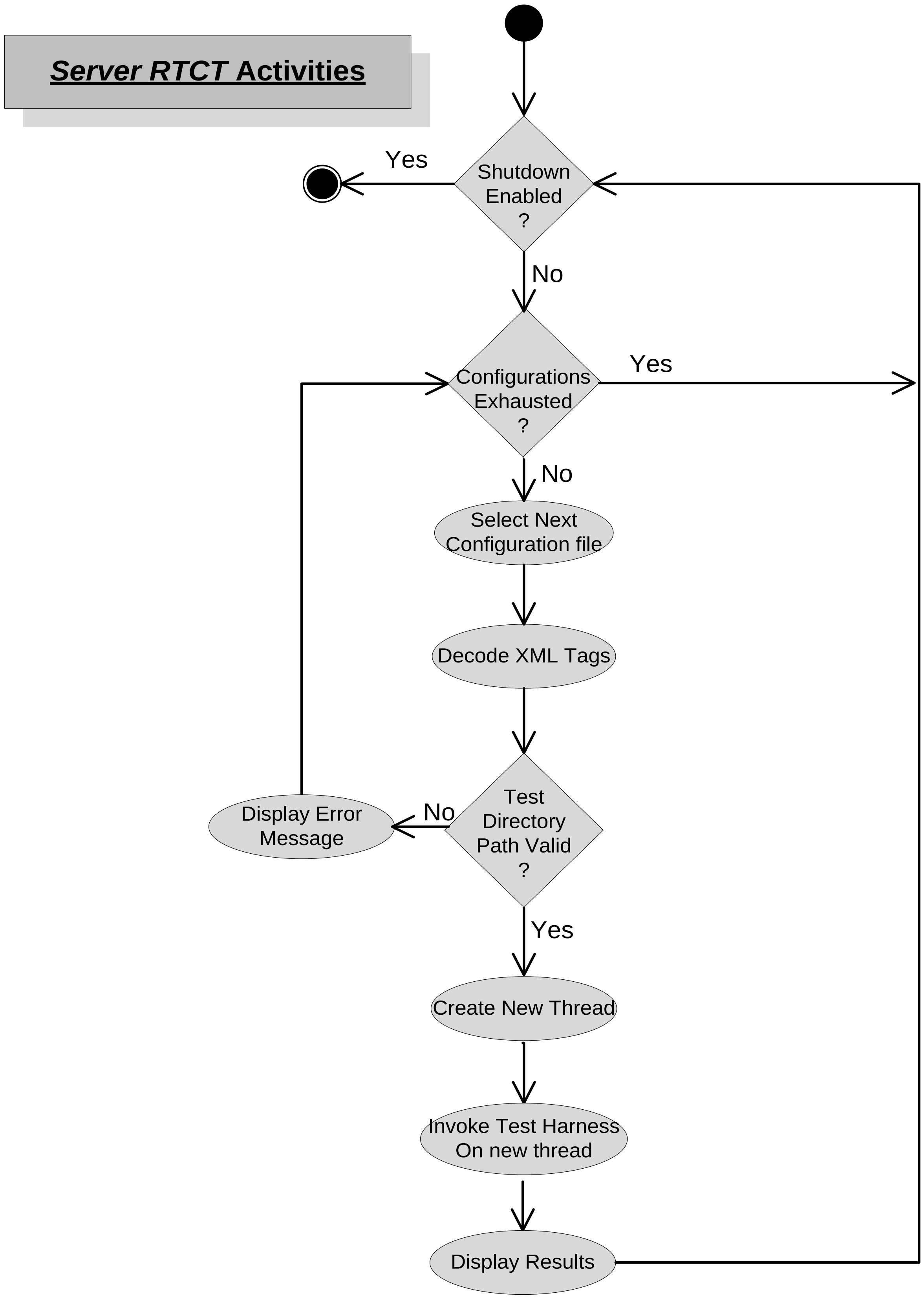


Figure 9: Server RTCT Activity Diagram

Conclusion: All main activities can be performed by dividing them into smaller activities. Such sub-activities were identified in this section. Users will not usually use server terminal fro configuring tests so *Server RTCT* will not generally start its GUI service.

12. Future Release: RTCT over 'the Internet'

This architecture of *Remote Test Configure Tool* is designed as simple as it can get, but the foundation on which it stands on is concrete. Many features can be added to get a smarter test configure tool.

One such feature is making the tool available over the internet so that every machine accessible via the internet will then be a part of the testing system and physical distances will no longer be a hurdle in continuous testing of remote dependent modules. This feature can be added by making changes in the *Communications* module. Communications over the 'Internet' are based on Hyper Text Transfer Protocol (HTTP) and in the proposed concept RTCT uses .NET Sockets for communication. If in place of .NET Sockets we use HTTP then the goal is achieved. It should be noted here that not much alterations will have to be done to this architecture because HTTP works on TCP/IP and Sockets is the underlying principle of TCP/IP protocols.

This can be done using the classes of System.Web namespace which has features implemented to enable client-server communication. It is done on the basis of its knowledge of current HTTP protocols. The important classes available for this are:

- System.Web.HttpRequest class: provides knowledge about HTTP request
- System.Web.HttpResponse class: manages HTTP output to the client
- System.Web.HttpServerUtility class: provides access to server-side utilities and processes

Said this, the document lays a firm foundation of concept, utility, architecture and issues of the tool. Based on the simplicity and flexibility of architecture many more features may be added in the future releases.