CSE 681- SOFTWARE MODELING AND ANALYSIS

Project #5

STORAGE MANAGEMENT SYSTEM DESIGN WITH CAPABILITY MATURITY MODEL INTEGRATION LEVEL 5 STANDARD

OPERATIONAL CONCEPT DOCUMENT

VER 2.0

INSTRUCTOR- DR. JIM FAWCETT

RAVICHANDRA MALAPATI

SUID: 22375-2155

Date: 9-Dec-2015

Contents

1. Storage Management subsystem- Executive summary

In software development industry, where the complex software systems are build, there are teams working from different locations but for a common project. Often the development team are distributed across globe. Maintaining these software in repository can be cumbersome. And also coordinating meetings with all these teams virtually sharing the available resources is difficult without a single system. At present in the market, all the available software management systems are based on Sql database and very few are document based. As there are many drawbacks with the existing storage management systems, we develop a new Software storage management system which can overcome all the drawback with the existing storage management system. For example with the existing sql based storage management systems the speed is very less and it is hard to store many files in a single package. And the other difficulty we face with the Sql database is maintaining the

packages as there can be different types of file extensions in a single package it becomes complex to organize using the existing BLOB and CLOB data types.

The basic services which the presently available Software management system provide are user authentication, check in code, checkout the code, compare the code with previous versions, should be able to support the cloud based architecture. In the present storage management systems the requirement tags, review tags, reviewers, customers are not part of the software management systems. In our proposed design, we provide inbuilt requirement request tool, review tool, customer access and reviewers' access tool along with the test harness tool which will validate the requirements based on the requirement request tool. Once the requirement is checked in by the developer, an automated mail is sent to the customer or team who requested the requirement. And the software management systems once runs the test harness, a quality report will get generated.

Continuous Integration (CI) and Build is a development practice which requires developers to integrate code in to shared repository any number of times a day. This is part of extreme programming (XP) which is followed by development teams across industry. The NoSQL database that we developed in the project #2 will be the best database for the applications like software management system.

We also include the feature called continuous build with a separate server. The continuous build server is evoked whenever someone checks in the code. The check in package along with the projects it is included is sent to the build server, the build server reads the input and builds the project. If the build is successful then an acknowledgement is sent to the repository server. This will be explained in the later sections in detail.

We also include the Review features in the repository. When the code is checked out by the developer the repository locks that particular revision of the package or file on the name of the developer who requested checkout. Once the checkout is done,

This document explains the architecture of the proposed Software Management system which contains primarily the following blocks.

1. Build Server
2. Test-Harness server
3. Client
4. Repository server
5. Requirements Data base

6. Review Database
7. Delivery Server
8. Quality Analysis Server

The primary users of this tool will be software developers, Managers, customers who delegate the requirements, reviewers, Quality Assurance team, Administrator of the server, software Architect, developers, programmers, architects, Quality analysts, Quality Auditors, Project Management Office (PMO), Testing Team, Client testing team and other stake holders of the software project etc. The project managers use it for monitoring and checking the deliverables. The Developers use it for checking in the code and checking out the code. The developers will use the repository server to hold reviews and enter the review points also. The architect uses the repository server to setup new project architecture. The Administrator uses the Repository server, Test-Harness server, Build server to grant access and remove access to developers. The Managers use the repository to check the deliverables during delivery and also to monitor the status of the project by tracking the number of commits per week etc. Quality Assurance team uses the repository to do audit checks and also to do spot checks. Reviewers use the repository to do reviews and provide the comments for the developers if any changes are required. The customer use the repository to access the customer specific packages. More about the customer interaction with the Software management system will be explained later in this paper.

The complete system communicates with each other using WCF communication using different data contracts as we presented in project #4. The message format will be XML or object reference. Based on the requirement and speed we choose XML or object reference while designing the message specifications.

Few of the critical issues we may arise during out design and development of software Management system are as follows.

1. How does the client identify which repository/test harness to connect? How to define the server destination address.
2. How to maintain multiple sessions of single user client with different servers.
3. How to schedule the server sync, as we have multiple repository servers, how a server does knows that a particular package has been updated.
4. What kind of error handling to be done so that we will not lose the data in the worst case.
5. As the repository is expected to last for decades and people tend to access the oldest

versions often, how can we make sure that the access speed for old packages and new packages will remain same.

6. How are we planning to log the access records, for example: user may access different servers from different locations at the same time? How are we going to keep track of all these activities?

With the advent of IoT (Internet of things) the number of devices on the internet are increasing tremendously. The number of connected devices on the internet doubled in the last three years which means if there were x number of devices in the market till 2012 now the count has become 2x in just a span of three years. The data generated by these device is increasing rapidly. Take the example of social network, the data generated by the social network in the last three years is almost equal to the data that has been already existed before three years. (The values presented above are just estimates from my reading in internet*)

In the technology market there is always demand to process the large amount of data that is present in the database, the existing RDBMS databases are not efficient enough to process this large volume of data. SQL has not been efficient in handling this large volume of data. Hence there are new database that have come in the market which does not use the conventional SQL but proven to be efficient handling large volume of the data. Since these databases doesn't use not only the conventional SQL but also other query methods, these databases are named as NoSQL Databases which means not only SQL.

In this project to solve the industry problems of accessing the large databases we will work on developing one of the NoSQL database. In the market there is long list of the databases, however they are not suitable for all applications, Hence we are developing a new database with which we can come over all the problems and can be used as single for any type of applications. For example mongoDB is used for large document storage whereas Cassandra is used for key-value database. Hence the existing NoSQL databases have their own merits and demerits. But we will try overcoming all these drawbacks and develop one single database using C# language and .Net framework. While doing so we will also try to retain some of the good features of the traditions SQL database. The following are the list of NoSQL databases that are available in the market with their drawbacks [4].

1. Key / Value Based
   e.g. Redis, MemcacheDB, etc
2. Column Based
   e.g. Cassandra, HBase, etc
3. Document Based
   e.g. MongoDB, Couchbase, etc
4. Graph Based

e.g. OrientDB, Neo4J, etc

As we talked earlier our goal in this project is to develop a database which can perform better in all the above scenarios and better than the above databases.

We will also use C# and .Net frame work in this project to develop the database. Visual studio 2015 will be our IDE to write C# programs with .Net framework.

We will also use the famous 3Tier architecture to implement the requirements. The above all implementations will be explained in detail in this document.

The tool that we are going to develop will serve the following purposes

1. Read XML data file with instructions from user and save it in key value database
2. Up on Query from user in the form of XML file the tool will read from the database and provide XML file to the user
3. Read the individual key value data(Hardcoded) from user and save it in database
4. Add the key values to the database up on request from user
5. Remove the key values from the database un on request from user
6. Modify the key values from the database up on request from the user
7. Create the child relationships between the key values when requested from user in metadata
8. The client tool will help us to communicate with the server
9. There can be multiple client applications running parallel
10. User will be provided with the options to select number of clients he want to run
11. The server and client will have performance measurement modules which calculate the time taken for the service answered time and service requested time respectively
12. The client can read the input from the XML and send requests to the server

The main users of the tool will be the developers, programmers, architects, Quality analysts, Quality Auditors, Project Management Office (PMO), Testing Team, Client testing team and other stake holders of the software project etc.

The database will also be accessed by the other applications online and offline from different locations. Hence the database should be able to communicate with other applications using APIs error free.

The main stakeholders of this project would be Dr. Jim Fawcett as guide and requirements elicitation, Ravi as software developer, software Architect and project manager, Microsoft for providing .Net framework.

2. Architecture of the Storage Collaboration Federation System for Software Federation

The basic architecture of the complete system is shown in the below Figure 1. However few components are hidden to provide only abstract overview and not to confuse the reader at this stage. This section provides basic understanding of the Software management system and its components. The details of each system and the interfaces between each system will be dealt in detail in the next coming sections.

We follow the popular MVC architecture and our system acts as desktop application and also can be tuned to act as the web application. Making the application available for both the desktop as well as the web application provides the flexibility for users to work from any devices with least hardware features. We develop the software in such a way that the maximum processing will be at the server end. Only the check in and check out involve the data transmission, apart from these no major data transmission will not take place and everything is handled at the server end hence providing the high access speed.

As part of our development of one of the most complex system to have storage management system, Test harness, Requirement database, Delivery server, Software collaboration system we propose the architecture as shown in the above figure.

The Client interacts only with the Executive management system, the executive management system will provide the required GUI interface to the client. The interface will include multiple tabs such as following.

- Access the repository (Check in, Check out, Reserve version for project etc.)
- Access the software collaboration system(Setup project meetings across globe, Archive the records for future use, Virtual Display Systems)
- Access the Requirements Management server(customers access the requirement management server to create new requirements, Architect create a new packages or reserve the versions for project)
- Access the Test harness server(Test the software against the requirements
- Access the Build server
- Message Passing communication system
- Review Data base server

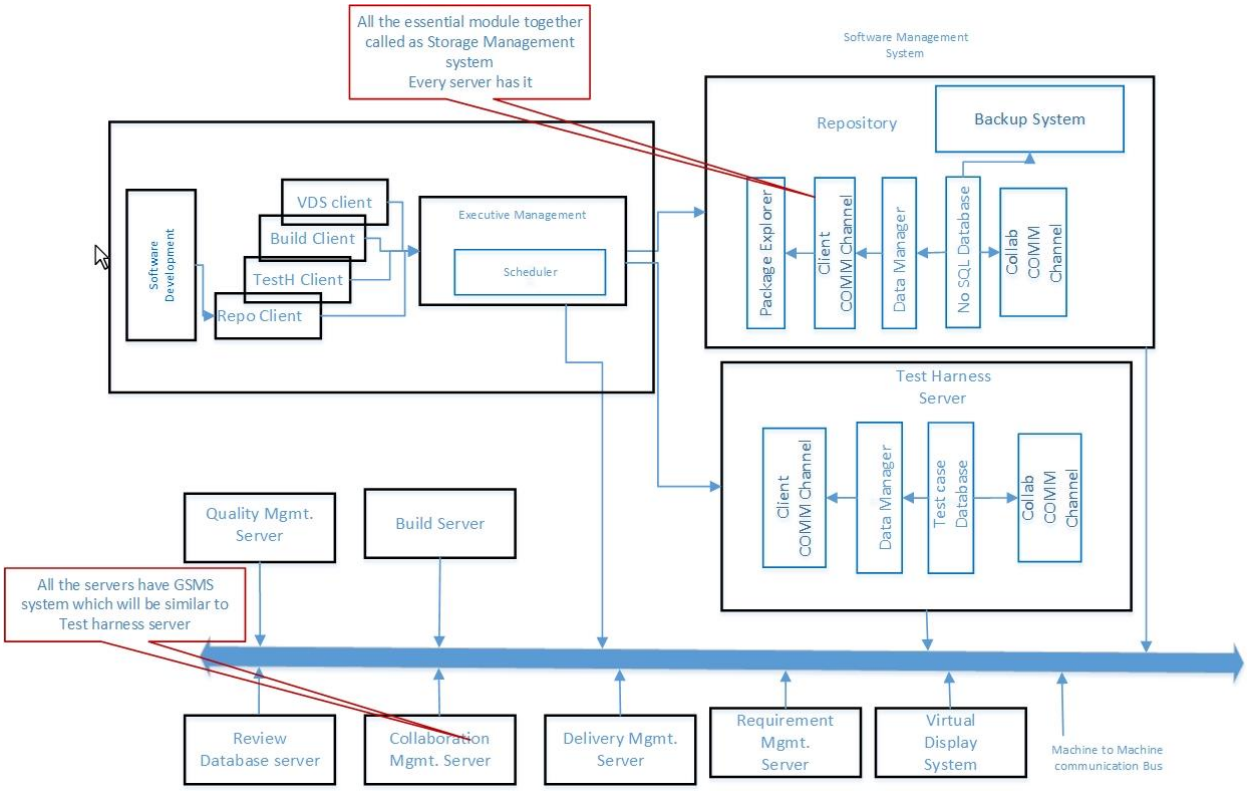


Fig 1. *Overview of the Architecture for Storage Management systems for Software Collaboration Federation*

2.1 Figure Explanation:

In the above picture we have the overview Architecture of Storage management system the storage management we design has all the components like the Repository, Test Harness server, Requirement management server, Delivery Management server, Collab. Management server, Review database server, Quality Management server, Build server and etc. However very brief explanations of these modules will be provided in this section and in the later sections all the deep details of these modules will be discussed.

We see from the above diagram that the communication with the repository happens in two directions one from client another with the communication Bus which is connected to other server like Test harness etc. The repository can get requests from the client or from the other machines. For example if a Test Harness need to do testing for particular package which is requested from the client independent of the repository. Then the test harness requests for the package information from the repository, the repository sends the requested information to the test harness server. The test harness server does the required operation by retrieving the test cases of the package, once the testing is done the result is sent to the client. In case if the request is sent to the repository then the repository will redirect the request to the test

harness server by providing the required information via the communication Bus.

In the similar fashion the client can send the request to either the repository or to the respective server.

2.2 Functions of Modules:

2.2.1 Repository:

Repository is the heart of Software Management System. The software management system uses the already available and developed NoSQL Database from project #2 and communication support from project #4 The Software management is responsible for maintaining the project software/code baseline.

To perform all the assigned responsibilities the repository will have the following modules further. We provide very brief introduction here and in the later sections we will explain in depth.

This provides the basic features like view the code, view the packages attached to a project, analyze the dependencies, Check-in code, Checkout code, Reserve version for a delivery. The repository will also be responsible to take the backup of the code bases available periodically whenever required.

The repository will also handle the following requests once the operations inside the repository are done. Let us consider that a client requests for a check in of file. The server does the following actions.

1. Before check-in operation, the repository monitoring module checks if all the syntaxes are correct or not. If the syntax is not correct an error is thrown to the client user who is requesting this operation.
2. Once the check in is done the repository module will set the status of the package to Check-in. previously the status of the package was checkout. Now the repository will request for the build server to give build. Once the build is done the errors are checked automatically and entered into the error file of the package. If there are no errors then the quality status in the quality analyzer server is update with the quality status of the package.
3. Once the quality checks is done, the repository sends the request to the Test harness server, The test harness server collects the package information of the server and checks if there is any test cases updated if the test cases are not updates the repository will notify the user and the integration process is terminated. If the test cases are available and the testing is successful then the repository will notify

the requirement server. The requirement server will check the requirement number that has been integrated and notify the customer who requested for this particular change or enhancement.

More on the repository we will see in the later sections.

2.2.2 Test Harness Server:

Whenever a package is created in the repository, a new test case folder will get created for each package. Each test case package will have the interfaces and the interdependencies mentioned in a separate file. Whenever a new check in is committed in the repository the test harness server will be called in the backend. The test harness server will do all the static tests and set the status of that particular package to "Tested". We will see the all states of the package in the later sections.

2.2.3 Build Server:

Build server is triggered by either client or by the repository. The client triggers when the user want to check only build specifically by providing the package name. The build server gets all the package code files required from the repository and builds the software. If any error then the user is notified or the repository is notified and the repository will reply the client user who requested it.

Once the build is through the software will be checked in by the repository.

2.2.4 Quality Management Server:

The Quality management is server is called from the user or from the repository or from any other third party server. The user can run the Quality Management server only when

2.2.5 Review Database server:

In software industry to maintain the quality standards peer review and team leader reviews are very important. Before delivering the software to the customer the quality team will hold quality audit and check who has conducted the review for particular package or for particular requirement. So to store all these information in the storage system we implement the Review Data base server. The motivation to provide the Review Database server is to make sure that the review has given the comments and the developer has taken care of all those points. In the later point after delivering the software if there are any bugs reported, there is always chance to check the review points if this issue has been addressed during development. If the issue has not been addressed we can add these points to the LLBP (Lessons Learnt Best Practices) using which we can make sure the similar bugs will be taken care during development.

In short it is always best to have Review Database for software development.

2.2.6 Requirement Management Server

The Requirement Management Server is mainly used to keep track of requirements provided by the customer and are commonly agreed. Every delivery say 'D' will have a set of requirements. In complex software systems closely related requirements are grouped and put together as single requirement. Each check in is related to single requirement.

A delivery D is defined as ={R1, R2, R3, R4…..Rn} and it is stored in the Deliver server which will be explained in the next section.

Each requirement is tagged to one version of the package. Let's say we are delivering four packages P1, P2, P3, P4, planned for delivery D = {R1, R2, R3, R4}. The following can be the one of the possible relationships.

P1= {R1, R2}

P2= {R1, R3}

P3= {R3, R4}

P4= {R4}

The delivery can be done only if all the packages are developed and build, quality checked and tested and reviewed by peers and the tech leads for that particular requirement. All these information will be tracked by the delivery server either on the intimation from repository or from other resources. An automated email is sent to the customer who requested the particular enhancement or feature. This kind of implementation will be in accordance with the CMMI Level 5.

2.2.7 Delivery Management server

As discussed in the previous section the Delivery is tagged to few requirement that are required for customer. Each requirement is tagged to a package where the changes are done. The Request is called a "RequestHandle" and is tagged to a package. Each package can have multiple requests and each request can have multiple packages. Once the Request Handle is created it is assigned to a developer. When developer check-in the software

2.2.8 Collab Management Server

Collaboration management server host the Virtual Display system, Chat system for employees, conference system and other meeting related software such as Webex, Microsoft net meeting etc. This server is completely dedicated to the support functionality. This server also will have miscellaneous central management details such as Team performance, Team structure and status of the projects etc. which interests the project management office (PMO).

2.2.9 Data Manager

The data manager is the internal component of every server. The main responsibilities of the data manager is to convert one form of data into another forma in such a way the NoSQL database understands it.

2.2.10 Virtual Display system

Virtual display system can be part of Collab Management server or it can be hosted separately based in the loading and the usage.

Critical Issue: The latency of the virtual Display system is very crucial and hence the system should be designed in such a way that the latency should be in milliseconds.

Solution: the communication system of this server should have a high resolution timer which will help to keep track of the latency, According my experience most of the time the latency is because of the inaccurate calculations. This situation can be better handled using the event based scheduling of the tasks that a Virtual Display system. The advantage of the event based scheduling is that the unnecessary looping of the routines is reduced. For example if we schedule the same instruction in the while loop the while loop will keep on executing the instructions irrelevant of the update of the information. Which means if a message is updated five minutes ago the while loop keeps on checking the queue is updated or not indefinitely, when we use the even based scheduling the instructions are executed only when the memory is updated which means when the message is received. And then the execution stops.

3. Message structure in SCF

3.1 Executive Summary

In SCF system the communication between different systems is classified into four types

1. Machine to Machine Communication
2. User to Machine Communication
3. Machine to User Communication

As it is already discussed about different types of systems in the first section of this document, the communication Message structure differs from each system to system. For Example: Repository system may have to communicate with the client and respond in different format that a client understands and when the repository system wants to convey some message or data to the build Server and Test harness server the same applies.

Though there are many types of message passing techniques available in the market we will discuss the following types of techniques.

1. XML based Messages (For messages)
2. JSON based Messages (For Data transmission)

It is very much important to standardize the communication Message structure to minimize the errors while implementing. The following are the types of messages that a system to system happens.

1. Control Message(Admin and other instruction)
2. Data Message (File transmission)
3. Query Message(To send Request)

As SCF is equally dependent on the message exchange and data transmission, it is very much required to choose suitable

XML is widely used for human readable content, which means the user can actually get 100% information by looking at the XML file. XML is text based and position independent.

The following are the advantages of the XML

1. Human Readable
2. Easy to program for developers
3. Very wide source code for processing the XML content
4. Self-Describing data

The following are the disadvantage of the XML

1. Long processing time for the processor probably higher number of CPU cycles utilized
2. Fat structure which means it provide more overhead in terms of maintenance, Parsing is not error free
3. Difficult to map to the object oriented design

As an alternative we have JSON which is very flexible than the XML. It is said that JSON is 50 times faster than the XML.

The following are the advantages of the JSON over XML

1. Very light weight
2. Easy for processor to process, Requires less CPU cycles
3. Easy to understand when compared to the XML

We will use both XML and JSON whenever required.

3.2 Organizing Principles

In SCF we design the following message structure.

3.2.1 Sending the status messages

The SCF server communicate with each other using the status messages, The status messages are used to communicate between the servers if the operation requested by the sender is completed or not. To keep the communication between all the servers simple and standardized we use the single common status message template which will be discussed in the later subsections.

An example of the status message looks like below.

```
< MessageElement >
 <Message_type>status</Message_type>
 <Result>success/failure</Result>
 <Operation-type>Build</Operation-type>
 <Detailed_report>Ex: The operation failed because</Detailed_report>
 <PackageName/File_Name/Project_Name>DBElement.cs</PackageName/File_Name/Project_Name>
</ MessageElement >
```

3.2.2 Passing the data

The SCF also uses the data passing messages in few scenarios such as when sharing the error reports for display purpose, displaying the list of packages in a project etc.

The message which handles the data will have multiple fields which are required to rebuild the complete package structure or something similar, sometimes the data can be in the markup language such as Html, XML etc.

The below is the sample of the data message structure we will use in this project.

```
< MessageElement >
 <Message_type>Data/Display</Message_type>
 <Display>
       <parent_menu>project 2</parent_menu>
       <child_menu>package1 etc. </child_menu>
 </Display>
 <Operation-type>Build</Operation-type>
 <Detailed_report>Ex: The operation failed because</Detailed_report>
 <PackageName/File_Name/Project_Name>DBElement.cs</PackageName/File_Name/Project_Name>
```

```
</ MessageElement >
```

3.2.3 Passing the files using JSON

To successfully transmit the files between two systems we define the message flow as below.

Sending Metadata and files in separate requests

Step1: Receiver sends the metadata to the Server.

Step2: server stores the metadata and generates an unique URL to which files should be uploaded, sends the URL response to the receiver.

Step3: The client uploads the file to the server using the received URL.

Sending the Metadata and files in the same request

We can also send the Metadata and the files in the same request which we will see, but the only dis advantage is that the network performs well when the data is split and send.

The following is the method that are used to send requests

Step1: The client encodes the file in the form of data in to the payload of the message structure.

Step2: The clients sends the encoded wrapped file data in the form of data message.

Step3: if the acknowledgement is not received the client sends the data again repeatedly until it receives the response from the server.

3.3 Users and Uses

The messages are used by every sever system in the SCF to communicate with other servers. The main users of the messages are as below.

Client

Client uses the message structure to communicate with Repository to send the file and to receive the files, with Test-Harness server to conduct test on the packages and projects which are integrated newly and are already existing, with build serve to send the commands to build server to build the software and send the reports. The messages are used to perform the following operations mainly.

1. Check-in Package
2. Checkout package
3. View packages in the repository
4. Create new packages and delete the existing package in the repository.

Repository

Repository uses the message structure we are going to propose to receive the requests from the clients and other SCF components like Build server and Test harness server.

The repository also uses the message structure to send the replies to the requested clients and other components. The repository contains huge data files which need to be send to the requested clients. Thus is uses the same message structures that we are going to define below.

1. Check-in Package
2. Checkout package
3. View packages in the repository
4. Create new packages and delete the existing package in the repository.

Test Harness Server

Test harness server uses the Message structure to receive the command to execute the test cases in the server and also to receive the test cases for a particular package. Hence the test harness server has some features like repository also where it maintains the database of the test case files.

In short test harness uses all the message types that we are going to define below.

1. Test command Message
2. File Message structure(To receive packages from the repository)
3. Data Message structure (To send reports to the clients)

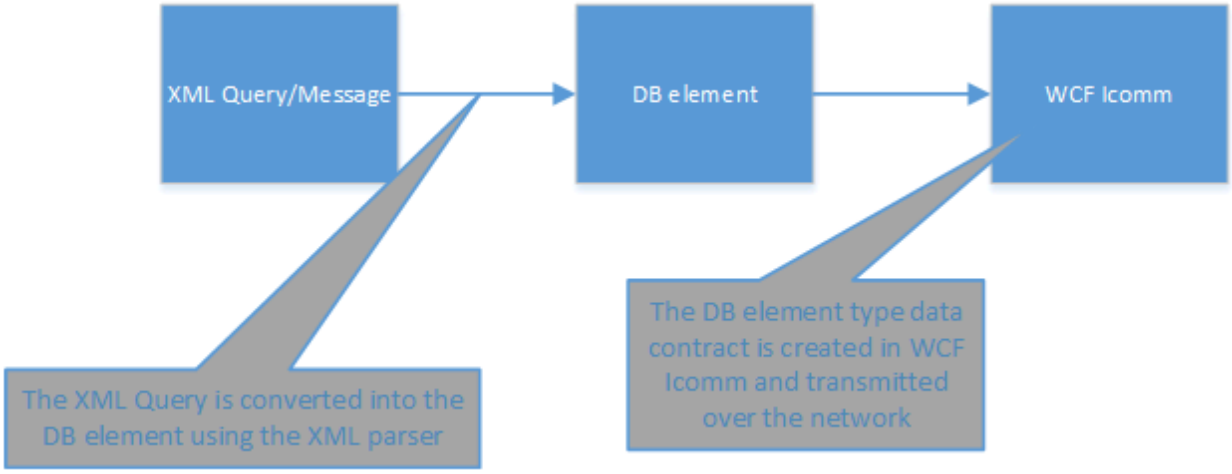
Build server

Build server contains build environment and hence it receives the commands to execute build on particular package. So build server uses the build command message. And also as build server does not contain any packages stored it uses the file Message to get the required packages to the build server before giving build.

1. File Message structure (to get packages from repository)
2. Build Command Message structure
3. Data Message structure(To send error notification to the requested clients)

3.4 Message Structure

We see two structures of messages one is Message passing without the data and another one is file transmission with data. Both of them can be done using the JSON or XML. But since our project #2 and project #4 are developed using the XML message passing we will discuss about it in detail in this section.



Instead of sending the raw XML file over the internet we send the DB Element object and the receiver will read the required information using the object. This is tested and used successfully in the project #4.

3.4.1 Sending simple status message using the XML

```
<?xml version="1.0" ?>
<root>
  <menu>File</menu>
  <commands>
     <item>
         <title>New</title>
         <action>CreateDoc</action>
     </item>
     <item>
         <title>Open</title>
         <action>OpenDoc</action>
     </item>
     <item>
         <title>Close</title>
         <action>CloseDoc</action>
     </item>
  </commands>
</root>
```

The above request can be sent using the JSON format then the message structure will look like below.

```
{
  "menu": "File",
  "commands": [
      {
          "title": "New",
          "action":"CreateDoc"
      },
      {
          "title": "Open",
          "action": "OpenDoc"
      },
      {
          "title": "Close",
          "action": "CloseDoc"
      }
```

```
   ]
}
```

We can see form the above that the message passing is simple using the JSON format instead of XML.

3.4.2 Simple Query Message format:

The following is the simple Query Message format in both XML and JSON. However we will use the XML format initially in this prototype and later we will extend it to JSON in the later enhancements.

In our previous discussion we have already seen JSON is very better performing when compared with XML.

```
  < MessageElement >
    <PackageID>DBElement</PackageID>
    <Category>Project1</Category>
    <Description>Project2</Description>
    <Query>QueryKeyMetadata/Query/QueryDesc/</Query>
    <File_Name>PackageList_modified.cs</File_Name>
    <searchMetastring>Ravi</searchMetastring>
    <searchTime>datatime</searchTime>
    <Author>Ravi</Author>
    <Country>NewYork</Country>
    <Time>2015-10-07T21:41:41.4911909-04:00</Time>
    <Payload>This package contains the following interfaces, private void
Test_R1()</Payload>
    <Children>
      <Child>
        <ChildID>Project1</ChildID>
        <ChildDate>1997-08-25T00:00:00</ChildDate>
      </Child>
      <Child>
        <ChildID>Project2</ChildID>
        <ChildDate>1997-10-03T00:00:00</ChildDate>
      </Child>
    </children>
  </ MessageElement >
```

Discussion:

In the above XML format we can modify to send the Queries, Check-in, Check-out etc.

3.4.3 Message structure for check-in:

The following message structure is used between repository and the client. The client sends the check-in request. While sending the check-in request the client mentions the version number in the tag <version> along with version comment <version comment>. If there is not mention of the version then the server assume the latest possible version as default and check-in.

```
  < MessageElement >
    <PackageID>DBElement</PackageID>
    <Server_address>127.0.0.0/https</Server_address>
    <Category>Project1</Category>
    <Description>Project 5 demonstration</Description>
    <Message_type>Check-in</Message_type>
```

```
    <File_Name>DBElement.cs</File_Name>
    <Version>1.1.1</Version>
    <Version comment>Delivery of software</Version comment>
    <Relationship-children>
       <Child>child1</Child>
        <Child>child1</Child>
    <Relationships-children>
    <Relationship-parent>
       <Parent>child1</Parent>
        <Parent>child1</Parent>
    <Relationships-parent>
    <Author>Ravi</Author>
    <Country>NewYork</Country>
    <Time>Present_time_data</Time>
    <File_path>/Desktop/path/input path</File_path>
  </ MessageElement >
```

3.4.4 Message for Checkout:

The following is the message structure the client sends to the repository to check out the package. The client sends the version number in the tag <version> if this is left blank then the server sends the latest available package which is tested. Note that the server will not return the untested versions.

```
< MessageElement >
    <PackageID>DBElement</PackageID>
    <Server_address>127.0.0.0/https</Server_address>
    <Message_type>Check-out</Message_type>
    <File_Name>DBElement.cs</File_Name>
    <Version>1.1.1</Version>
    <Version comment>Delivery of software</Version comment>
    <Author>Ravi</Author>
    <Country>NewYork</Country>
    <Time>Present_time_data</Time>
    <File_path>/Desktop/path/outputpath/</File_path>
  </ MessageElement >
```

3.4.5 Message for View Package

The client requests for a project or a package to view via the GUI which we will discuss in the later sections. The server reads the request and sends the list of packages satisfying the query. The GUI of the client displays the package structure on the GUI. The GUI provides the birds eye view and the explorer view to view the list of packages and their dependencies.

This query can be extended for searching for package with certain string values etc.

```
<MessageElement>
    <PackageID>DBElement</PackageID>
    <Server_address>127.0.0.0/https</Server_address>
    <Message_type>View</Message_type>
 <PackageName/File_Name/Project_Name>DBElement.cs</PackageName/File_Name/Project_Name>
  </ MessageElement >
```

3.4.6 Message for Test Harness server

The following is the message that either repository or the client sends to the Test harness server. There can be two types of requests one is for conducting the testing another one is to

get the test cases for viewing purpose on the client GUI. Any server in the SCF can send this request to the client.

```
< MessageElement >
    <Test-PackageID>DBElement</Test-PackageID>
    <Server_address>127.0.0.0/https</Server_address>
    <Operation-type>Test/getTestcase</Operation-type>
    <Test-PackageID>DBelement</Test-PackageID>
    <Message_type>View</Message_type>
 <PackageName/File_Name/Project_Name>DBElement.cs</PackageName/File_Name/Project_Name>
 </ MessageElement >
```

3.4.7 Message for Build Server

The client sends the command message to the build server. The build server on receiving the command reads the packages or the project from the repository and builds the software. The build server replies the result with warning and error details if any. If not the build is successful and report is sent to the client or whoever requested the build execution.

```
< MessageElement >
    <Build-PackageID>DBElement</Build-PackageID>
    <Server_address>127.0.0.0/https</Server_address>
    <Operation-type>Build</Operation-type>
    <Message_type>View</Message_type>
 <PackageName/File_Name/Project_Name>DBElement.cs</PackageName/File_Name/Project_Name>
</ MessageElement >
```

3.4.8 Status Message

The status messages are the data messages exchanged across all the servers. The status message can have the status of the operation such as build and test running status. In few cases the status message also can be used for all the purposes just by using the

```
< MessageElement >
      <Build-PackageID>DBElement</Build-PackageID>
    <Server_address>127.0.0.0/https</Server_address>
      <Operation-type>Build</Operation-type>
    <Message_type>status</Message_type>
 <Result>success/failure</Result>
<Detailed_report>Ex: The operation failed because</Detailed_report>
 <PackageName/File_Name/Project_Name>DBElement.cs</PackageName/File_Name/Project_Name>
   </ MessageElement >
```

4. Core Services and Policies

4.1 Services

As we have already discussed the main purpose of having the Software Collaboration Federation is to help the software development teams to provide single point of serve for all the needs of the software development. Though there have been basic services like repository, build serve available at present the difficulty is integrating them and reducing the amount of effort required by the software developers.

To efficiently use the SCF system by all the software developers it is very much required to carefully formulate the services that the SCF provide for its users. While formulating the services the service time should be taken care since the service time is directly propositional to the wait time of the developer which is in turn directly proportional to the investment by the company to pay the salary to the employee.

Hence keeping all these in mind the following core services are finalized.

1. Data passing service
2. Queries
3. Communication
4. Notification
5. Navigation
6. Global Management
7. File passing service
8. Centralized Notification services
9. Remote Storage Administrator support
10. Ability to introduce new Servers
11. Information access for Disaster Recovery
12. Real time synchronization with other servers and reporting
13. Easily upgradable
14. Encrypted and secured information
15. Loose coupling between systems
16. Multi Language support
17. Easy integration with existing systems
18. Centralized Logging service

4.1.1 Data passing service

The data passing service is used by all the servers and client in the SCF. Hence we discuss briefly in this section and in deep in GSMS (Generic Storage Management System) section. The data between the components of the SCF is handled in the form of data contracts defined in the WCF communication module. As per the analysis there are three types of data that the servers exchange

Exchange of messages with very less payload such as status messages.

Exchange of notification reports
Exchange of files between two nodes
Exchange of Messages

4.1.2 Notification Service

Notification service is an integral part of each and every server and client in the SCF. Notification service is triggered when a task being executed in the server wants to inform the source of the request about the status. A notification is a message that is displayed to the user outside of the server application. When the server tell the system to issue a notification, it first appears as short message if the user is currently logged in or else the notification will be logged both by the client and the server.

In SCF a notification can provide different kinds of information, however we concentrate on a notification per request. The following are the types of notification that a server can issue to the client or user who is executing the request.

- Email Notification
- Local and Remote notifications
- Delivery Notification
- Action notification
- Event Notification

Email Notification

The SCF provide email notification service, the email notification service is the way of communicating that action has been completed. When a user performs an action the email is sent to the pre set receivers that the action is completed. The email notification is completely configurable as per the user request.

However few notification settings such as security notification cannot be overwritten by users, special privileges are required.

Local and Remote notifications

The SCF components also provides the Local and Remote notifications.

Local Notifications

Local notifications are issued to notify in the server itself, As there may be users logged into the server accessing through remote login the notifications issued as local notifications are displayed to the logged in user.

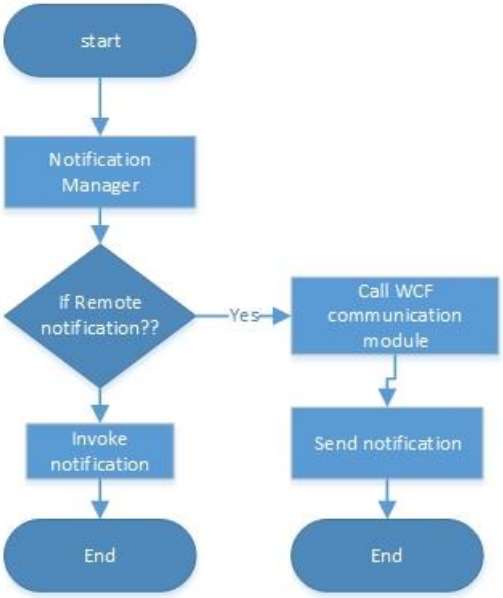


Figure: Local and remote notification using the delegates and message passing.

Remote Notifications

Remote notifications are issued to notify the users who logged in through the client, the notifications are sent using the following message structure. All the components in the SCF supports both the local and remote notification. Even clients support both local and remote notification.

Delivery Notification

Delivery notification is configured for every project delivery that is created for the project. A delivery can be a single package or for multiple package. The users who want to be notified when the package is available for delivery can subscribe for the delivery version of package.

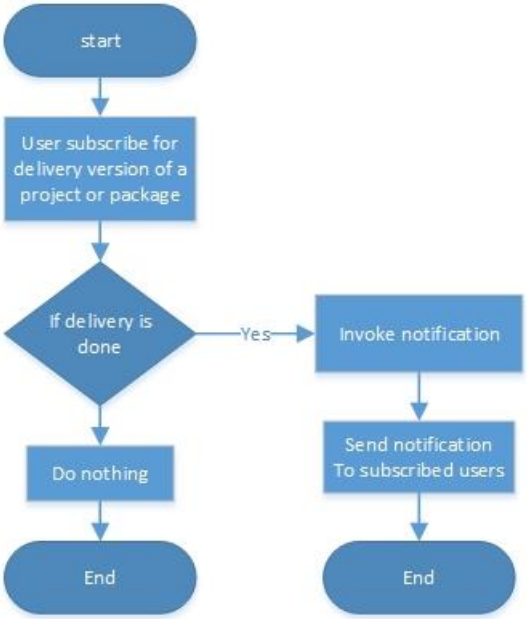


Figure: Delivery notification for SCF servers

The following activity diagram shows the process how delivery notification is created and how the users can subscribe to the delivery.

Action notification

Action notification is created by users when they want to get notified whenever an action happens. This kind of notification are set when certain security critical action takes place in the SCF servers. For example when particular package is accessed outside of a project when a security breach is performed etc.

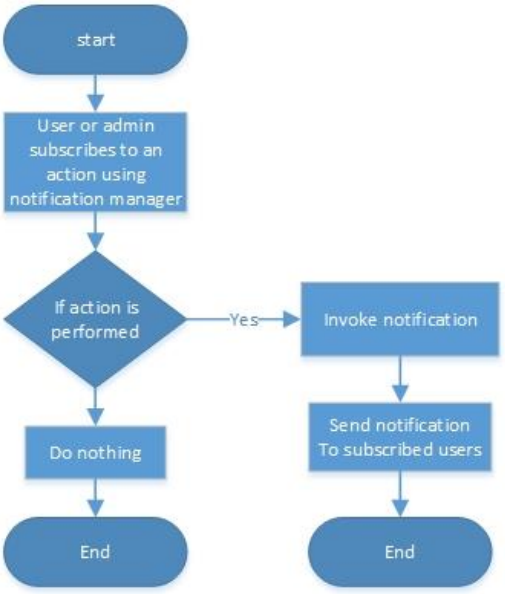


Figure: Action notification for SCF servers

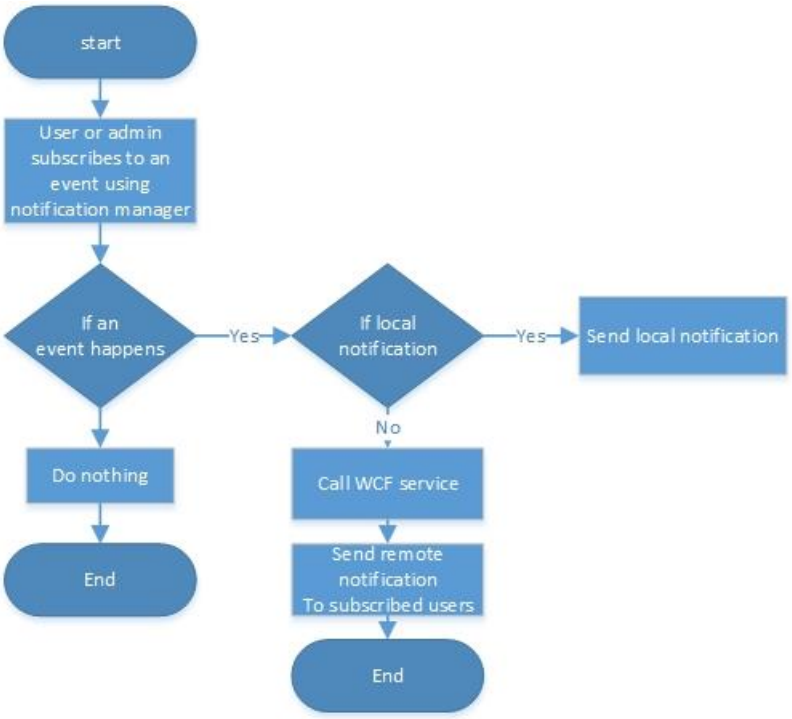
The above diagram shows the activity of the action notification. The action notification is similar to the and for both delivery notification and action notification we use delegates and for delivery notification

```
public delegate void NewuserAddDelegate(User Username1);
public void AddUsers(NewuserAddDelegate Newuser)
      {
          foreach (User b in list)
          {
             Lisuser.Add(b)
                  // Calling the delegate:
                  processAdduser(b);
          }
      }
```

The above is the sample code snippet how the delegate is invoked to send a notification to the user when a new user is added to the list of existing users.

Event Notification (Remote and Local)

Event notifications are similar to the action notifications as we discussed in the earlier sections, except that we use generic delegates for event notifications. The notification manager checks if he notification is for local or remote. The following activity diagram shows how the local and remote servers are called.



The following code shows how an event can be created and how the users will subscribe to an event remotely.

Creating an event

```
public interface notificationInterface
{
   event MyDelegate anEvent;
   void fireEvent();
}


public class notification: I
{
   public event MyDelegate anEvent;

   public void fireEvent ()
   {
      if (anEvent != null)
         anEvent();
   }
}
```

Subscribing to an event:

```
I i = new notification ();
i.anEvent += new MyDelegate(f);
```

Communication

4.2 Policies

The policies of SCF define how the organization of the service is done. As we have seen few services in the previous section, there are policies which govern these services in direction.

The following are the list of policies that govern the SCF

- Ownership
- Versioning policy
- Test execution policy
- Build policy
- Notification
- Relationship management between Repositories, Build server, Test Harness, Requirement Database, Review Database etc.
- Integration and Delivery policy

Ownership:

In repository the packages can be created by anyone who is part of the project. Hence whoever creates the package they will be the owners for only that version and if another user

creates a revision over the package then the ownership will change. However the owners will cannot modify the versions of the package after check-in of the package. And also no one can check-in the packages which are already checked-out by other users.

The super user will be owner of all the packages and it will be the administrator. In project only one super user be present and that super user access is limited to the project itself. The following table shows the list of users and their scope in the repository.

User Type	Scope of ownership	Access type	Outside project scope
Normal User	Only packages checked out by user	Normal	Nil
Super user	All packages of a project	Admin for project	Normal user
Admin user	Complete repository	Admin for Repository	Admin

Table: Ownership in SCF server(this holds good for Test harness server and build server)

Versioning policy

Versioned repositories store an application's repository assets—that is, assets that are deployed as repository items to a SCF repository in the production environment. In SCF the versioning is used in all the servers like build server, test harness server, repository.

In repository the versioning is used to maintain the history of the packages. Every project is baseline and has many packages. Each packages are treated as children of the project.

In repository we use the versioning to store the history of the documents stored in the repository. As it is stated earlier the project is the parent and project will have many packages under it. No package can exist without any parent.

Activities for versioning:

User creates a new version in the checked out state.

X represent the project variant version

Y represent the new enhancement

Z represent the bug fix

R represents the minor change which will not affect the features.

Now if the user want to create a enhancement to this package then the version will be x.y+1.z;0

Now if the user want to create bug fix for this version then the package version will be x.y+1.z+1;0

Now if the user want to modify the checked in code which will not affect the requirements change then the package version will be x.y+1.z+1;1
Now If the user want to take this package into new project with different customer then the version will change to x+1.y+1.z+1;0

State of the Package in repository

The following are the states of the package in the repository.

state	status	version
Checked -in	Checked-in Available(Freeze)	x.y.z;r
Check-out for bug changes	Checked out	x.y.z+1;0
Create edition in server for test build	Checked out edit	x.y.z+1;0
Check-in after changes but no testing	Checked-in-Not available	x.y.z+1;0
Check-in with testing done	Checked-in Available(Freeze)	x.y.z+1;0

In checked out edit stage the user can test and upload the code files to the server any number of times.

Dependency of the packages:

Before check-in the repository or database checks if the dependencies are present. If the dependency are not handles or not present and still the check in is requested then the server then the server throws a dependency error.

Multicheck-in

In any server there are requirements when a user want to check in the multiples packages at a time. During this check in the user will have option to provide the list of packages that needs check-in with the version. If no version is mentioned then the server creates by default a revision and not a new version.

5. User Interaction

In SCF only the client system allows the users to access the SCF. Even though the SCF system is developed in such a way that most of the operations such as build, test are automated using check-in, the users interact with SCF using few services.

The following are the services the user can directly interact using SCF.

- Check-in, check-out, Delivery using repository server
- Viewing the project and package information using the repository
- Give build in the build server specifying the package or project name
- Editing the requirements in the Requirement database using the requirement server
- Requesting for new delivery using the requirement server
- Testing the software using the Test Harness server providing the package or project name
- Sending code for review or/and doing review for software using the Review database server.
- Delivering the software using the delivery server in the SCF

5.1 Check-in check-out, Deliver using repository server

The user once logs in the client as proposed in the client section can request for check-in , check-out and all other operations in the repository. The following GUI designed will show how these operations are done. This is discussed in detail in the client main section.

5.2 Viewing the project and package information using the repository

6. Generic heterogeneous Storage Management Subsystem (GSMS)

6.1 Introduction

In SCF there are many servers implemented to handle different activities but all of the servers require the communication modules and the database modules. Most of the servers (Build Server, Test Harness Server etc.,) in our proposed architecture in the previous section uses commonly the storage system, Data manager system, Event handler, Event logger system and WCF communication system. Hence in this document a new module called Storage Management Subsystem is proposed. To make is easy for the users and developers to use this in many other applications we make this implementation generic. Generic means the proposed system supports threaded applications, process based application and event based application and also this proposed system can be used as normal database or can be used as the Software Management systems such as Repository.

The rest of the section is organized as follows.

- Executive summary of GSMS
- Organizing principles
- Architecture
- Packages in the system
- Activities
- Uses and Users
- Critical Issues
- Critical Analysis
- Views
- Interfaces

6.2 Executive Summary

A generic storage Management system will have the data base we developed in the project #2 and the communication structure we developed in the project #4. This generic storage management system will be used in the following modules in the later section.

- Repository(Software Management System)
- Test Harness server
- Build server(partial use)
- Requirements Management Server
- Review Data base server
- Virtual Display system
- Delivery Management server

As the system that is being proposed is used in as the subsystem in many other modules care should be taken such a way that the system should be modularized and flexible to remove and add the available modules based on the requirement. The subsystem will have all independent module with more flexibility and adaptability.

Before we design and go ahead with the development of Generic Storage Management system we analyze the following types of Architectures.

- Multi-Process Architecture
- Multi Thread Architecture
- Non-blocking I/O Multiplexing Architecture(Event Driven Architecture)
- Combined Approach Architecture

In addition, to effectively design this we can also implement the combined approach which can be much effective to use the Storage Management System for any of the requirements as stated earlier like Repository, Test Harness, Build Server, Requirements Management Server etc.

6.3 Organizing Principles

To effectively design the GSMS, it is very important to have proper request handling system. As the GSMS will have many number of connections from the client and from other servers which are part of SCF.

The complete GSMS is organized in six sub modules

- WCF Client Communication
- WCF SCF Communication
- Authentication and Security
- Configuration
- Event handler
- Event Notification Manager
- Event Logger
- Data storage manager
- Storage management service
- No SQL database

6.3.1 Multi-Process Architecture

Though this is traditional approach this can be considered as an option to design our GSMS system. This kind of system architecture provides a dedicated process for each and every connection that is coming from the clients. Which means a separate stack in the process stack will be created in the server system. While this kind of architecture is used to server the heavy operations, this consumes most of the CPU resources. As every computer has their limitation GSMS limit the maximum number of simultaneous connections.

Firstly creating each process for client will take more CPU resources, so let's say creating a new process takes 200 machine cycles, it is worth doing this if we have a request from client which takes at least 10000 machine cycles. Which means creating a new process to service a client which hardly takes 1000 machine cycles is waste of resources and hence not suggested.

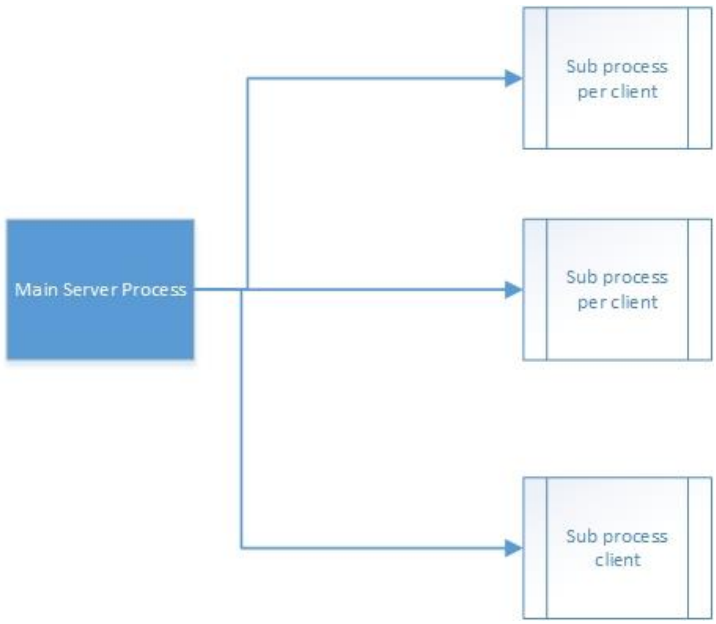


Figure: This figure is included here to provide an idea of how the requests are handled in the proposed system.

What we understood from this is that before deciding to start a new process the GSMS should interpret the request from the client and know if it is really worth. Hence we will make sure that we categorize this in our message structure where we can provide a new parameter which states if the request is heavy or light weight one.

Limitations and Critical Issues:

Issue with Race Condition: There will be always an issue that the two different processes will be requesting for a single resource. While this can create a race condition and can freeze the server without any result. This should be handled in a proper way.

Solution:

To avoid the race condition while implementing the multi process architecture we make sure the data structures we use the server application are not interdependent and implement the code to improve the granularity. Granularity in accessing the shared resources will surely avoid the race condition.

6.3.2 Multi-Thread Architecture

This is a simple approach where a new thread will be created for each incoming connections from clients. This kind of architecture prevents the concurrency and isolation from other client actions thus providing the secured connection which is independent of any other clients. While this is the best and robust, there is always a risk that the server we are going to design might run out of resources by creating the thread stack.

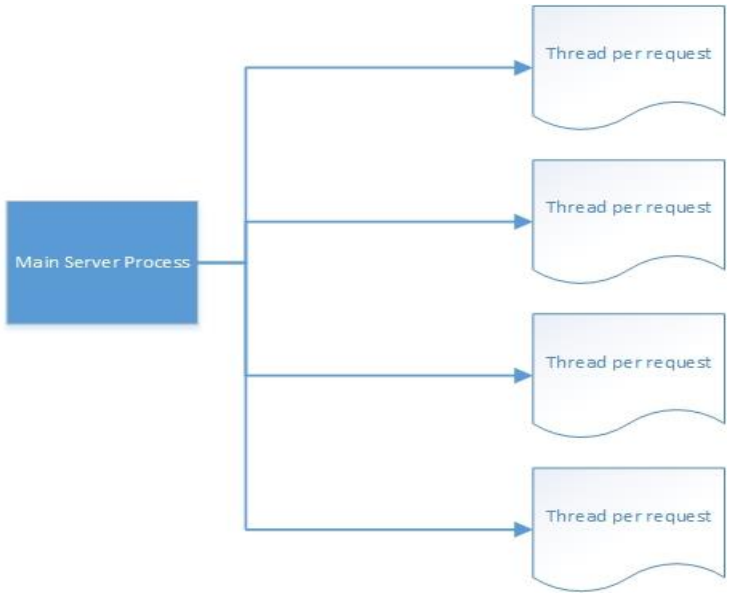


Figure: This figure shows how a multi thread architecture works.

Threads have light weight memory foot print when compared with the processes. One way to enhance the Multi process architecture is to implement the thread wherever it is possible. For example as shown in the above figure we can have the multi process with the multi thread for each process. This provides us the maximum granularity to avid race conditions. Even if the race conditions occurs only one of the client gets affected and restarting the client should be enough. As the individual user will be monitoring the, the user can manually monitor the client requests. And also the server will have the

Scalability Characteristics:

When a client requests for a connection the server or the GSMS will create a thread instead of the process and this is handled independently until the life time of the connection is over. This model of operation provides a decent concurrency when compared with the multi process architecture as discussed earlier in the above section.

Limitations:

The once limitation of the multi thread is that under heavy load the multi thread application consumes most of the memory and processor resources limiting the number of client requests that a system can handle. But still this is better when compared to the multi process architecture.

Solution:

The one solution to reduce the CPU load increasing exponentially is to limit the number of absolute number of simultaneous client connections, this solution may reduce the number of simultaneous client connections but can improve the performance greatly.

6.3.3 Non-Blocking I/O Multiplexing Patterns (Event Driven Architecture)

The other most common approach handling the client requests is the event driven approach which we can use to avoid the drawbacks of the Multi thread and multi process architecture. A common model that is used is mapping a single thread to the multiple client connections. Whenever a new connection comes from a client, the request is added in to the Queue.

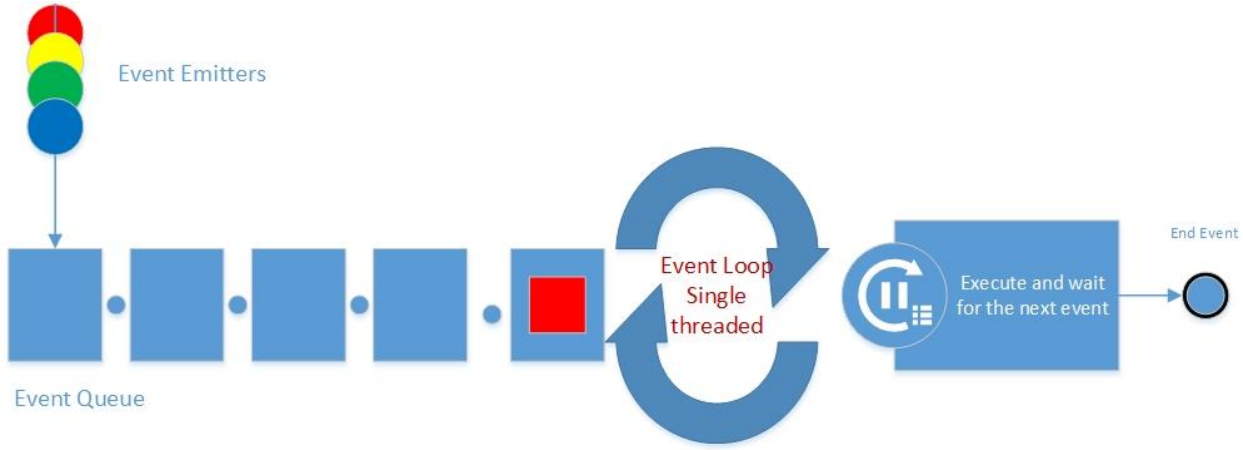


Figure: The figure shows that a event emitted by the event emitter is collected in the Queue and the Queue is read using the loop by the Single threaded application in the GSMS. The event collected is read and executed by the GSMS once the event is services the event is end.

Proactor Pattern

Proactor pattern can be used to implement asynchronous non-blocking I/O operations. This supports completion of the events instead of blocking the event notification. When an I/O operation is done the dispatcher is notified.

Scalability Considerations for Event Driven Architecture:

Since the Event driven architecture is

6.4Architecture of Generic Storage Management System

The architecture of the generic storage management system has a communication module and a NoSQL database which are core of the Generic storage management system.

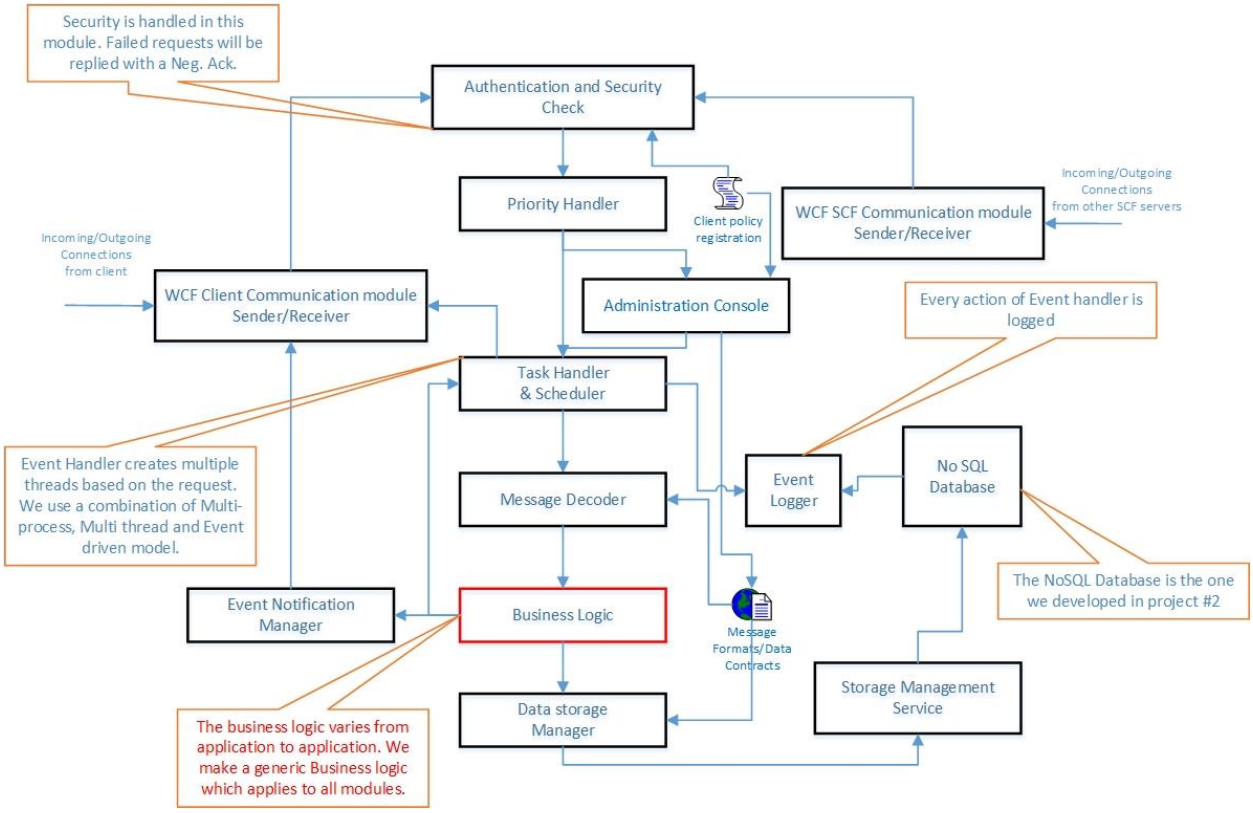


Figure: Generic Storage Management System (Only business logic changes for other SCF servers)

The Architecture of the Generic Storage Management Systems is shown in the above picture. Storage management system is the heart of any server systems in SCF. Storage management system has in build NoSQL Database which was developed in the project #2 and it has the communication system which was developed in the project #4.

6.4.1 Overview of the Architecture

The architecture shown in the above picture explains how an incoming request is handled by the storage management system. Firstly there are two WCF communication channels. One channel is dedicated for handling the requests from the client and other channel is dedicated to handle the requests form the other SCF modules like other Storage Management Subsystems.

The incoming requests are received by the WCF commination modules, the preference is given to the client communication module as this is more important and the user will be waiting. Most of the automated requests will be coming from the SCF communication module hence less priority is given. The incoming requests are collected by the WCF and are forwarded

to the Authentication and security check module. This module will check the credentials of incoming connections and forwards the passed requests to the Administrative console if the request is about changing any of the privileges to the users or related to some administrative tasks. If the request is not about the administrative task then the request is send to the Even Handler.

The Event handler is the main module in the GSMS which schedules all the events. As we have seen in the previous sections how the events are handled the event handler analyzes the request and schedules the even accordingly. If the request is from the existing client connection then the request is threaded to that particular thread. If the request is from the new client connection then the event handler will create a new request this will be discussed in detail in the later part of this section in a separate block.

Once the event is scheduled the request is forwarded to the Message decoder which again decode the message part and sees the operations required. Business logic module will tally the information. If the information is just an insert or delete operation then the operation is forwarded to Data storage manager which converts the data in to the forma that NoSQL understands finally the storage management service is called which is responsible for interacting with the NoSQL Database.

In this project we will be using the NoSQL database for repository where all the software is managed, build server use the NoSQL database to store the build log files and the error logs, Test harness server is used to store all the unit test case, integration test case written by the user. For every server module the Business logic, Data storage manager and Storage management server varies.

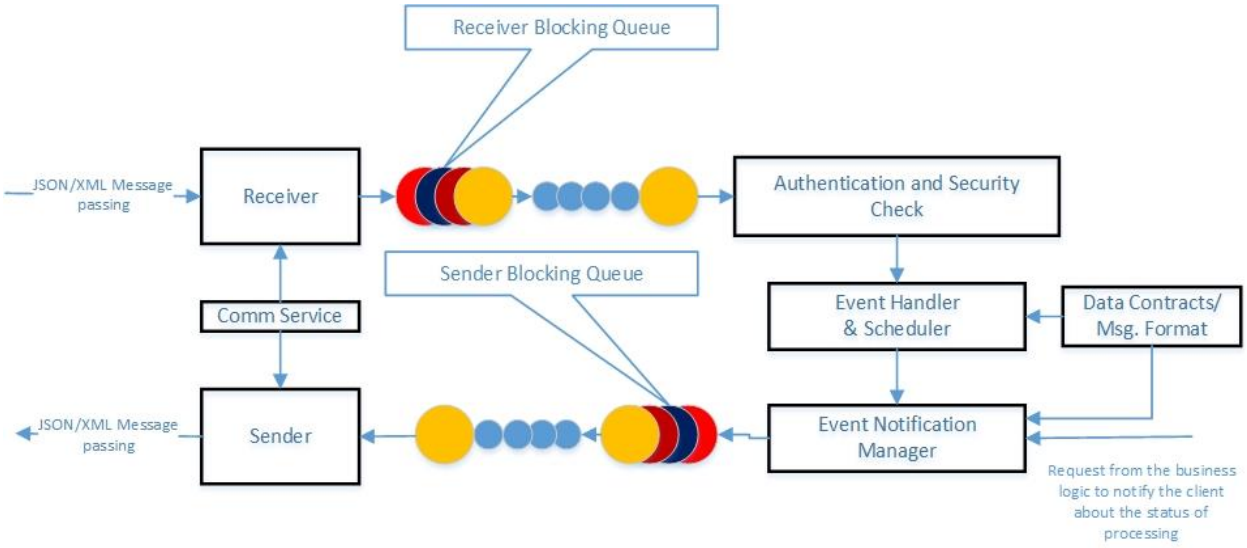
6.4.2 WCF Client Communication channel/WCF SCF Communication channel

Overview

The GSMS receives the requests form two kind of systems one is client and another one is any server part of SCF system. This module handles the requests from the Client applications and as well as other Software.

Structure

The below is the basic structure diagram for the WCF client communication package. The WCF Client module receives the messages of either JSON or XML format. Once the messages are received at the receiver they are queued in to the Receiver Queue. The event handle is scheduled to pick the messages from the receiver queue and schedule them with the data storage manager. The Event handler checks the message format by contacting the Data contracts/Message Format module.



Once the processing of the request is done the Event notification manger will format message according to the result and sends back to the client which requested. The Event notification manger has complete list of clients that are connected and hence it will send once the processing is done.

In case of the Repository the event notification manger will send the notification to the GUI client whenever there is change in the state of the packages.

Activity diagram:

The below diagram shows the basic activity diagram for both WCF client communication module and WCF SCF communication module. The message is received at the receiver and the queue is de-queued and the message is processed by the Event handler/Scheduler the event handler processed the message and takes the necessary action and in the later if it is found that the client requests the acknowledgement then the Event notification is sent to the sender of the particular message.

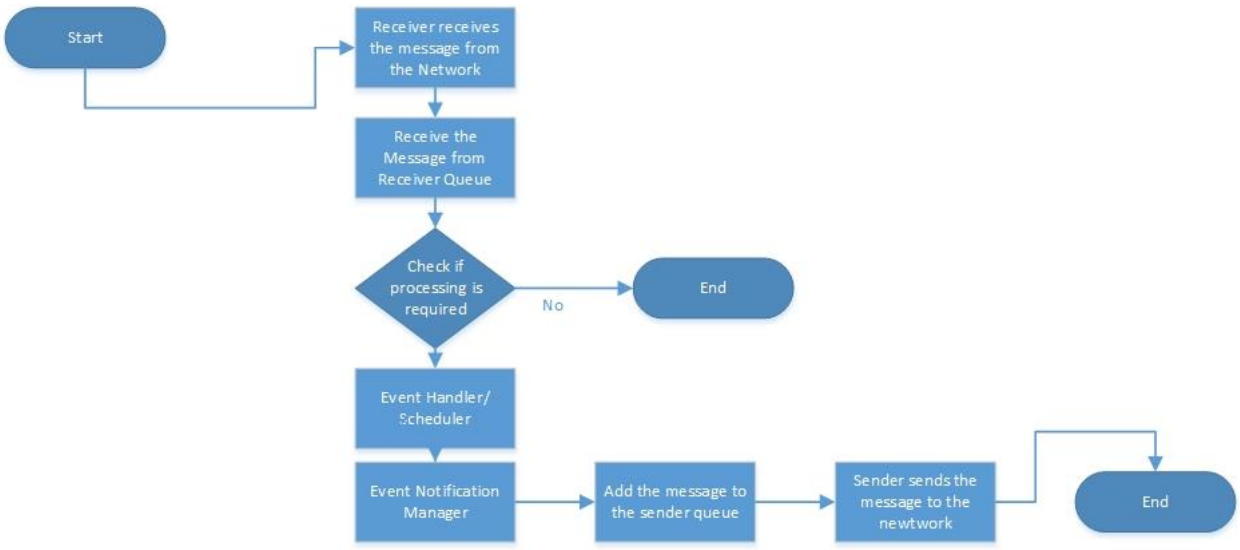


Figure: Activity diagram for WCF communication channel

The event Handler and the Event Notification manager will contact the Business logic of that particular server. It should be noted that he business logic for each and every system such as repository, Build server, Test harness server are different and it depends on the configuration of the GSMS if all these logics are implemented in common or implemented separately when required.

The below code snippet shows how the Blocking Queue can be used to implement this features. The blocking queue is implemented for both the receiver and the sender.

Code Snippet:

```
// static rcvrQueue is shared by all instances of this class

    private static SWTools.BlockingQueue<Message> rcvrQueue =
      new SWTools.BlockingQueue<Message>();

/*This function receives the message from the */

public Message getMessage()
    {
      if(Util.verbose)
        Console.Write("\n  this is CommService.getMessage");
      return rcvrQueue.deQ();
    }

/*this function receives the message by calling the Recevier Function as defined  above*/
  public Message getMessagefromQueue()
    {
      if(Util.verbose)
        Console.Write("\n  calling CommService.getMessage()");
      Message msg = svc.getMessage();
      if (Util.verbose)
        Console.Write("\n  returned from CommService.getMessage()");
```

```
        return msg;
    }
```

Critical Issues:

Queue Full problem: As GSMS is handling two channels in the Storage Management subsystem we may face problem when two receiver queues are filled up quickly.

Solution:

When the queues in the receiver are filled up and no more requests are accepted by the GSMS the best way is to send and negative acknowledgement to the client who is requesting it. Sending negative acknowledgement will in fact consume the extra CPU cycles in this case it is better to design in such a way that if there is no response form the server then the client will wait for some pre-determined time and send the request until it is answered. This is the best solution to avoid the requests getting missed.

6.4.3 Authentication and Security

Overview

This module or package in the GSMS will act as security gate way, when a new connection from the client comes to the receiver, the receiver will forward the request to the Authentication and Security check. The authentication and security block will access the Client policy registration database. Client policy registration is a list with all the client information and their status. The client policy registration will be discussed in detail in the later section. Hence this section is limited to the Authentication and Security discussion.

The below picture provides an overview about the Authentication and Security module in Storage Management System. The security and authentication system mainly have the following four components build in.

- Authentication manager
- User name and password Decryptor
- User registration manger
- NoSQL Database entry

The administration console registers the users and administrators in the database. The details of the user names and passwords are stored in a secured access table in the NoSQL database. This particular entry in the database is secured and the access is given only to who has the administrator password. It should be noted we use the same NoSQL Data Base for storing the user name and passwords.

In the below over view picture whenever a client requests for the access to the storage management system the request is forwarded to the Authentication manager which is present inside the Authentication and security block. The authentication manager will check who the sender of this request is and forward the request to the Username and Password decryptor.

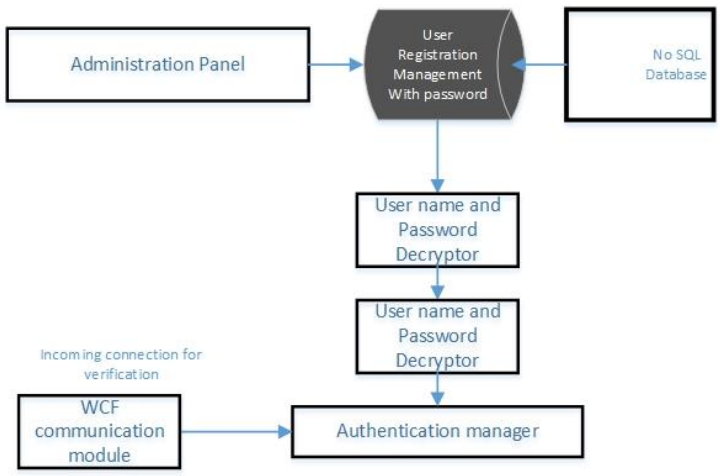


Figure: Over view of the Authentication and Security module in GSMS

The decryptor module includes a suitable method of decryption of the user name and password. The user name that is decrypted is compared to the existing username and password in the NoSQL database. The NoSQL database will have a separate entry for storing the user name and password an access sis provided to only few module like Administrator module and User registration and Management module. The existing username and password that is already present in the system is retrieved by decryptor through User Registration and Management module. The verification logic will compare the user name and password with the decrypted user name and password. Once the user name and password are verified, the requested operation is forwarded to the Event handler and Scheduler.

The following picture shows the work flow of the Authentication and security block.

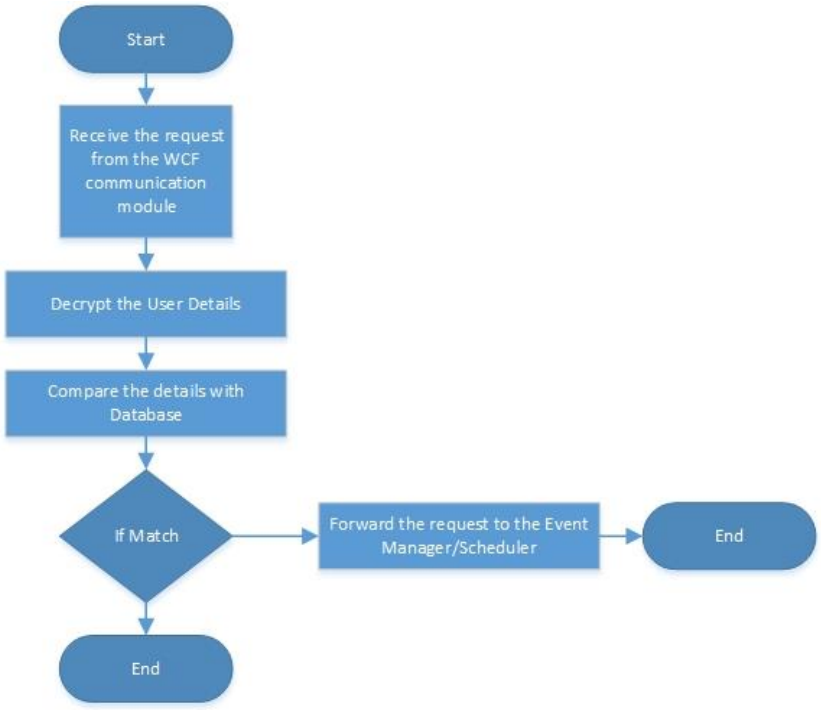


Figure: Activity diagram for Authorization and security module in GSMS

Critical Issues:

Security is always problem for the present internet connections, since we are sending huge amount of important data on internet there may be immediate risk of hacking and other related security breach.

Solution:

The possible solution for this is to make sure that each server and the client maintains the table of ip addresses. Using these ip addresses it is easy to fight security concerns of the internet hacking. A new instance of the data base will be created in the GSMS to store the SCF systems information.

6.4.4 Event Handler and Scheduler with priority Handler

Overview

The event handler and scheduler are the heart of the GSMS. In this module the requests received from the clients are scheduled or handled. It also decide if the request is to be processed with threads, processes or events. The following are the main properties the event handler will extract and finalize.

- Life time of the event
- Thread or process or event
- Priority of the events

In organizing principles it is already discussed which model either thread based or process based or event handling is best for this application. From that discussion it is clear that we should go for a combination of thread based, process based and event based. This package also supports the Task based scheduling or executing the requests from client.

6.4.5 Event Handler and logger

Event handler is the core of the storage management subsystem hence in detail it is dealt in the specific server section in this document. The event handler is present in the forefront of the WCF communication module.

The below is the sample code of the event logger for windows syste.

```
private void btnEventLogger_Click(object sender, EventArgs e)
{
    try
    {
        string FilePath, Line;
        FilePath = "C:\\MySampleFile.txt";
        StreamReader m_StreamReader;
        m_StreamReader = File.OpenText(FilePath);
        while ((Line = m_StreamReader.ReadLine()) != null)
        {
            MessageBox.Show(Line);
        }
        m_StreamReader.Close();
    }
    catch (Exception ex)
    {
        EventLog m_EventLog = new EventLog("");
        m_EventLog.Source = "MySampleEventLog";
        m_EventLog.WriteEntry("Reading text file failed " + ex.Message,
            EventLogEntryType.FailureAudit);

        MessageBox.Show(
           "Successfully wrote the following to the System Event Log : \n"
                      + ex.Message);
    }
}
```

The event handler is the one which schedules and handles the events that are generated by the database or the schedules the requests that are requested by the users or clients.

6.4.6 Event Notification Manager

Notification manager is the explained in the earlier part of this document and hence can be referred from there. The event notification manager provides different types of the events notification, action notification, Email notification, local and remote notification. All these are discussed in the earlier part of this document.

6.4.7 Message Decoder

Message decoder lies right after the WCF communication module. The message decoder has access to the message structure and the message decoder compares the incoming request and maps it to the correct message format. Before doing this the message decoder will also have any possible cryptographic techniques if they are required.

6.4.8 Business Logic

Business logic is specific for each and every server, the repository has a different business logic and the test harness has different logic. Hence it is better explained in those sections as it is not common for all the servers.

6.4.9 Data storage manager

The data storage manager is mainly used to convert one form of message into another forma of message and also the data storage manager converts the message from the XML to DBElement and vice versa.

The following shows the generic DB Element structure which can be used for implementing test harness server and build server.

```
public class DBElement<Key, Data>
    {
        public string PackageID { get; set; }
        public string Category { get; set; }
        public string Description { get; set; }
        public string Query { get; set; }
        public string Name { get; set; }
        public string Author { get; set; }
        public string Country { get; set; }
        public DateTime Time { get; set; }
        public DateTime Time2 { get; set; }
        public string Payload { get; set; }
        public Child[] children { get; set; }
        public int count { get; set; }
        public string searchMeta { get; set; }
        public DateTime searchTime { get; set; }

    }
```

As part of standardization we have only one DB element in all the servers which means the servers adopt the same DB element structure for Repository, Test Harness and build server.

6.4.10 Storage Management Service

The following are the storage management services that are provided by the GSMS. The GSMS provides storage of files and their indexes, storage of user details, storage of the immutable package contents, storage of the reports etc.

Storage of files service

The GSMS provides the storage of file service by storing the files in the hard disk in unique location and when a version is created, then the location is changed to a location where the data will be archived but still accessible.

Technically the files are stored in the hard disk but the location of the file is stored in the key value database.

Storage of package relationship database

The GSMS system creates a different database instance for storing the user details if a user trying to access without registering with the database encounter then a security notification is generated.

We extend the category DB which we created in the project #2 for this purpose. Every entry in the project will have dependency on any other packages, hence the maintenance of the

6.4.11 No SQL Database

Executive Summary

No SQL data base is the integral part of the Storage management system. It is highly impossible to have a storage management system without a data base. Though there are many databases available in the market which are based on Querying language, there are drawbacks such as low speed high complexity to store heterogeneous datatypes in a single database, not flexible for scaling. After analyzing all these drawbacks it is decided to go with NoSQL database. There are also many NoSQL databases available in the market but however no database is flexible to support small data and very big data. The NoSQL database which is built in the project#2 is very flexible and easy to use for complex systems such as NoSQL database and also very flexible to use for small and simple databases such as user data storage etc.

The advantage with the NoSQL database is the scaling without using much memory. For Example if the user want to use the database for storing simple key value, it can be done easily with NoSQL database. If the user want to store very large files in NoSQL database, this is also possible by using NoSQL database. The data base uses the memory which is actually required and unlike other data bases there won't be any wastage of the memory.

In this project to solve the industry problems of accessing the large databases we will work on developing one of the NoSQL database. In the market there is long list of the databases, however they are not suitable for all applications, Hence we are developing a new database with which we can come over all the problems and can be used as single for any type of applications. For example mongoDB is used for large document storage whereas Cassandra is used for key-value database. Hence the existing NoSQL databases have their own merits and demerits. But we will try overcoming all these drawbacks and develop one single database using C# language and .Net framework. While doing so we will also try to retain

some of the good features of the traditions SQL database. The following are the list of NoSQL databases that are available in the market with their drawbacks.

1. Key / Value Based
   e.g. Redis, MemcacheDB, etc
2. Column Based
   e.g. Cassandra, HBase, etc
3. Document Based
   e.g. MongoDB, Couchbase, etc
4. Graph Based
   e.g. OrientDB, Neo4J, etc

As we talked earlier our goal in this project is to develop a database which can perform better in all the above scenarios and better than the above databases.

We will also use C# and .Net frame work in this project to develop the database. Visual studio 2015 will be our IDE to write C# programs with .Net framework.

We will also use the famous 3Tier architecture to implement the requirements. The above all implementations will be explained in detail in this document.

The tool that we are going to develop will serve the following purposes

- Read XML data file from the data storage manager with instructions from user and save it in key value database
- The NoSQL database will read the values from the database when a key is provided as input. This will be a Key Value database where the value can be either a string or a file or a metadata.
- Should be able to store the big files with parent child relationships maintained with the help of metadata stored in the value.
- Should be able to maintain the version history so that this database can be used for Repository without losing the earlier data.
- Provide immutable feature where an entry is moved from normal database to immutable database when check-in happens.
- Remove the key values from the database un on request from user
- Modify the key values from the database up on request from the user
- Create the child relationships between the key values when requested from user in metadata

The main users of the tool will be the developers, programmers, architects, Quality analysts, Quality Auditors, Project Management Office (PMO), Testing Team, Client testing team and other stake holders of the software project etc.

The database will also be accessed by the other applications in SCF such as Build Server, online and offline from different locations. Hence the database should be able to communicate with other applications using APIs error free.

We will also use C# and .Net frame work in this project to develop the database. Visual studio 2015 will be our IDE to write C# programs with .Net framework.

Obligations

The main objective of key-value database project is import XML files and save them in the database using key-value relationship and export the data that is present in the database to the XML when requested by the user. The project also should be able to scale the database size based on the requirement. The explanation of obligations of this project is provided clearly in the below points.

2. To provide Generic feasible database for all the SCF servers such as Test-Harness server, Build Server, Repository, Review DB server, Requirement DB server.
3. Provide multiple instances of the databases which support check-out, check-in, Integrate, Delivery features.
4. Should log all the operations happening in the database
5. Should persist the database with three parameters.
   Every scheduled time
   After predefined number of operation
   Before shutting down
   Whenever user requests
6. Should maintain the file history in metadata of the file, metadata should be generated by the database when the user modify the states of the package.
7. Maintain the relationship of the packages like parent child relationship and dependency relationship.

Organizing Principles

We will modify the already developed NoSQL database in project#2 and project#4. We take the code from the project#4 and divide it as WCF communication modules and Code Database. The division module are mentioned below.

1. WCF modules explained earlier which will connect to Event handler
2. Core Database modules will be discussed in this section

We have already explained the communication module in the previous section and now in this section we divide the code Database in to the following module which will help to organize the functionalities of the Database to suite the requirements of Repository, Test Harness, build server and other SCF modules which we have seen in the beginning of this document.

The following are the divided sub modules of the Core Database.

1. Immutable Database Engine(To store the freeze data which should not be modified)
2. Database Engine(To store the editable data before the delivery is freeze)
3. Persistence Engine(To persist the data both mutable and immutable)
4. User database (To store the registered users data)
5. Category Database(To organize the data required for a project)
6. Sharder (to shard the data of the database is too much loaded)
7. Query formatter(formats a query from the received request XML)
8. Query Editor
9. Display this is used to administration who want to login to the SMS
10. Memory Management (This module manages the memory so that all the database elements are accessible in equal time)

Key architectural designs for NoSQL database for GSMS

Key-Value database tool which is part of repository uses the DB Engine package, DBElement package, DBExtensions package, Display package and Utility extensions package developed by Dr. Facwcett.

The Client interface is implemented using the WPF and the communication channel will be implemented using the WCF. To explain the key architectural design of the NoSQL server and client database it is necessary to mention the life cycle of the Query from the client to the server and the data service from the server to the client. Please find the below

Life cycle of an operation in NOSQL Database:

a. Get the information in the form of a XML/JSON from *Data storage manger*
b. The XML/JSON metadata *for example* says the following data
    1. <operation> check-in/check-out/deliver/freeze</operation>
    2. <package-name>package name</package-name>
    3. <sub-package-name> sub package name <sub-package-name>
    4. <file-name>file name</file-name>
    5. <project-name>project name</project-name>
    6. <file-path>project name/package/sub package name/ file name</file-path>

    The above is the example for repository the requested operation is performed in the database after extracting the information from the Data manager data.

c. database converts the JSON data to XML(if in XML format) and provides the query to the Query engine

d. Query Engine read the input and query the DBEngine
e. DB engine provide the output to the Query Engine
f. Query Engine provide the received output to the Data storage manager interface
g. There can be multiple database instances as required and there also can be multiple clients in the network requesting for the same server which increases the load on the database.
h. Server will also have a performance measurement module which will record the time taken for the task execution and the details of the user executing the instruction and also keeps track of the IP address of the client

DBEngine: DBEngine takes input of key and value pair. The DB engine consists of Insert, Delete, getValues, getValue, getKey etc API. The query generated by the Query formatter is forwarded to the Executive module and the executive module calls DB Engine.

XML to JSON converter: If the client request for JSON data then the database will wrap the data in the JSON format and send back to the Data storage manager. As the GSMS should be generic and support heterogeneous data types we intend to provide all kind of conversions.

Admin Panel: we will develop an admin panel both in the server module and the client module. It will be possible for administrators of the NoSQL database to log in remotely and execute Admin privileged tasks remotely.

XML to Object Converter: This package is used to read the input XML reader and convert it into the DBElement object, this feature is required when the Data storage manager requests for object data format.

Object to XML Converter: To persist the data base from the DB engine element we need to convert it into the XML string and write it to the XML file in to the hard disk.

DBelement: DB element package provides the interface to create a new key value pair which should be passed to the DB Engine. Executive package uses the DB element to create an instance of the key value pair before inserting data into the DBEngine.

DBExtensions: DBExtensions package is dependent on DB Engine and DBElement package. DBExtesnsion package is used to extend the functionality of the DBEngine and DBElement.

UtilityExtensions: UtilityExtensions package is not dedicated to single package and it includes all the APIs that are for all the other packages.

File Handler: File handler package provide an interface to store the incoming file at a location specified by the DB engine. As it is already discussed there can be scenarios where the Data storage manager need to store the file in the database and have the path of the file as Value in the key value database.

Server Display: The display package is an API for DBExtensions, DBEngine, DBElement, DBExtensions and UtilityExtensions package. The Display package is developed in such a way

that it formats the output of DBExtensions, DBEngine, DBelement, DBExtensions and UtilityExtensions.

Database Performance Module: Server will have the performance measurement module which will measure and log the time taken for each instruction execution that is requested by the client. We will use a customized high resolution timer to measure the performance accurately.

Use Cases

In project5 we are planning to use this NoSQL database we have developed in project 2 and project 4 for implementing the repository. Once the project 5 is implemented we will have full pledged software package repository which can be part of the SCF. The repository will be so well organized by grouping different packages and maintaining the parent and child relationships.

All the stakeholders of the project accesses the key-value database tool on daily bases, the following are the project stakeholders that are identified as of now.

1. Software developer
2. Software Architect
3. Testing team
4. Integration team
5. Project Management Office(PMO)
6. Technical team at Client
7. Client-End user
8. Quality Analyst
9. Quality Auditor
10. Any other project stakeholder who uses the tool
11. Other tools that uses API to interact with the Database

Use in Repository

In repository the database can be used to store the files, packages and projects information and to maintain the user and administration details. The special requirement in the repository

is that the status of the files should be maintained with three status check-out, check-in and freeze. The database should move the record of the particular key-value pair to different databases based on the status of the file or package.

The following are the status and the respective database.

1. Check-in element will be in check in database
2. Check-out elements will be in check out database
3. Freeze element will be in freeze database

The freeze database is immutable database which will be discussed in the later part of this section.

Use in Test Harness Server

The NoSQL database can also be used in the test harness server for the following applications.

1. To store the test cases of all the package elements of the repository database
2. To store all the test related information and the test results
3. To store the users registered database and few other which will be discussed in the later sections of this document.

Use in Build Server

The NoSQL database can also be used to implement the build server. The build server compiles and builds the projects that are present in the Repository server. Once the compilation and build is successful the build server will store the build result with warnings, errors and etc. information in the database. The build server also store the user information the database as key value pair.

Use in Client

The client uses the NoSQL Database to maintain the user preferences, to store the logged in user details and also it store the events a user has executed from the client. The client plays a major role in storing the user activities which will be easy for the user to connect to the SCF modules.

Use in the Requirement database Server

The NoSQL database is used to store the requirements that a customer has requested and maintain the relationship between the requirements, packages, projects and delivery.

Every requirement in the requirement database is linked to at least a delivery, package and project. Every delivery will have at least a project and project will have a package and package cannot be implemented without a requirement. All these relationships are maintained in the requirements database.

Software developer

As a Developer: On daily basis the software developer checks in multiple files which he codes. The software developer may also use the key-value database to store the records later he can share it to the colleagues in other locations. Hence our software can be used by the software developers as both the repository and also as software sharing database.

As end User: The software developer also will use this to provide some help files to the end users. The key-value database is used on daily basis to store the programing files, to share the programming files with other developers etc.

The software developer also can develop other applications to interact with the key value database using APIs.

Software architect

The software Architect uses the key-value database to set up the initial project setup before the actual project development started. When the database is used as repository the software architect uses it to access the program files. If the database is used for developing another product or project then the project architect will use the key-value database to evaluate the performance of the product or project.

Testing team

Testing team can use the database as a repository to keep track of the test cases and test results.

Testing team may use this project for testing such as to check if all the requirements are implemented as per the requirements document that is provided during the project initiation.

Quality analyst/Quality Auditor

The quality analyst and the Quality Auditor will use the key-value database to check if the database

End User

The end user may use the database to save different data type of the data such as photos, videos, code files etc. as part of their daily usage.

Database Admin

Database can use the client software to control the database server from remote location. Database admin also uses this

Application Activities

Key-Value data base analyze the metadata of the file to know some important information about the data present in the file. The metadata provides the information about the author of the file, the software package that the file is part of, other relations with other files and

packages. In order to get the metadata of the file, the XML file should be parsed for the required information.

Receiving input from Data storage module

The user will open the client application and loads the query or data that is to be checked into the server. The client parses the XML input or the manual hard coded input into JSON format. After converting the input in to JSON the client puts the data on the network by providing the server IP address.

Receiving the input from the Client (Server)

The server parses the query and prepares the output and converts it into the JSON format and puts it on the server as a packet. The server also wraps the data in JSON format using the IP address and puts on the network.

Display system of the server (server)

The server will also have the feature where the log file will have all the operations that are happening in the server. The server also keeps track of the user information who is executing the operations in the server. Server also displays on the console the result of the operation and will record in the performance module to measure the performance.

Parsing the input file

The user provides the input file while starting the application via client. The user also provides in the metadata of the file if he needs to modify the existing file in the repository or he wants to append to the existing file in the database or he want to enter the new file in to the database. The server obtains all the above data from the clients in the network.

The above information is extracted using the file parser by the Executive package.

Generate the Key-Value pair

After parsing the file the key-value pair is generated along with other important child keys. The key value pair is used to identify if the key is already existing in the database. If the key is not available in the database then a new key is created and the value is saved in the database. If the key is available based on the information provided in the metadata the data is modified or appended.

Extracting the Key information

If the user wants to delete the record then he will provide only the key information in the metadata and mark for delete. The executive package will extract the key by parsing the file and use the key to delete the existing records. Incase if the user does not provide any information about the operation to be performed then the database considers it as new key-value pair and saves in the database.

Persistence of Data and Queue

The database should also make sure that the data persists when modified and not ephemeral. The persistence database will always preserve the earlier versions of the data that is modified

and can be accessible at any time by querying. As we know in code repositories and in social network when data is changed it is very much required to retain the previous versions.

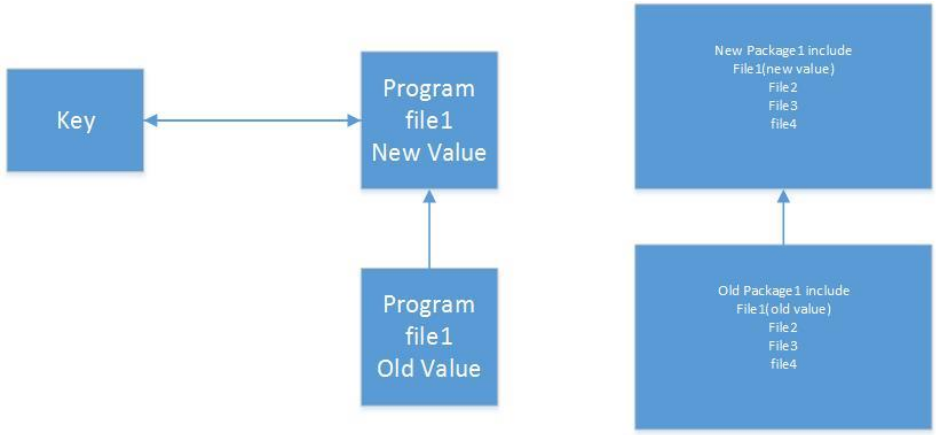
As we are planning to use the database for repository purpose and if possible we can also use the database for social media and other web applications. Hence according to government's rule in many countries it is mandatory according to law to persist the data along with all earlier versions after modification.

For persisting the data the server will pause the operations requested by the clients and persists the data and once the persistence is done the server continues the activities that are requested by the clients. The Queue will help to hold all the requested operations. We will use mostly the priority queue which gives preference for the instructions that take less time.

The server we are implementing gives preference to answer more clients rather than the amount of data. If the data that the client is requesting is huge it is given low priority in the Queue.

Saving Data to Database

The database saves the data once received from user, the key value table is created after saving the data. If there is an update request then the data is updated at new address by persisting the previous versions of the data. Now the new data exists at one address and the old data exists at another address. Hence a new relationship between the old data, new data and the primary key to be created along with the package information.



Once the value is update the database also should update the new package with the latest file. During the retrieval of data the database should be able to refer to the old package as well as new package. This is the main purpose of the data persistence and code repository. This feature is mainly useful during implementation of social media projects and code repositories which we are planning to implement in project 5.

Retrieving Data from Database

The database will return the package of program files, a single file and an individual value on query from the user. The modification of data will be accurate and each version of the file will

be linked to respective packages. From the above figure we can understand that the obsolete versions of the file or the key are retained even after updating the key-value pair. This feature makes the database robust and can be used in any critical applications.

Heterogeneous Database

Unlike all other NoSQL databases the key-value database that is being implemented in this project will perform better with all type of data types. This database will support the following types of heterogeneity.

Technical Heterogeneity

This database will support multiple file formats, multiple communication protocols (to access remotely).

Data model Heterogeneity

Semantic heterogeneity

Logging and Debugging

The database will support full activity logging and debugging. For every actions that is performed on the database will be logged in the system.

The log report typically consist of the changed key values and the changed child relationships with the package. For example if a key is changed from one package to another package then this complete activity will be logged with user details. This feature of logging and debugging will help users to keep track of the changes that happen on the database.

Dynamic Schema

The database will support dynamic schema which gives flexibility to the user. Since most of the projects that are run in the real time industry are following agile process of development, it is very much required that the schema of the database should be dynamic.

For example while the customer providing the requirements he is not sure how many users will use the database and how much data inflow or outflow will be required in the future. Hence by implementing the dynamic schema we can make sure that the growing data will not affect the existing data.

The dynamic schema is also best at supporting unstructured data. In real time unstructured data is very common and static schema cannot handle the unstructured data.

Sharding of files

The database in this project deals with big data and the size of the file or value of the key can be as small as 1KB to 10petabytes. Since single system cannot handle such a huge data we implement Sharding of huge files. Even though there are many types of sharding models and algorithms we implement automatic integrated sharding.

In automatic integrated sharding we shard the file whichever is above some predefined size. However the user will be notified before sharding and no additional learning or understanding of sharding is required from user side as the database system can handle in such a way that the speed will remain same.

Scheduling

The database scheduler schedules periodical data backup to make sure that the data is not lost in case of system failure. The scheduler is also used to read the stream of input from the executive package. As we know in big data the data can be as bigger such that the data keeps on coming and waiting in the queue it is the duty of the scheduler to take care and handle multiple requests at the same time.

The scheduler also should make sure that all data queries and data operations are attended in equal time. The scheduler will keep database stable and healthy.

Immutable Database from query results

The database on command saves the results of the query by creating a new immutable database. The DBEngine creates immutable database on request from user. The immutable database holds the child parent relationship of key-value pairs.

Zero configuration DB Engine

The database will be zero configuration database which means there is no configuration or installation required by the user. The database will be folder with the executable file.

Error reporting

The database system will report error if the input XML file is not according to the format specified by the database manual. The error reporting scenarios include if the XML file missing some mandatory information or if the file is not according to the format specified by the database such improper XML metadata tags.

Display Results

The final step after doing the operation that is required by the user is to display the result. When user request for a new key-value entry the result displayed on the console will be the state of the database before the operation and the state of the database after the operation. The display function also will display the records that are modified and the state of the database to the console after user operation is performed successfully. The display function also provides the error information such as no key found in the file that is input, no author information found, no child key information is found, the file is not part of any package etc. the error information will be discussed in deep in the later sections.

Packages or Modules of Server

The design of this project will result in the following packages or modules. Each module or package is explained in detail.

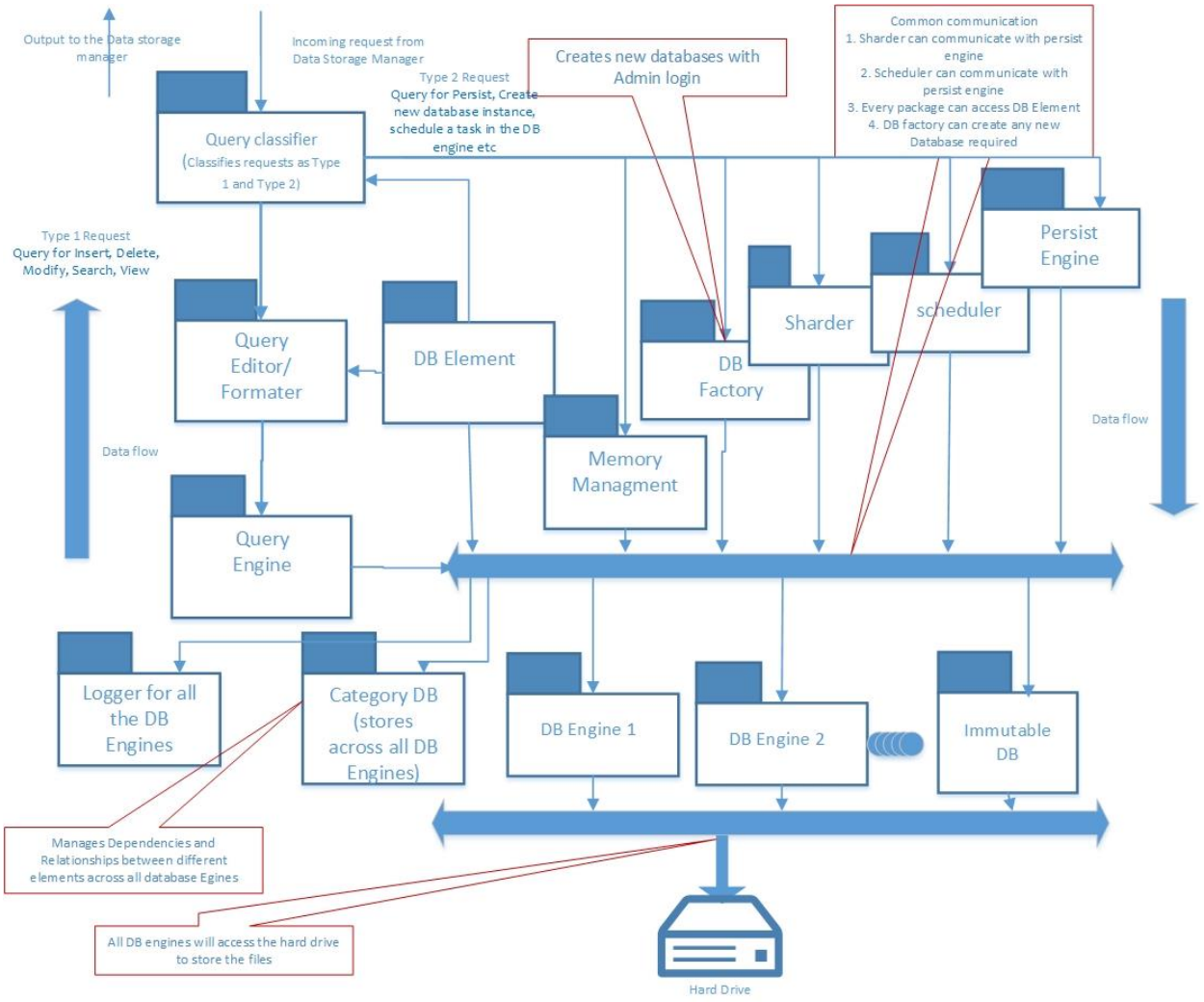


Figure. Package diagram of NoSQL DB

Explanation about Package Diagram:

Query Classifier (Server)

This module acts like a controller module, this module controls and calls all other modules in this project. This module receives two kinds of requests from the Data storage manager.

1. Query for Insert, Delete, Modify, Search, View
2. Query for Persist, Create new database instance, schedule a task in the DB engine etc.

The module will analyze the query received and classifies it either as Type 1 or Type 2, then the request is forwarded to the proper module to communicate with the Database.

The interfaces for this module will be as follows.

1. DB Element
2. Query Editor
3. Memory Management
4. DB factory
5. Sharder
6. Scheduler
7. Persist Engine

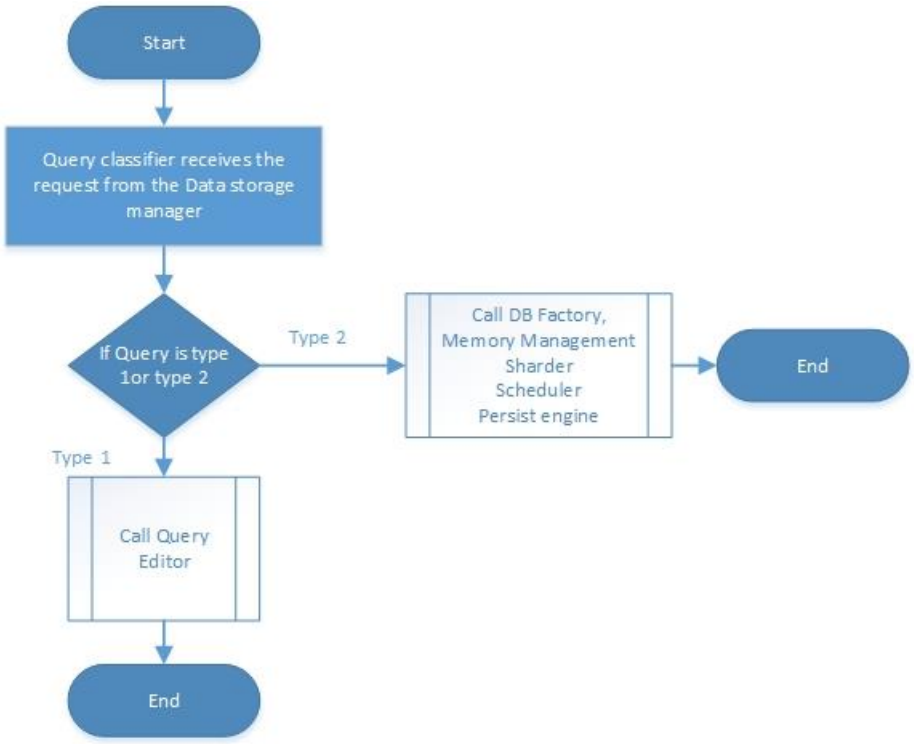


Fig. Activity Diagram for Query Classifier

The module receives the file and sends the instructions to Query Editor which analyses the input file from the user and sends the valid data to the Query formatter. The Query formatter formats a query with the input data. The Query formatter also generates the key value pair and send back to the executive package. The executive package receives the response from the Query formatter and checks if the query is to read the data or write the data. If the query formatted is read then the query engine is invoked or if the query formatted is write then the DBEngine is invoked.

After receiving the inputs from the Query engine or DBEngine the Query classifier module calls the Data storage manager module. The display module displays the result to either to console or XML file or GUI depending on the user request.

Query Formatter (Server)

The Query formatter plays important role of making a query by taking inputs from the executer. The input is the metadata provided by the user in the file. By using the meta data the query formatter forms a query which can be understand by the Query engine and passes it to the Executive package. Query formatter is also responsible for generating unique key value pair.

The responsibility of the Query editor is to take inputs from the executive package and read the key value index and provide them as input to the Query formatter.

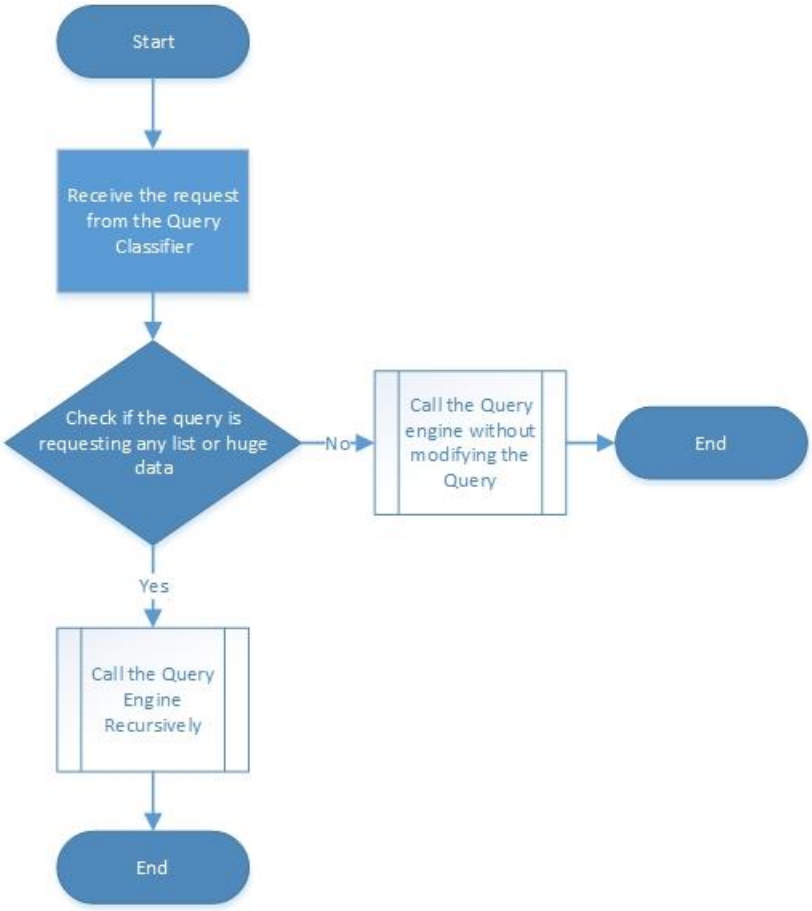


Fig. Query Editor/formatter Activity diagram

Executive package will request query editor in case the input is not understandable by Query formatter. The query editor converts the raw form of the metadata and handovers to executive package later the query formatter will be called.

Logger

The logger is very important package, the responsibility of the logger is to keep track of all the changes that are done on the database for every operation.

The logger is controlled by executive package and it will save each change in the data base with the time stamp.

The logger is responsible to log all the operations that are happened in the database. The database logger will also log the user IP address and other authentication details. For improving the performance the server also keeps track of the average query handler time for each client. This will be very useful to decide the priority of the client operation during multiple requests.

Let's say there are two queries from client X and client Y, client X has the average execution time of 20milli seconds and client Y has the average execution time of 20seconds, before serving the client requests the server checks the average performance time and prefers the client X as the queue will be reduced.

Category DB(Key Value package relationship index)

The key value index package is called by the executive whenever there is change in the key value relationship. The key value index will also have the details of the packages that the key is assigned to.

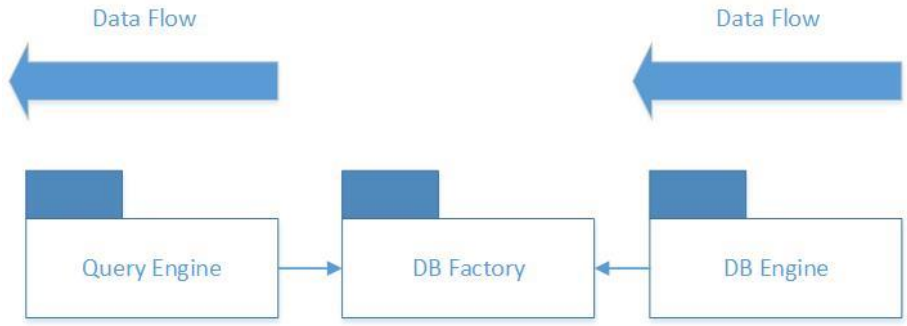
Sometimes if the Query editor requires any information about the relationships then key value package interface will provide with the required information.

DB Factory

DB factory is creating multiple Database instances if required and is also responsible for storing the data and maintaining in the memory. DB factory communicates mainly with two packages one is DB Engine and another one is Query classifier. The Query classifier queries the data elements that it want to read and provide the result to the user via Data storage manger and console which can be understand from the package diagram at the starting of this section. Query engine mainly collects the data from the DB Factory and provides it to user. Whereas the DB Engine collects the data from the executive package and provides it to the DB Factory which keep the data and persists the data later when the scheduler runs.

The database factory will create a new database collection if required based on the user command.

DB Engine writes the data on command from the executive package whereas the Query Engine reads the data from the DBFactory on command from the Executive package.



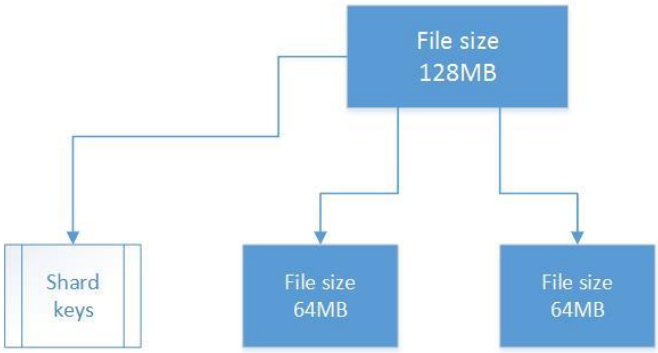
Memory Management

Memory management is the responsibility of this package. As there will be huge amount of data that is writing into the database on daily bases, the memory management system will make sure that the closely related data will be kept at nearby addresses in the memory. In future the memory management package will also defragment the data. Defragmentation of the data will increase the speed of the data access.

Data Sharder

Data Sharder's responsibility it to identify if the collection size is above certain size and split the file into small files and persist it. The sharder is also responsible for keeping track of the addresses of the data storage relevant to a collection.

Afeter sharding, the sharder generate the shard keys and share the shard keys across all the sharded servers.



Scheduler

The scheduler receives the input time interval or number of writes from the executive package. Using this trigger the scheduler will persist the changes.

The scheduler is capable of scheduling timed or event triggered backups of the database, timed or event triggered persistence of database. The scheduler also will have flexibility of grouping similar jobs and running them together. The scheduler package provides the log information to the logger to keep in records. The log information contains the list of tasks that are run and the time stamp along with the status of the jobs. The scheduler package can also be configured by the user.

Monitoring jobs: The scheduler is also responsible for monitoring the jobs until the jobs are completed.

Group by Database

In project #2 we implemented a new feature where we group the keys based on the package they are related to. For example if we have three projects which are using the key value database, then while entering the keys in the database the keys will be grouped based on the project they are related to.

Display

Display package mainly receives input from the executive package. Based on the requirement the executive package may request for console display or XML file output, or GUI display.

The server will display the performance reports of each client operation that is served. The server will also displays the part of the operation that it is presently running which can be called the status which will be helpful in the debugging.

The Basic GUI display package receives input from the display package. The basic GUI display package uses the .Net frame work to provide proper GUI display.

The console display package receives input from the display package, based on the input from the display package the Console Display package displays the value content of a particular key. The console package also displays the other information requested by the display. The project also contains the XML generator which will generate the XML file when requested by the user. This XML generator is embedded in the XML display package.

Critical Issues

Handling Queue Overflow (Server and Client)

When the load from the clients if more the client will maintain the queue which keeps track of the instructions and provides to the network or database as FIFO. If the queue fills up then the queue overflow may happen which will block the communication.

Solution:

1. The solution we have here is either to use priority queue where the server executes the instructions or client requests which take very less time so that the no of elements in the queue will never get overflow.
2. The other solution would be to use vector where the queue size can be incremented whenever the queue is full.

User authentication and securing the communication channel

As we are planning to use out database server over internet it should be always secured. But event with advanced cryptographic techniques no system is secured.

Solution: In our project we are using random key generation algorithm which is proved to be secured and being also used for banking databases.

Pluggable sharding strategy

The pluggable sharding strategy would be critical. As of now most of the databases are implementing fixed or count based sharding strategy. In key value database since we are using the database for real time big data applications we require to implement pluggable data sharding strategy which is complex in nature and chances of losing the data.

Solution: In key value database the following pluggable strategies will be followed. The user can configure and run the hash based sharding. While the user providing the input file to the key value database a new XML tag will be provided to the user where the user can mention that for a particular file the sharding should happen with predefined condition.

We can also design the server in such a way that the user cannot leave the sharding option blank when providing the configuration to the server.

Multi Thread

As the database system is accessed from different locations and by multiple users at a single time, serving all the users concurrently is challenging task. To solve this issue we will implement a multi thread service system where all the requests are handled with equal priority and are served on constant time.

Solution: A new task scheduler will be implemented to solve the issue with the multi thread operations. When the database server serves multiple priority requests.

Distributed Architecture

Since the database will be handling huge files which are of peta byte data sometimes, we need to have multiple servers for storing these data. Implementing distributed architecture where the data is saved on different servers and maintained by multiple database systems can solve this problem but at the same time it is challenging and critical in nature.

Performance for Big Data files

If the file input is of big size, the time taken by the database to process it and save it will increase. And also if the output file requested by the user is of big data size then also the time taken by the database to process will increase. How the tool ensures the time taken by the database for big files and small files is same.

Solution: The database uses the technique data sharding. The data sharding will split the files which are huge and saves them in different data collections using multi thread. To read the big data files when queried by the user, using multi thread technique each file is processed independently and then finally put together and provided as output to the user. Hence using the multi thread we will make sure that the time taken to read and write the big data files is same.

Without GUI and Console how does user provide input

It is mentioned in the requirements that the tool will not have any GUI and Console interface. Without any interface it will be difficult for the users to interact with the database.

Solution: The XML file that is read by the database will be designed with the tags which says what operation the user want to perform. Using the tag <operation> the user can tell the database if he want to read the record or delete the record etc. Incase if no tag is present the data base treats it as new record and generate new key-value pair and save it in the database.

Maintaining relationships with old packages and new packages

If the child parent relationships are to be maintained in the database it can become complex to relate the child parent relationship after few routines.

Solution: to make sure that the child parent relationships are maintained even if the database is growing, the key-value database provides a feature which will allow the user to create a separate index database up on request.

Data loss

Since the database is a complex software system there can be failures either due to the hardware or software failure.

Solution: to avoid data failure we implement the data replication feature. Data replication is the process of creating multiple copies of data at different server since the database support the distributed architecture the data replication can be implemented.

Extended Applications

Extended application as Code Repository

The developed NoSQL database can be used for code repository by software companies to store the code and access from multiple locations. Since the key value database has feature of maintaining the child parent index the package structure of the code can be easily retained and accessed without much hassle.

All of the above features makes the key value database a suitable database for implementing code repository.

Extended application in social media

Since the key value database will be very fast and responsive, this database can be used for the real time analysis of data (data analytics).

Extended applications in Internet of Things

The key value database is also well suited for storing the internet of things data as the database is heterogeneous. The sensors provide different kinds of data logging, hence the Internet of things applications require Heterogeneous database.

7. Software Management Subsystem (Repository)

7.1 Executive summary:

Though there are many servers in the SCF, repository is special as it stores the code files of the software. The repository also maintains the versioning of the files and keeps track of the changes.

Often the projects run for decades with thousands of new requirements and enhancements, Hence, the repository system should be flexible to upgrade whenever new hardware comes available or whenever new requirements come. Since there will be hundreds sometimes thousands of teams accessing at the same time, hence there can be multiple repositories at different location where users can access the code without much latency. However this flexibility comes with a problem. These problems will be discussed in the critical issues section.

The synchronization of the multiple repository servers in the SCF is very important as it can lead to big problems where the data may be deleted or over written.

Note that the project is group pf packages and package is group of source code files or group of sub packages. While sub packages is also group of source code files.

7.2 Organizing principles

The repository mainly consist of the package manger, Version manager which are core for the repository. The executing part is divided in to Task/Event handler. And the communication part is divided in to WCF client communication and WCF SCF server communication. We also introduce a notification manager which will provide notifications to the users based on the business logic or the instructions set by the user in the account settings. The following shows the four layer of the repository system.

1. Communication layer
   a. WCF client communication(User to machine communication)
   b. WCF SCF communication module (Machine to Machine communication)
2. Security Layer
   a. Authentication and Security check and message decoder
   b. User database
3. Execution and Business layer
   a. Priority handler
   b. Notification Manager
   c. Package Manager
   d. Version Manager
   e. Repository Task Handler and Scheduler
   f. Request forwarder
4. Backend Layer
   a. NoSQL Database

b. Event Logger

In GSMS we have seen all the modules except Version manger and Package Manager. And in the further subsections we will see the critical issues with respect to the repository.

As the repository's main duty is to maintain the code in the database with versioning and also maintain the relationship. We define the following versioning policies.

7.2.1 Versioning Policy:

Meaningful versioning should be followed to avoid the confusion in the later stage hence in this versioning system there are four numbers as below

A generic version number: X.Y.Z; R

'X' represents the major project release
'Y' represents the major feature change in the project
'Z' represents the bug fix of the package or file
'R' represents the revision or modification of the file

If a user want to create a new project for a new customer then the number 'X' will be incremented. If a user want to create a major release of the project then the 'Y' will change. If a user want to create a bug fix version then the 'Z' will change and if a user want to create a new revision with some small changes then the 'R' will change.

This system of versioning is followed across industry and hence this will be easy for the customers to use.

7.2.2 Compare the code

The repository also will have the code comparer which helps the developers to compare the code in the server itself. Comparing the code and editing giving build in the server itself provides great flexibility for the users to use the code repository and also save lot of resources when used in the server rather than in the local system as the chances are less for losing the data and it is secured online in the server. The code repository also provides functionality to use any tool to compare the code.

7.3 Uses and Users

7.3.1 Software Developers

Software developers use the repository to check in and checkout the software packages they develop. They also use to view the project status and the number of packages available in the project and who is responsible for which package. The Software developers also use the repository to deliver the software after freezing the software.

The software developers also can give the build using the repository and the repository will contact the build server present in the SCF and give build for that particular package.

Using the repository's dynamic build and integration whenever the user check-in code the repository checks the static build errors and notifies the user indicating the problem. The repository also provide the functionality to execute check in code in the test harness server. The repository in the backend connects to every other server present in the network which makes it simpler for the user without contacting the individual servers.

7.3.2 Project Management Office (Active Stake holders of the project)

The stake holders of the project management office uses the repository to check the status of the project from time to time. The repository also helps the stake holders to know the ground reality of the project which means the actual code that is working, tested and build through. The customers also can have limited access to the repository in projects where software sharing is required.

7.3.3 Test Harness server

The test harness server sends the request to the repository to provide the packages that are required for the test harness execution as requested by the users. As we have already stated earlier the Test harness server has access to the Repository. This can be seen in the starting of this document.

7.3.4 Build Server

The build server is the subsystem in the SCF, the build server accesses the repository to get the package details with code files and give build in the build server when the user requests. The communication link between the build server is established if anyone of them requires to build software.

7.4 Package diagram of Repository

The Repository has all the components of the GSMS which we discussed in the earlier section. The only difference is that the business logic and the operations on the NoSQL database vary. As the repositories main responsibility is to maintain the version of the files and the packages with versioning we emphasize more on the package manager and the version manager. We define the policy to provide the versioning more efficiently.

Policy of Versioning: This is already discussed in the previous subsection and hence it will be skipped.

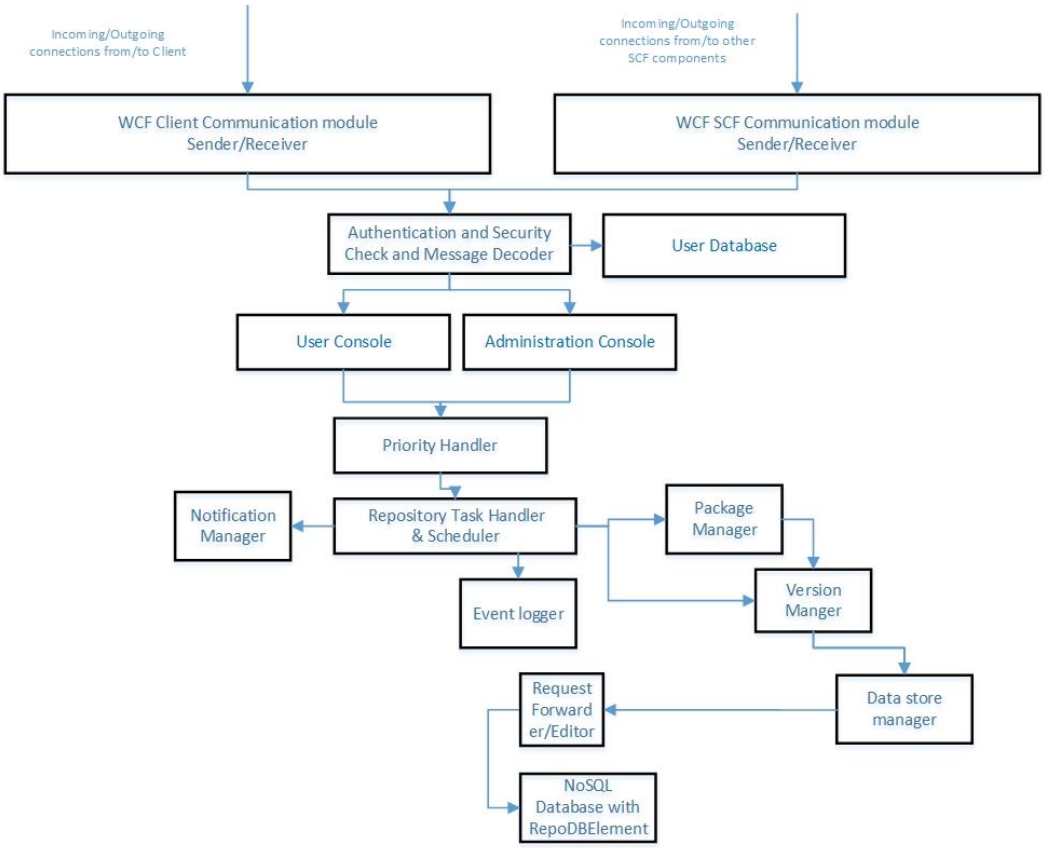


Figure: Package diagram of Repository

The request is received from the WCF communication service module and it is forwarded to the authentication and security module. The authentication and security block checks the user database and authenticates the user trying to connect to the repository.

After successful authentication of the user the authentication module forwards the request is given with either Administration privileges or normal user privileges. The request is forwarded from the user/administrative module to the priority handler here, based on the user profile from which the request is coming the priority is set which means the request from the normal user will have less priority and the request from the administrative user will have the highest priority.

Once the priority is set based on the priority the request is forwarded to the Repository task handler and scheduler. The repository task handler will check the request if the request is accessing the database and editing the files the request will be forwarded to the NoSQL database. The NoSQL database will reply back with appropriate action. If the request is regarding the package information the request is forwarded to the package manger which will contact the NoSQL database and provides the appropriate information back as reply. If

the request is regarding the scheduling of the task or event then the scheduling is done by creating a new task or thread or event based on the request.

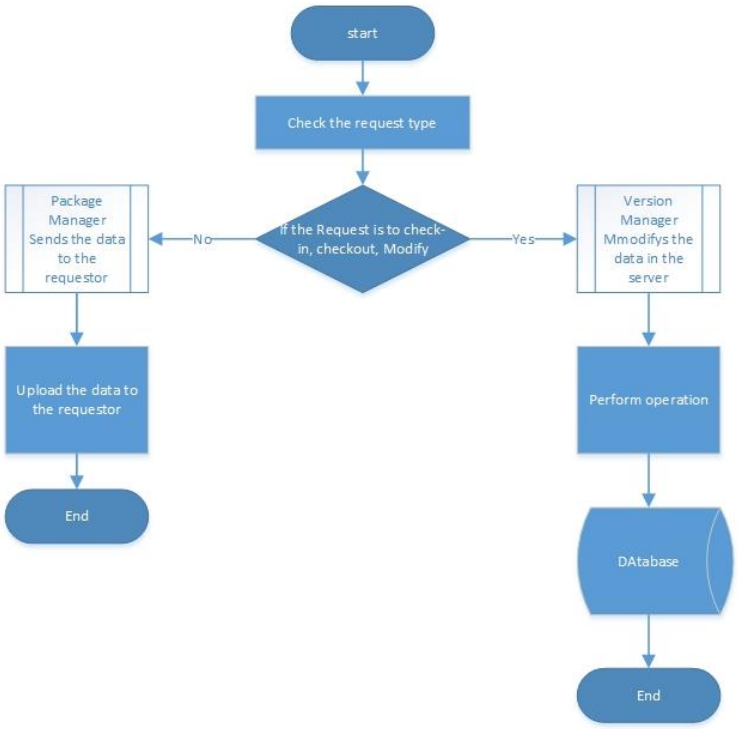


Figure: Activity diagram of the Repository

As there are only two kinds of operation in the repository one is to modify the data and another one is retrieve the data. The business logic checks if the request is asking for the data or to edit the data based on the request the operation is selected in the repository.

Once the task execution is done the notification message is prepared by the notification manager.

Interfaces and Integration

The following table explains the interfaces that are required for implementing the repository system

Interface Name	Reason	With
Repository task handler	To collect the request from the priority handler	Priority handler(GSMS)
Repository task handler	Notification management	Notification manager (GSMS)
Repository task handler	To log events	Event logger (GSMS)
Package manger	To maintain version in the database	Version manager (GSMS)

Using the above interfaces the repository system can be easily integrated into any other system which uses the GSMS and it is easy to be part of SCF.

7.4.1 Package Manager

The package manager is the heart of the Repository which maintains the relationships of all the package like which package is present in which database and what is the latest status of the package. The package Manager have a complete list of the packages that are available in the repository and it maintains which version of the package belongs to which database. As it is already claimed that a package manger will have NoSQL database which maintains the relationships of all the packages this relationships are used to get all the packages of the repository.

Uses: The package manger is used in the repository to store the relationships of the packages and the child packages. The packages are also related to the created user, owner and project.

7.4.2 Version Manager

Version manager is the core of the Repository, it keeps the old files named with a version and allows us to edit. The main purpose of a repository is to store a set of files, as well as the history of changes made to those files. Exactly how each revision control system handles storing those changes, however, differs greatly as discussed in the Versioning policy in the previous sections. These differences in methodology have generally led to diverse uses of revision control by different groups, depending on their needs. An example of how the versioning us performed is shown in the below figure in this section.

The version manager also includes the following information into the File's metadata.

1. It stores the historic changes of the files.
2. It stores the set of commit objects.
3. Stores the set of dependencies to commit objects, called heads

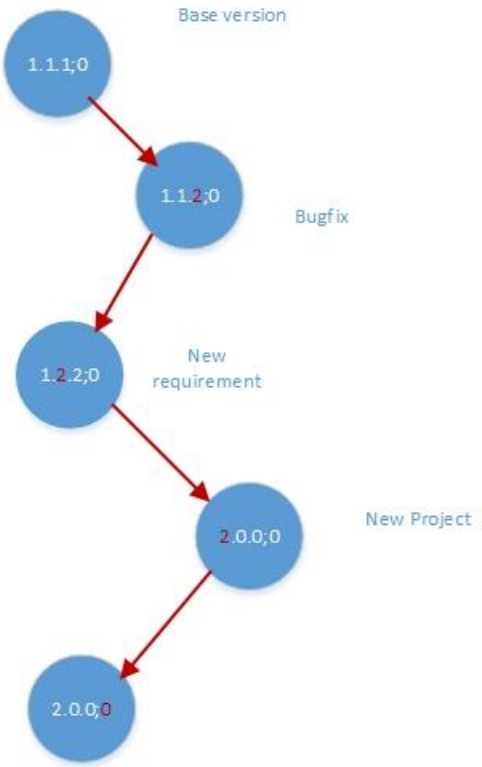


Figure: A bird eye view of version policy

From the above picture it is very clear how the versioning is done using the version Manager. Whenever there is a change in the version of the file the location of file is shifted to the Archival database.

Baseline version management:

The version manager should keep track and update the base version of the software whenever there is a commit in the database. The user will be able to checkout from any historical versions and there is no constraint that the user can only edit the latest version.

7.5 Responsibilities

Version control

Relationship database

Creation of immutable database

7.6 Activities

User sends the request for operation

The user using the client sends the request to check-in/checkout/delete/modify/create new version. While sending the request the user sends the version he want to create. If the user does not provide any version then the version manager assumes that the user is creating the revision and increments the last number after semi colon in the version.

User selects the file to check-in

While sending the request the user sends the file in the message to check-in /modify (if not checked in)/. The repository reads and creates appropriate version and sends the acknowledgement to the user after the operation is done.

User creates a new project or new package

User who can be either admin or normal user requests for the creation of the new project with base line versions mentioned. The base line versions of the previous packages are taken default if the user does not specify which versions of the packages are to be used to create a new package or project. Note that the project is group of packages and a package is group of files.

User gives build on a package

The user can request a build on a package or a project and the repository will contact the build server and execute the build instructions. The repository will have access to build server in the back end and the user need not provide any details apart from normal check in or just right click on the package and give build.

User Review the packages

The user can right click on the package view in the repository and view the reviews held for that package and if the user is selected as reviewer he also can edit the review and provide the review points. Any user can be reviewer and the owner of the package will have option to select the reviewer.

7.7 Critical Issues

Synchronization among repositories

As there can be any repositories depending up on the number of teams and their locations, the synchronization of the data among these repositories is very much important. The repository is very crucial because there are chances the data will get overwritten by multiple users and the original data is lost. There is also possibility if the proper synchronization is not done the data can be lost.

Solution:

This problem can be tackled by providing the centralized package structure, however this is not shown in the architecture of the present proposed SCF. The centralized package database will maintain the status of the each and every file and the package. Which means when a user

does changes the repository updates the status of that package or file to busy and no one will be able to edit that particular file. That means the object will be locked and no repository can provide the access to package that is already being accessed.

Critical analysis

Saying that this will introduce the additional data exchange between the servers hence delaying the operations. To avoid this problem we use the binary representation of the status of the package either 0 or 1 if the status of the object is 1, the object is locked and if the status of the object is 0 the object is not locked.

Security

The security is always a concern for the modern day internet and hence the same threat is there for the repository that we are proposing. As we know that the source code will be company's intellectual property and this is the hard work of hundreds or ever thousands of employees, securing this source code is high priority while transmitting on external internet, while there are company leased lines with dedicated internet connection where no one will have access this will limit the performance.

Solution:

To provide the security any lightweight encryption and compression techniques can be used and this not only reduces the code size but also provides security.

Scalability

I modern day software systems scalability is always a major issue, as the number of software developers increase in a company or when a company wants to start a new venture in different location, often software companies face the challenge of the scalability.

Solution:

As it is already discussed earlier the repository system is designed to exponentially increase the number of repositories in the network without any limit hence there will not be any issues with the scalability of the servers.

8. Test Harness Server

8.1 Executive summary

Test harness server is the automated testing framework for any software. Test harness servers will be specific for each language as the test harness server need to understand the syntax of the language in which the source is developed. Though there are many test harness servers are available none of them suite the requirement we have in the proposed SCF.

The test harness server will provide the following features which are basic in nature.

1. Automate the testing process
2. Execute test suites of test cases
3. Generate associated test reports

While providing the above features and functionalities the test harness server can also provide he below benefits.

1. Reduce the investment in hiring the testing human resources
2. Increase the productivity
3. Decrease the errors in the testing process making it efficient
4. Repeatability without any human interpretation
5. Increase quality of software components
6. Offline testing
7. Simulating the difficult use cases and the test conditions

8.2 Organizing principles

The test harness will use the GSMS system we proposed in the previous section of this document. The generic Storage Management System will provide all the required modules implementation. The Test harness server will have the following components in the architecture and most of them are covered in the GSMS section in this document and hence we will discuss the business logic part in this section. The business logic is the where the testing is actually done. The test harness server will also have the NoSQL database to store all the test reports that are generate for future analysis purpose.

The test harness server is the most frequently used component in the SCF and hence the load will be more when compared with the other components of the SCF network. The system also maintains the database which will store the test cases that are required for a certain project. The test harness server database will hold similar packages that are used in the repository. The only difference is that the test harness server holds the test cases written in different formats and the repository holds the source code of the software.

The another difference between the test harness server and the repository is that the repository does not execute any code it merely stores the code and provides whenever required, Whereas the test harness server holds the package information, test cases of each

package and software component which executes the written test cases when require. The heart of the system is the test harness processing unit.

The test harness processing unit should also support the multiple programming language support which is lacking with the present market test harness server. As we mentioned earlier, this comes in the scalability and flexibility to extend for other

As already we have seen for the repository the rest harness server will also have four layers of software components as mentioned below.

Communication layer

1. WCF client communication(User to machine communication)
2. WCF SCF communication module (Machine to Machine communication)

Security Layer

1. Authentication and Security check and message decoder
2. User database

Execution and Business layer

1. Priority handler
2. Notification Manager
3. Package Manager
4. Version Manager
5. Test Harness Task Handler and Scheduler
6. Request forwarder

Backend Layer

1. NoSQL Database(for storing the test cases)
2. Event Logger

8.3 Uses and Users

The test harness server is used by an entire organization at different levels. However the main stake holders of the project are the intensive users for this system hence we will discuss those uses in this section.

The users of the Test harness server are as follows.

1. Software developer
2. Test Engineer
3. Project Management Office(Project Manager, Customer other stake holders who are not involved directly)
4. Administrator

Software Developer

The software developer is the main user of the test harness server, the software developer after writing the source code writes the test cases executes them in the test harness server and stores them in the test harness server which can be accessed by other people in the team.

Test Engineer

The test engineer is also one of the crucial user of the test harness server, the test engineers will conduct all kinds of testing on the software using the test harness server. The typical testing that a test engineer conducts are Unit testing, Integration testing, Acceptance testing and to store the reports in the test server database.

Project Management Office

The project management office includes also customer who are the sponsors for a project may request for access to the test harness server. The project managers also use the test harness to track the delivery information and also to track employees.

Administrator

The administrator can use the database to maintain the database. The Administrators are also responsible for creating new users to access the test harness server. The administrator also keeps track of the loading of the testing harness when the loading is more than the test harness server can handle he may request for new server instance.

The uses of the Test harness Server are as follows.

1. For executing the test cases on a source code(Unit Test, Integration Test and interface testing)
2. For automating the testing on packages
3. For storing the test cases and versioning

For executing the test case

The test harness server is used to execute the test cases on project packages. The test harness will have the test engine whose sole responsibility is to execute the test cases on the source code and generate the test reports.

For Automation the testing

The test harness server is also used for automating the testing on the packages, for example if a package is check in and the package is updated in the centralized server without testing then the test harness server tests the package, generates the report and notifies the user about the status. And also the users can automate the testing for the complete project when required but requires more processing power from the test harness server.

For storing the test case and versioning

The test harness server also used to store the test cases for each version of the code present in the repository. It can be said that the test harness server is the replica of the package

structure present in the repository but with the test cases. In simple the test harness server acts as the test case repository for SCF.

8.4 Packages and modules in Test harness server

The test harness servers uses all the components of the GSMS system which we have discussed earlier in this document. The test harness has the packages that are distributed among four layers which are discussed in the previous subsection. This test harness server acts as an execution server for executing the test cases and also acts as the database for storing all the versions of the packages in the database.

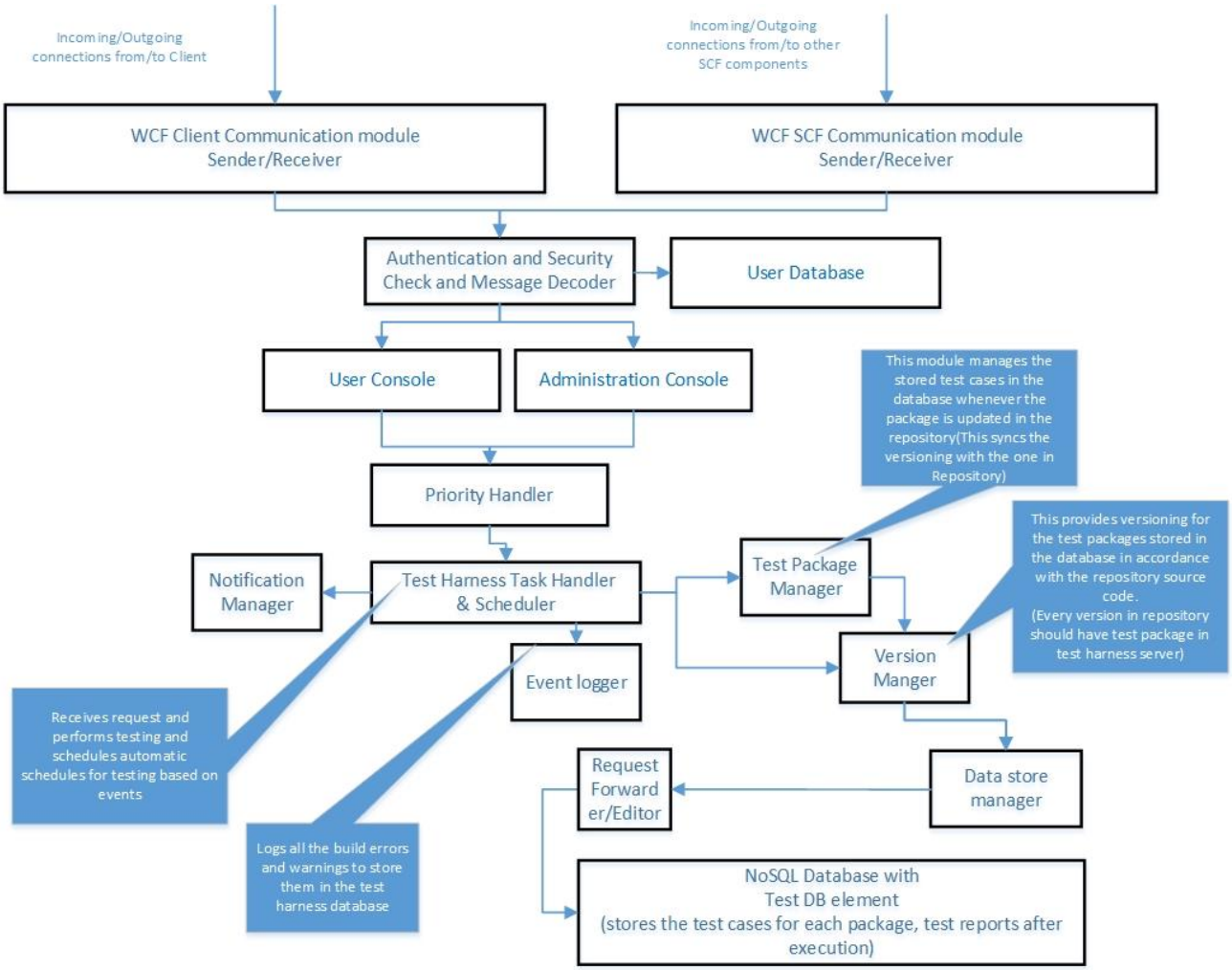


Figure: Package diagram for Test harness server with extension from the GSMS

Functionalities

The processing of the received messages in the test harness server remains same as that of the earlier discussed repository except the business logic and even scheduler and the DB element. The request that is authenticated by the Authentication and security module is

provided to the priority handler and after that the based in the priority the request is forwarded to the test harness task handler, the test harness task handler checks the request if the request is to store the test cases then the test package manager is called. The test package manger calls the data manger if there are any data conversion required. If no data conversion is required for the particular request then the data is sent to the NoSQL database. The NoSQL database uses the TestDB element to store the test cases in the database. As we are storing the files directly in the hard drive, the test DB element does not differ much when compared to the Generic DB element in GSMS.

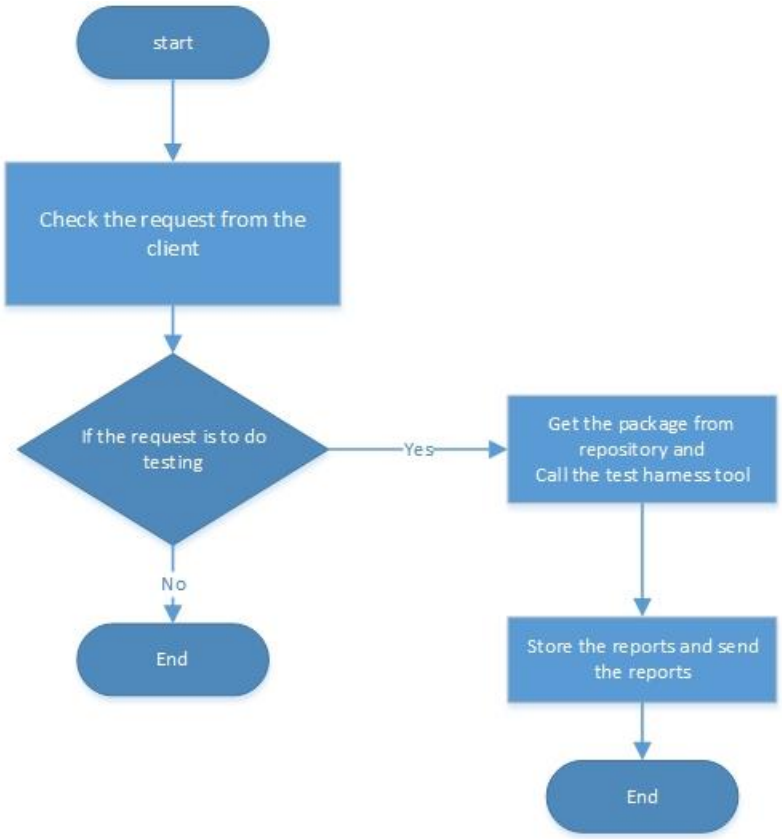


Figure: Activity diagram of Test harness server

The following modules are explained in detail with in the following subsections.

Interfaces

The following is the table of interfaces for the Test harness server. Using the below interfaces it is easy to develop the test harness server using the Generic Storage Management System.

Interface Name	Reason	With
Test harness task handler	To collect the request from the priority handler	Priority handler(GSMS)
Test harness task handler	Notification management	Notification manager (GSMS)
Test harness task handler	To log events	Event logger (GSMS)

Test package manger	To maintain version in the database	Version manager (GSMS)

8.4.1 Test Harness Task Handler & Scheduler:

The Test harness Task handler & scheduler which is the heart of the test harness server, It receives the request from other servers or from the users and checks if the user is requesting for any automation based testing or not. If the user is requesting for any automation based testing them it schedules the testing accordingly if the request is to store the test cases in the server then it directs the request to the package manager which will check with the centralized package manager and takes the necessary action.

8.4.2 Test Package Manager

The test package manager manages the packages that are stored in the test harness database. The test package manager also checks if the package is part of the registered package database or not. If this is not registered in the database then the test package manger throws an error saying that the test package is not registered with the centralized package manger.

If the package is already registered then the test package manger will direct the request to the versioning manage.

8.4.3 Test Versioning Manager

The test versioning manager creates the appropriate version as directed by the test package manger. Once the test package is stored in the database then the test versioning manager sends an acknowledgement to the test package manager which will update the version in the centralized database.

8.4.4 Data storage Manager

Data store manager plays the same role which we discussed in the GSMS, if the format of the data or files are not according to the standard then the data store manager will convert the data into required format.

The data format of the test cases in this case is .csv or .xml or .xls if the existing test cases are in .xls and the user is requesting to store the .csv file then then data manager converts the data into the .xls and stores the data in the database. The status of the same is logged and updated to the test package manager.

8.5 Activities

The following are the activities that a user or other SCF servers can perform on the Test harness server.

User requests to store the Test cases

The user login into the test harness server and request to store the new test case that he wrote. The test harness server will check if the package is registered with the same name or not. If it is registered it store by creating version if not provided by the user. Usually the user should request for the version of the test case he want to register.

User Request to download the test Case

The user can request for a test case already stored in the test harness server. The user does so by mentioning version and package name of the test case. If the user does not provide the version of the test case he wan tot down load the test harness server provides the latest available test case.

User performs Testing

The user requests a test to be performed on a package by providing the version of the package and package name. The test harness server checks if there is a test case available for that particular version if it is available the test harness server executes the test cases by downloading the code from the repository. The test harness server has the ability to connect to the repository and get the code from the repository and perform the testing.

Administrator Access

The administrator can access the Test harness server and he has access to modify any check in version of the database. The administrator has access to add or remove the users form the user database. The administrator also can undo changes if there is approval for the same. All these events are logged for future reference.

9. Client

9.1 Summary

Client is the front face of the SCF that is proposed in this document. Client contains the interfaces required to communicate with all the servers available in the SCF. The client application also contains the database of all the available servers and also ability to store all the user login information using a local instance of the NoSQL database.

The client has the following User interfaces to connect to the servers we mentioned above.

- Repository
- Test harness server
- Build server
- Virtual Display system
- Review DB server
- Requirement Database server

9.2 Client Package diagram

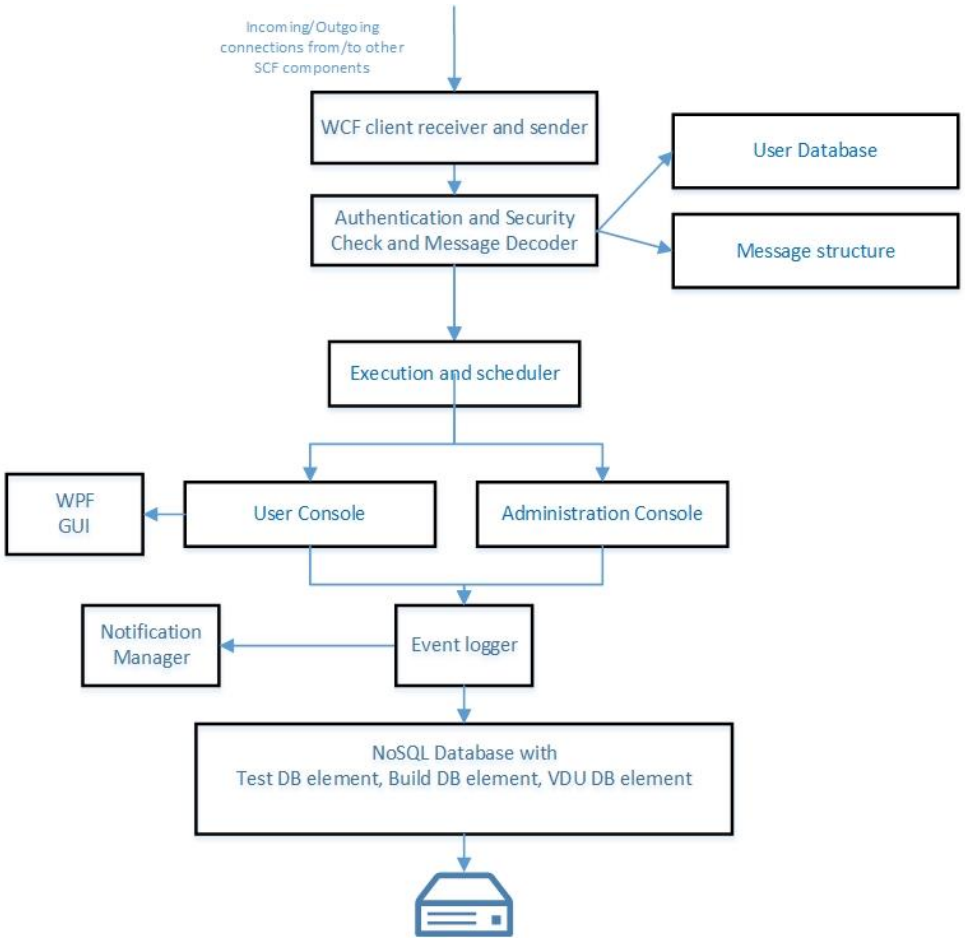


Figure: Package diagram for client system in SCF

The above picture shows the package diagram of the SCF client the SC client will have WPF GUI which is again broken down into three tabs one for test harness server, second one for build server and the third one for repository. The package diagram also has one user storage database and another one for storing the message structure of all the connected or registered servers.

9.3 Management

The client sends the message to all the servers across the SCF whenever the user sends pushes the send button. The received reply is read by the WCF client and the scheduler collects the information and forwards it to the WPF module to display to the user who requested.

Further the client will have the following sub clients which connect to different servers.

9.4 Functionality (Views)

The client has the following functionalities as per the design

- Connecting and viewing the SCF servers
- Storing the list of servers present in SCF network
- Storing the user login data and storing the log information in the database
- Provides the GUI to connect to all the servers in the SCF network

9.4.1 Connecting to Repository

The GUI uses the WCF communication channel that is a part of the proposed GSMS system. The client provides the GUI with tabs where the users can connect to the Repository, Using this repository the user can view the package structure like folder structure. The user also can do check-in, check-out, delivery operations on the package. Being said that the users also can perform operations like testing, build software using this GUI.

The GUI is provided below for understanding purpose.

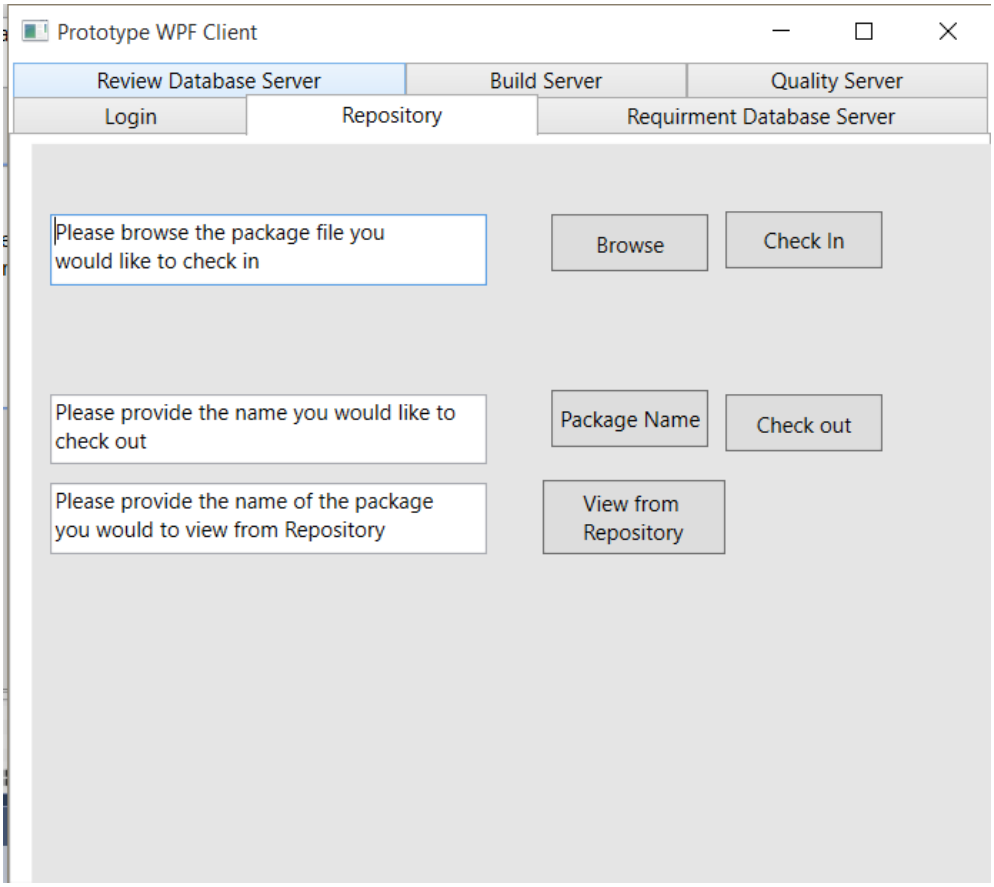


Figure: A sample view of the repository server GUI at the client side.

The user need to provide the package name to do operations. As of now only two operations are shown in this picture and later more details will be added during the project development. The user can provide either package name or the project name.

The GUI also provides the birds eye view of the packages however in this prototype we are not obliged to show.

9.4.2 Connecting to Test harness server

The client provide a separate GUI for test harness server. The user can provide the package or project name to test the software code. Once the user clicks the test harness button then the GUI connects to the test harness server. There can be many test harness servers in the network. But the GUI client decides which test harness server's latency is less and chooses the best among them.

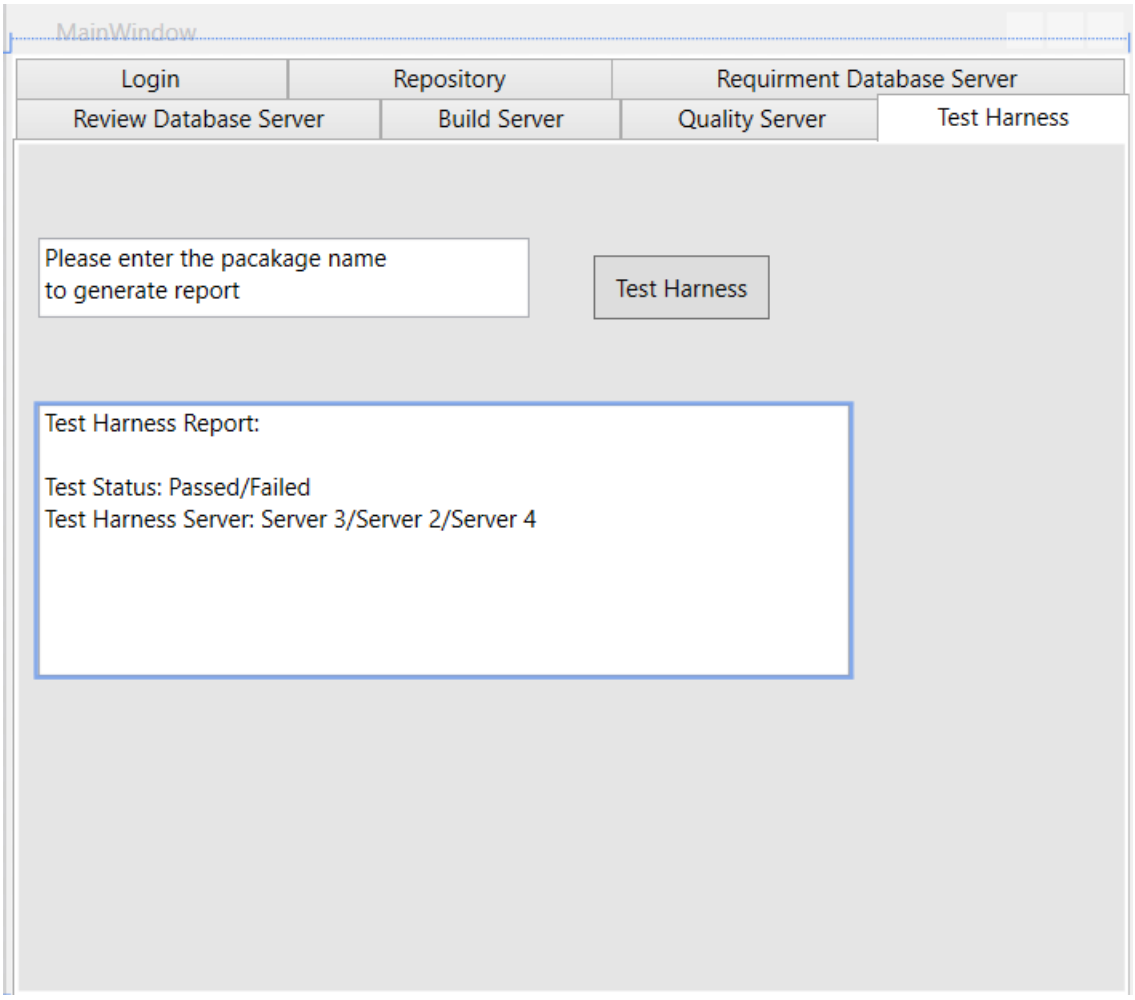


Figure: Test harness GUI client

Once the test harness is done the user gets to see the result. The sample result is shown in the above picture. The test results are also logged in the test harness server as well as the client database.

The test harness GUI provide the time taken for the test to finish etc.

9.4.3 Connecting to build server

The client also provide the GUI to connect to the build server. The functionalities the GUI provides are to give build just by specifying the package name and the build server will get the code from the repository. The GUI displays the result of the build along with results. The user also will have option to select if he want to see live update of the build status. In live status update the GUI displays the current running instruction and keeps on rolling displaying the status of the build.

The following is the sample GUI design for the build server.

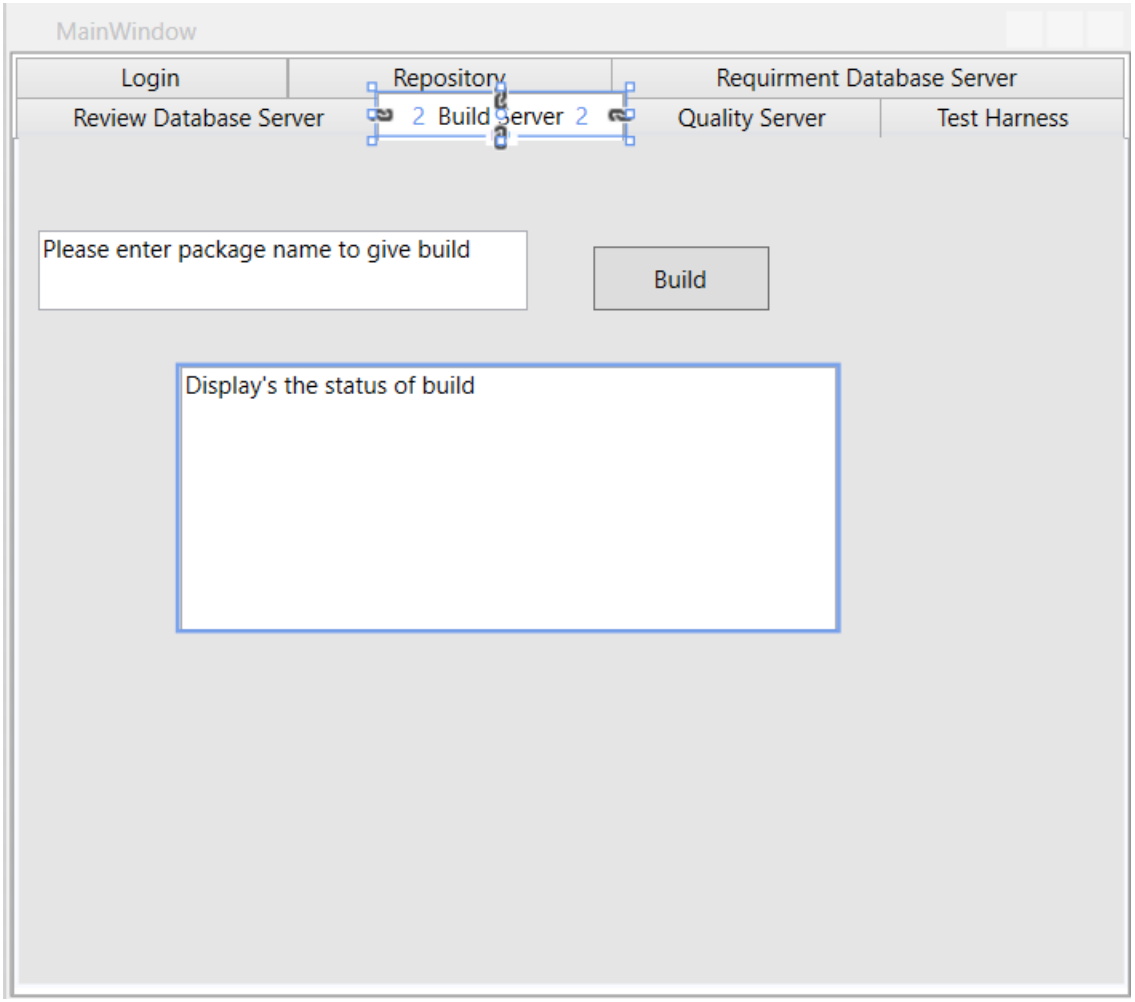


Figure: Client GUI for connecting to build server

9.4.4 Connecting to Virtual Display system

The virtual display system is used to support the display system in between the working team. The team sets up the display system where they can share the information. The client generates the email invitation and the user selects. All the invitations are sent to the team. Once the team starts joining the meeting the display starts. The virtual display system also involves the web camera using which the users have a good interaction with each other. The user can perform the following actions using this GUI

1. Send invitations

2. Conduct virtual meetings
3. Record the meetings content for future use
4. Share the files in the meetings

9.5 Data structure

The below is the data structure that the client uses to interact with the repository. The below Data contract has package list where the repository returns the list of packages matching the query sent by the client.

```
   [DataContract]
     public class RepoMessage
 {
          [DataMember]
       public List<RepoDBElement<key, value>> packageList { get; set; }
          [DataMember]
       public DBElement<string, string> package { get; set; }
         [DataMember]
       public List<Child> childList { get; set; }
       [DataMember]
       public List<string> keyList { get; set; }
       [DataMember]
       public string query { get; set; }
       [DataMember]

    public string fromUrl { get; set; }
    [DataMember]
    public string toUrl { get; set; }
    [DataMember]
    public string content { get; set; }  // will hold XML defining message information
  }
}
```

The below is the XML version of the message structure. This is detailed in the previous section where the complete message structure of the SCF is dealt.

9.6 Analysis

From the above sections we have seen that the client plays major role in connecting the SCF with the actual users. When we analyze this, the main part of the client is the WCF communication module.

9.7 Critical issues

The following are the critical issues that are identified in the client.

Connecting to multiple servers

The client may need to connect to multiple clients at a time, hence there can be more latency when the client tries to connect to all the servers at a time.

Solution

However, the solution is to use the events to read and write to the WCF reader and writer clients. Doing this the un necessary load on the client system can be avoided.

Selecting the best server out of many instances

As there can be may test harness servers and as well as many build servers the client will have many options to connect however the client is not aware of which server is loaded at particular time. As the load on any server can be dynamic, there are days when a particular server can be overloaded, there are no methods which says which server is loaded and which one is free.

Solution

The solution for this issue is the client pings the servers one before sending them the long instructions to execute. Then the client calculates which is one is minimum, likewise the client sends request to the minimum latency server.

10. Build Server

10.1 Executive summary

The build server is the most crucial component of the SCF. The build server takes request from clients and other SCF servers. Once the request taken the build servers downloads the package data from the repository and executes the build command on that folder. Once the build is successful the reports are stored in the build server Bo SQL database and the also sent to the requestor. Incase if the build is not through then also the build server stores the information in the database and forwards the result to the requestor notifying the result of the build.

In few cases the build server receives the request asking for the status of the build based in the request the build server sends the status of the build whenever there is change or intermittently.

10.2 Functionalities

The following figure shows the package diagram of the Build server it uses all the components of the GSMS we discussed earlier hence this section explains more on the functionalities of the Build server.

Once the build server receives the request form the SCF servers or from the client, the build server requests the package or the project information from repository. If the project is not present then it throws the error. If the project is present then it downloads the code files and executes the build command on the project folder. if the build is successful then the build report is sent to the requestor for download and also stored in the database for future reference.

The clients can request for two kinds of status update.

1. Build status update intermittently
2. Build status update on event based
3. Build status update after the build is done

Based on the request of the client the build server sends the response.

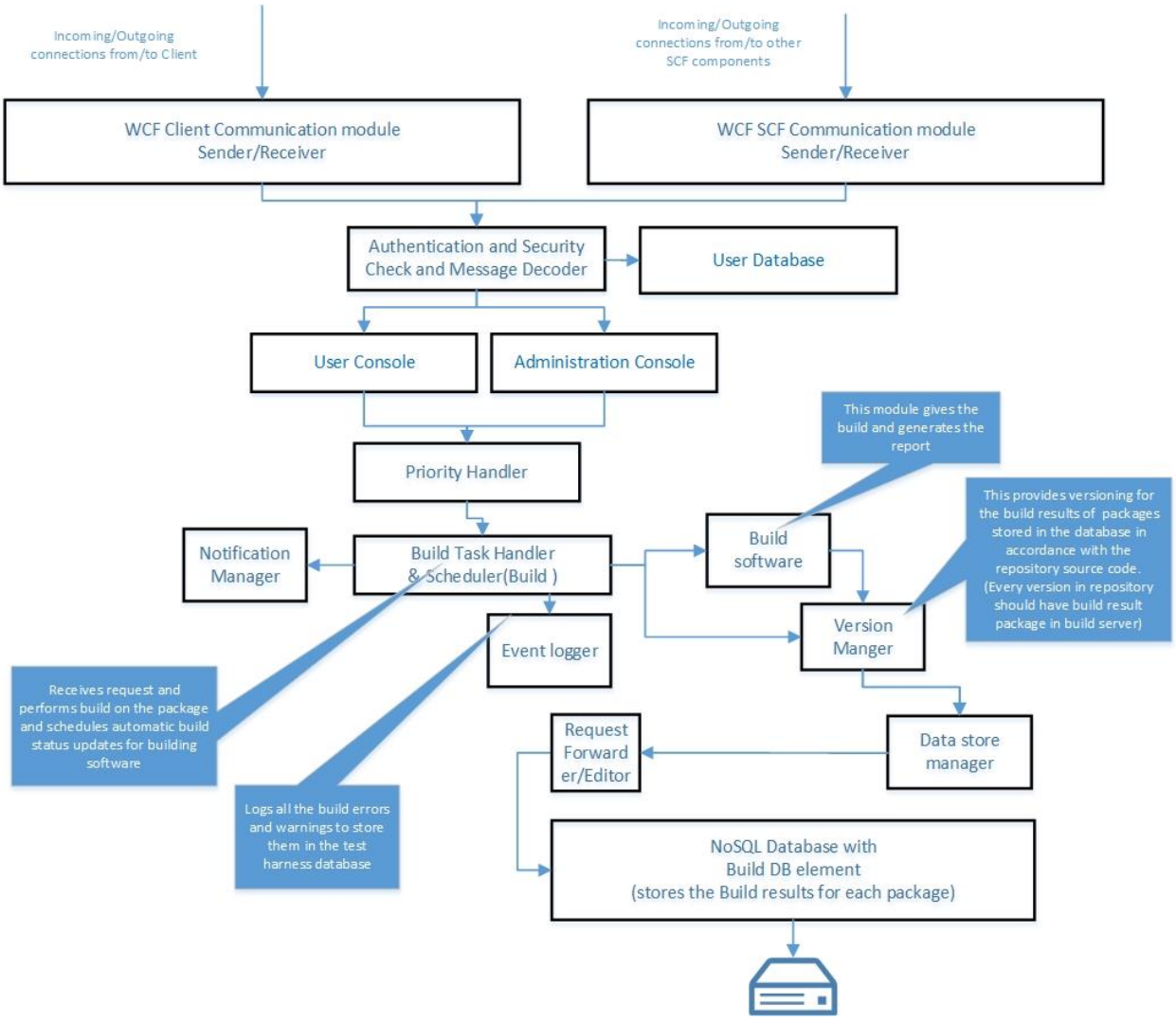


Figure: package diagram of the build server

The build task handler forwards the request to the build software block which executes the build instructions on the downloaded folder.

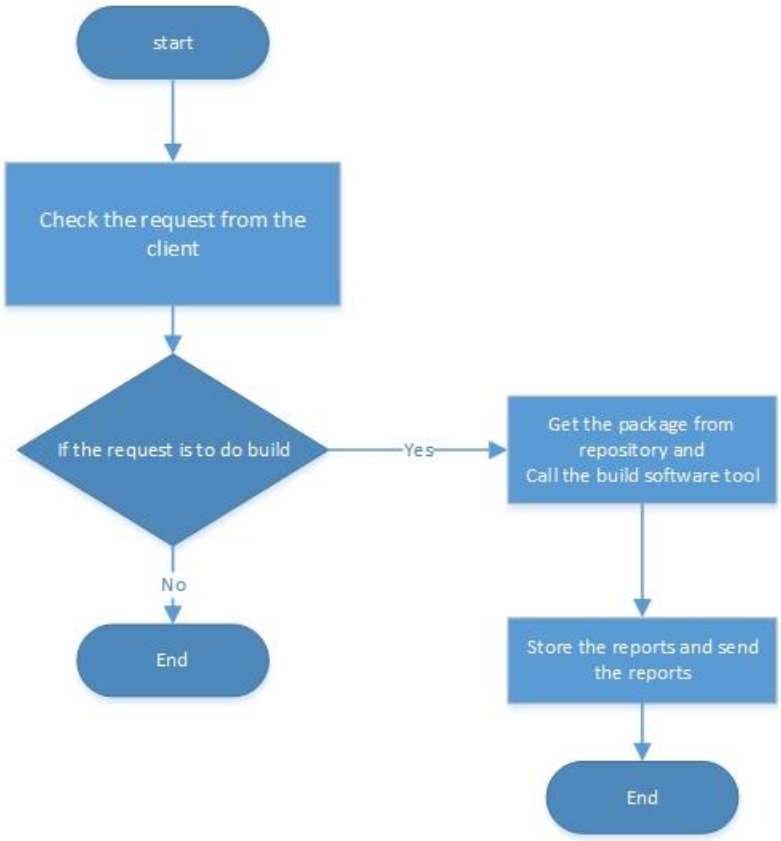


Figure: Activity diagram of Build server

10.3 Interfaces

The main interfaces of the build server with the GSMS system are Build task handler which is the executive part of the build server. Except the business logic part of the

Interface Name	Reason	With
Build task handler	To collect the request from the priority handler	Priority handler(GSMS)
Build task handler	Notification management	Notification manager (GSMS)
Build task handler	To log events	Event logger (GSMS)
Build software	To maintain version in the database	Version manager (GSMS)

10.4 Critical issues

Long time running

The build server runs for long time and the number of concurrent requests the build server can run is very little when compared with the repository.

Solution:

The build server number of servers should be increased and auto configurable. Which means when the build service receives the request more than it can handle the build server acts as proxy and forward to the other servers which are free. Apart from this in the SCF system we also implement test ping by the client or any other requestor. The test ping says if the server is busy if so how long it will take for the server to be free. While this is already implemented in the client. As a precautionary measure we can implement request forwarding with auto configuration.

Scalability

Scalability is and will be always an issue with respect to server handling. In modern day software systems scalability is always a major issue, as the number of software developers increase in a company or when a company wants to start a new venture in different location, often software companies face the challenge of the scalability.

Solution:

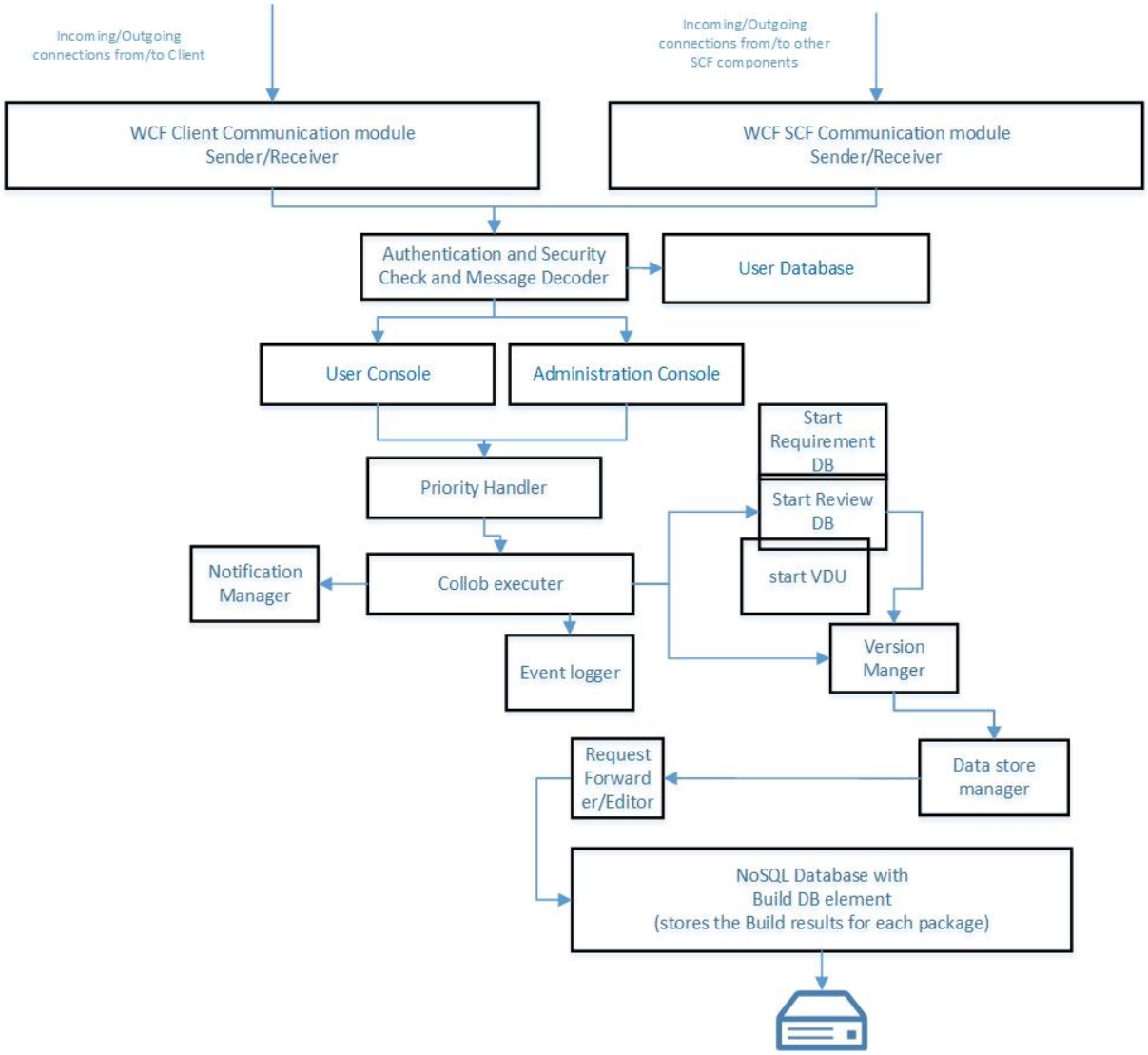
As it is already discussed earlier the repository system is designed to exponentially increase the number of build servers in the network without any limit hence there will not be any issues with the scalability of the servers.

11. Collaboration Server (Virtual Display system, Review Database, Requirement Database)

11.1 Executive summary

In collaboration server there can be many servers in the SCF system like the virtual display system, Review Database and requirement database all of the requires to store the data in the database. For example the virtual display system needs to store the project meeting status in the database. Hence the collaboration server uses the GSMS we are developing in this document.

11.2 Package diagram for Collaboration server(Functionalities)



In the above picture the collob executer communicates with VDU which virtual display unit, requirement database and review database. The Collob executer receives the instruction on what to be done lets say there has been some data generated from the meeting and the user want sot log in to the server and store the data in the database. Then the user sends a request

to the Collob server. The collob server collects the data analyzes it and stores in the database and hence can be recovered any time from the database in the later point if it is required

The following is the activity diagram of the collob server.

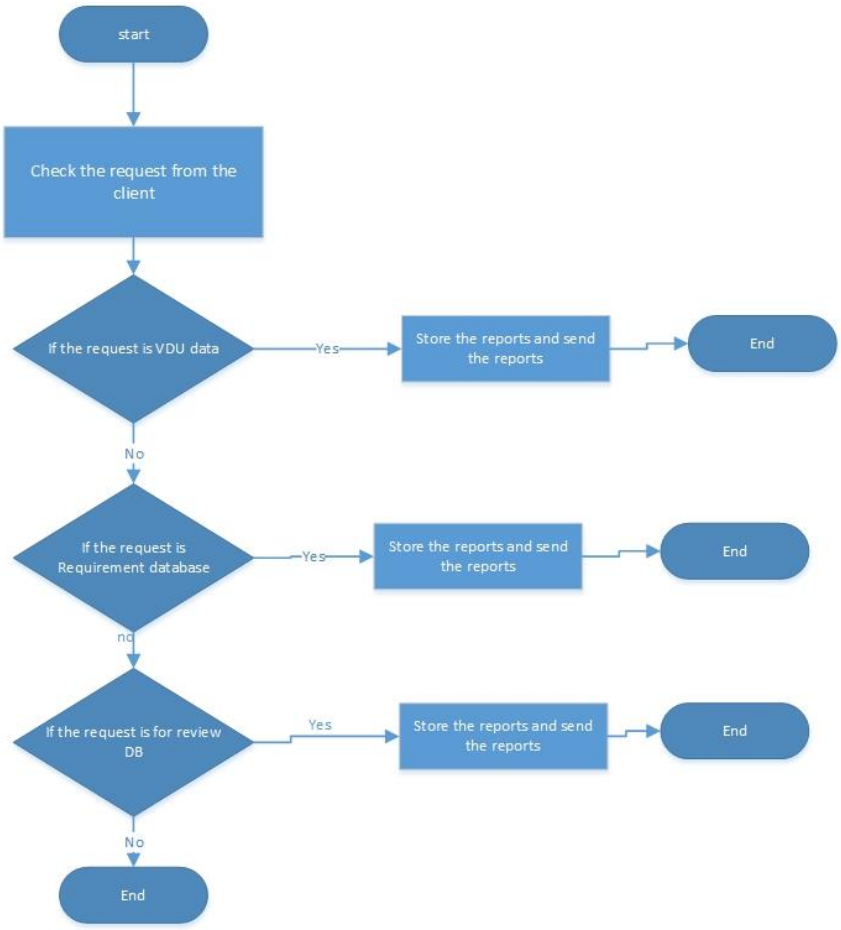


Figure: Activity diagram of the collob server

The collob server once receives the request from the priority handler processes the request it checks the request for which tool and instantiates the tool after the tool work is done the tool writes down all the data to the database which is present in the backend.

11.3 Interfaces

Interface Name	Reason	With
Collob task handler	To collect the request from the priority handler	Priority handler(GSMS)
Collob task handler	Notification management	Notification manager (GSMS)
Collob task handler	To log events	Event logger (GSMS)
VDU tool	To maintain version in the database	Version manager (GSMS)
Requirement database tool	To maintain version in the database	Version manager (GSMS)

Review DB tool	To maintain version in the database	Version manager (GSMS)

Table: interfaces for the collob server

11.4 Tools

The collob server uses three tools as follows

1. VDU tool
2. Requirement Management tool
3. Review management tool

VDU tool

The VDU is Virtual Display system, which means the online conference tool for software development team who work from different location, when the users want host meeting they access this tool via the Collob server. Once the meeting is done the tools retrieves the data and stores it in the database. To store these data we use the Storage management Subsystems in this module.

Requirement Management tool

The requirement management tool keeps track of the requirements in the project and attaches them to the versioning of the package. For example a customer requests for a enhancement the project manager creates a requirement in the tool and assigns the requirement on the employee name. Which means the developer is the owner of the requirement. Once the developer is done with the development and checks in the code the repository intimates the requirement management tool to close this requirement as the package is checked in. The tool automatically delivers the software to the actual requestor of the requirement.

Review DB tool

The review DB tool is used to maintain the reviews in the server. For example a developer develops a code and sends for peer review. Now the peer reviews the code in the server and provides some review points to check. To maintain these point we need the review DB tool in storage management subsystem. Now the Review DB tool stores all these review points and complete details in the server, which can be accessed later stage.

11.5 Critical issues

12. Prototypes:

Prototype1:

Prototype1 attached will log the error in the server in the log file. It is taken from internet and modified.

Prototype2:

Prototype 2 implements the NoSQL database ad creates a server and WPF client which.

Protype3:

I learnt from online from codeproject.com about task parallelism and submitting this code as prototype.

13. References

[1] Key Value Database requirements provided by Dr. Fawcett

http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/lectures/Project1-F2015.htm

[2] Key value database project #2 requirement provided by Dr. Fawcett

http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/lectures/Project2-F2015.htm

[3] NoSQL blog written by Dr. Fawcett

http://ecs.syr.edu/faculty/fawcett/handouts/webpages/BlogNoSql.htm

[4] Comparison of database management systems

https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models

[5] DBEngine, DBElement, UtilityExtensions, Display code provided by Dr. Fawcett

http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project2HelpF15/

[6] Heterogeneous Database information provided by wikipedia

https://en.wikipedia.org/wiki/Heterogeneous_database_system

14. Appendix

14.1Prototype GUI screens

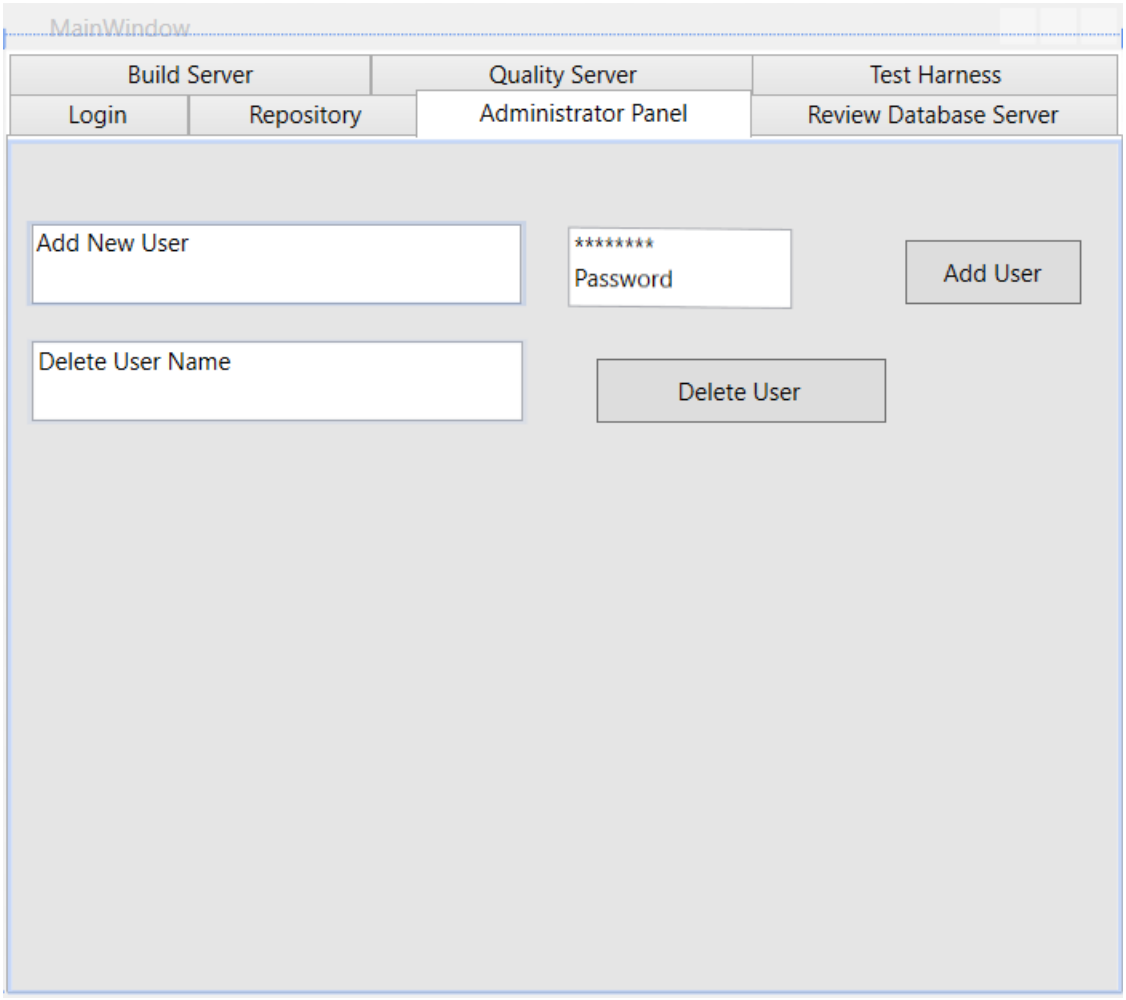


Figure: Administrative panel GUI design

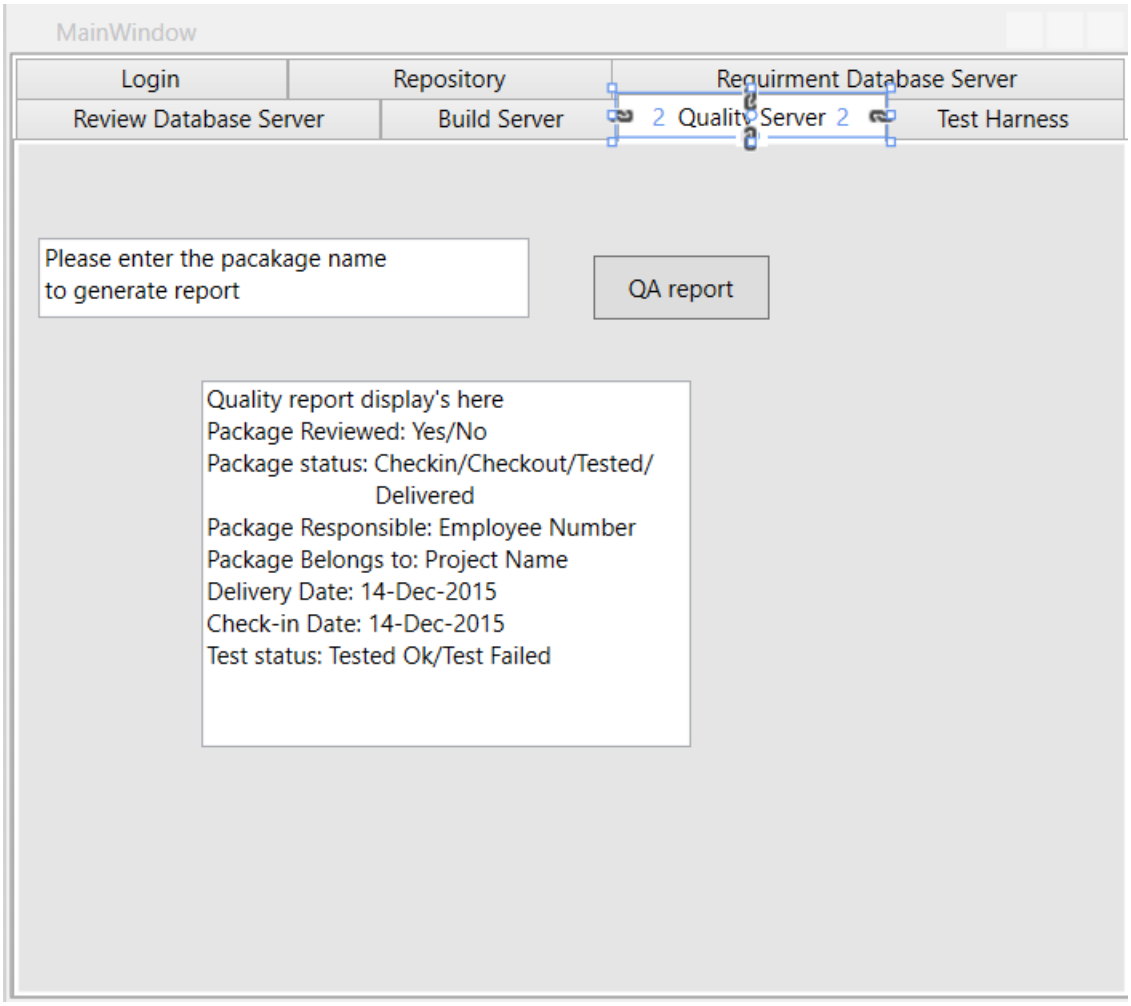


Figure : Quality Client GUI design

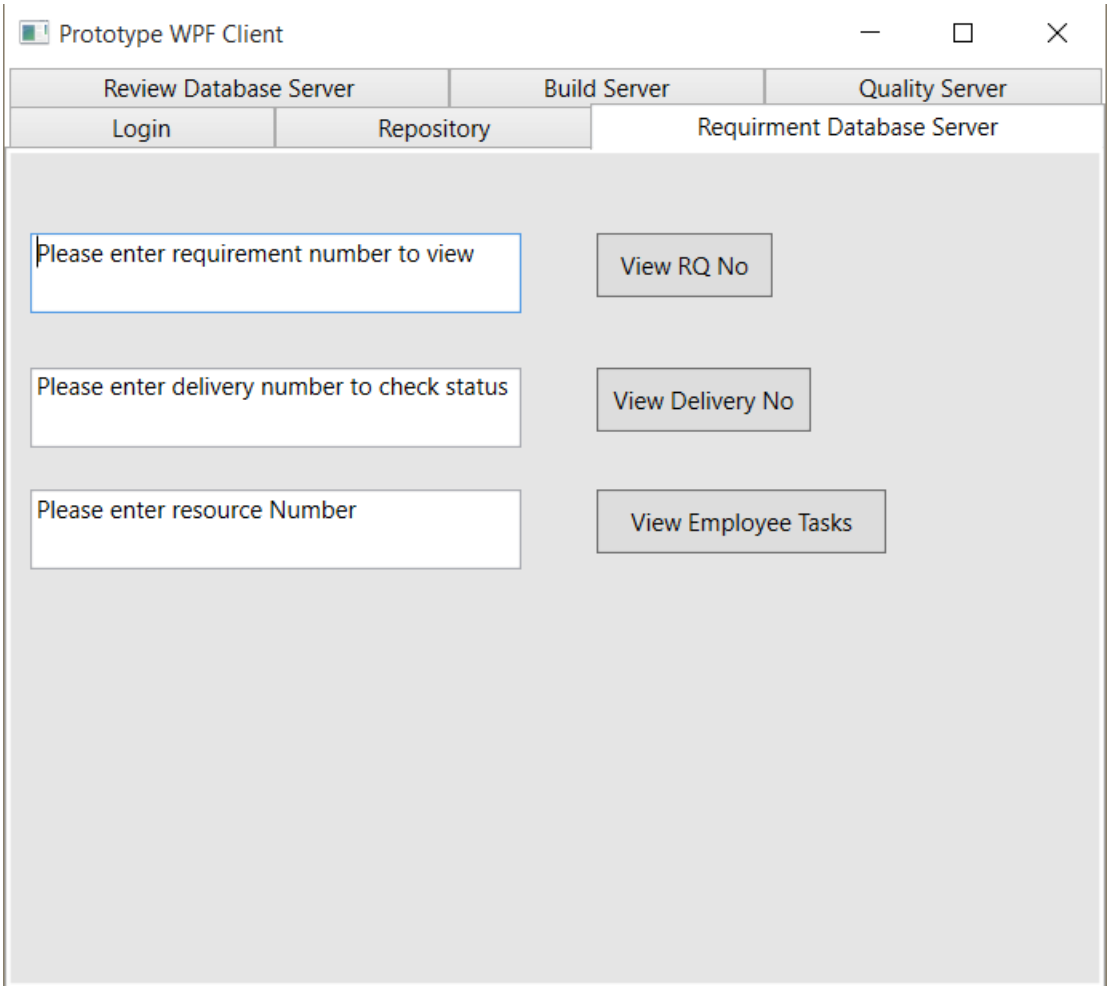


Figure: requirement Server GUI design

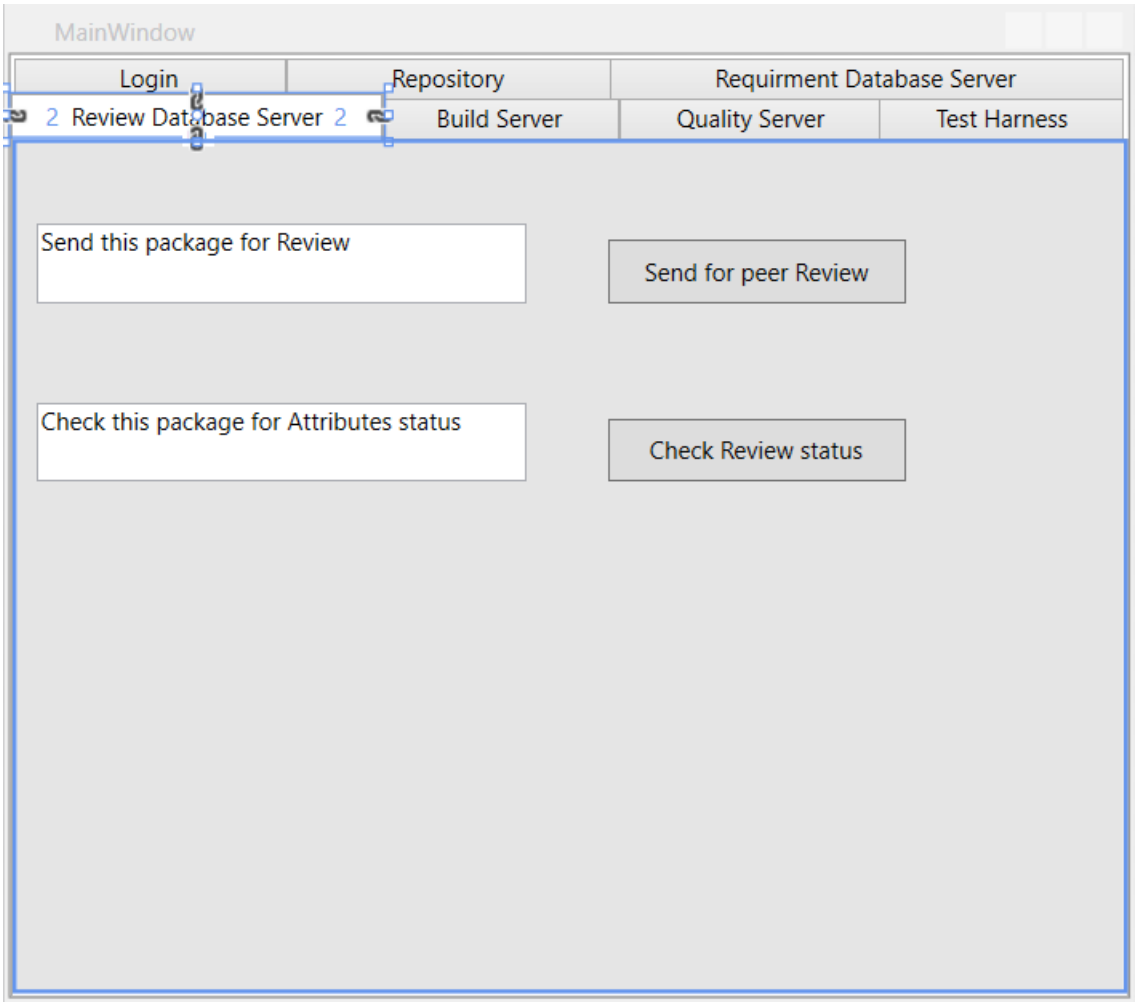


Figure: Review DB server Client GUI screen

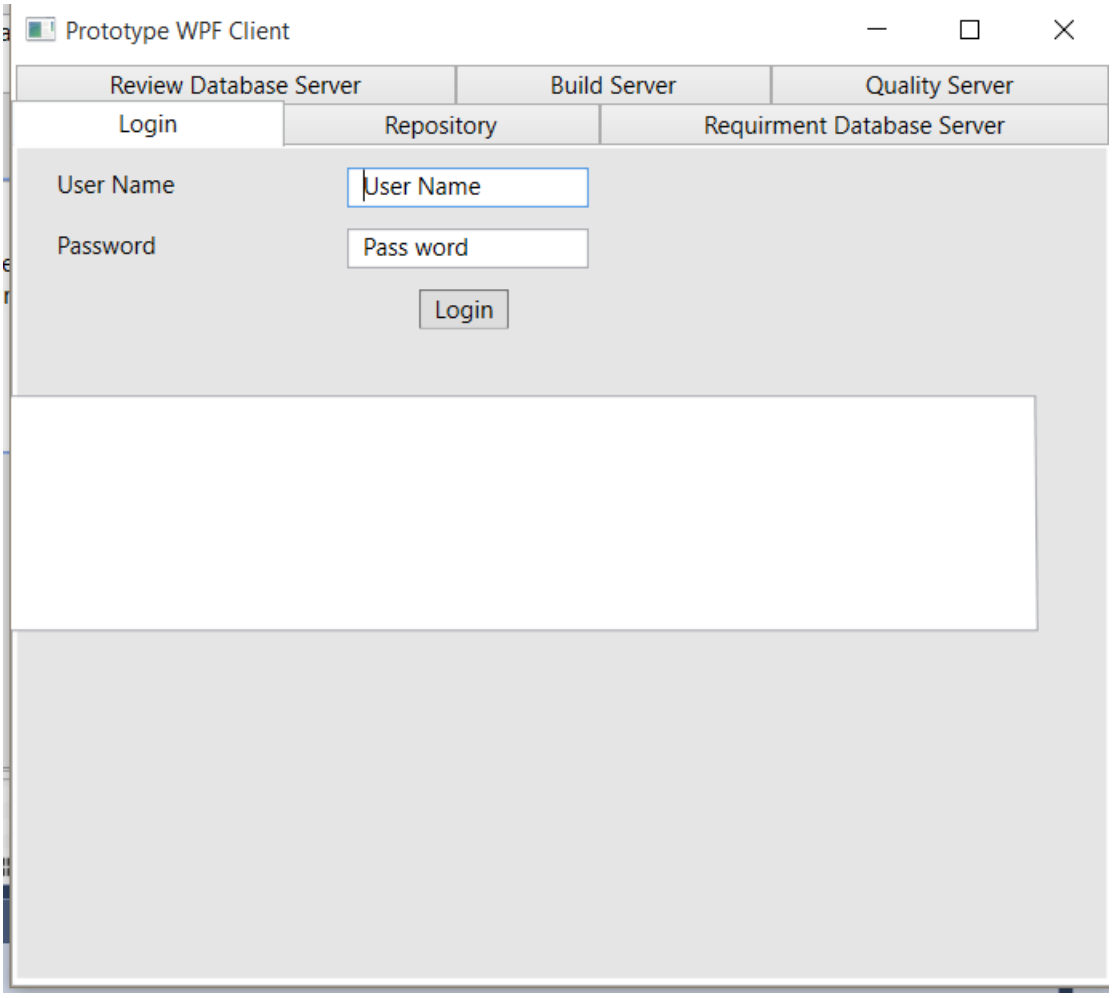


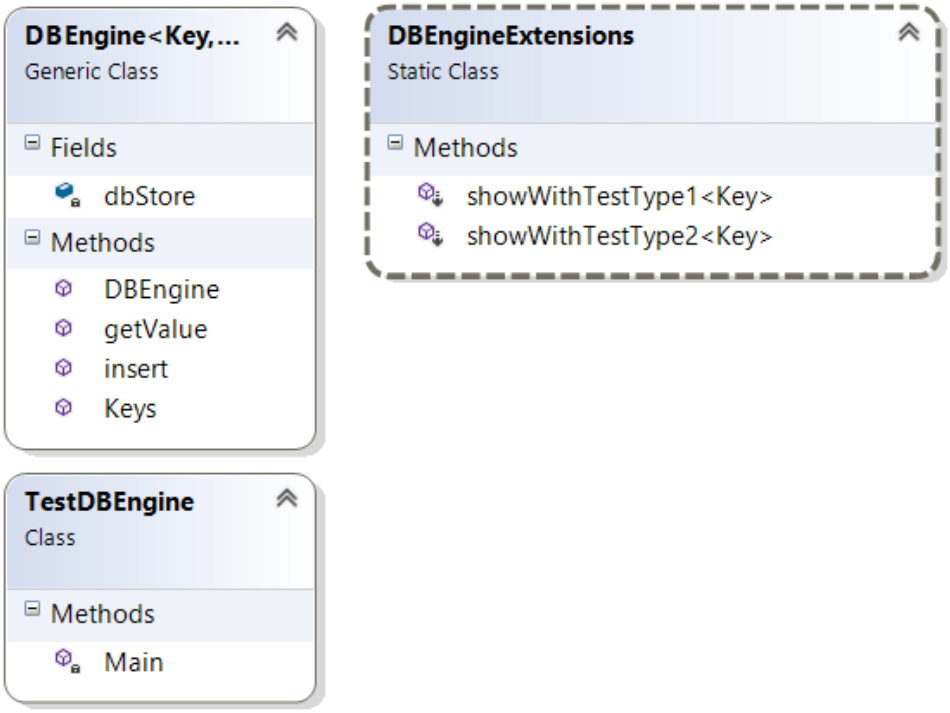
Figure: login screen for Client GUI

14.2 Sample status display screen

a. Connecting to server
b. Server busy query entered into the queue
c. Network is busy
d. Written successfully

14.3 DBEngine

The following class diagram will help to understand to the DBEngine package.



The DBEngine has insert function, getValue function and keys function.

Sample output of the DBEngine is provided for reference below.