**Source Code Repository Architecture**

Operational Concept Document

Yufan Gong

SUID: 412179310

CSE 681-Software Modeling and Analysis

Dec 8th 2013

1. Introduction 3

1.1 Executive summary 3

1.2 Brief architecture 3

1.3 Requirements 4

2. Context 6

2.1 Context Diagram 6

2.2 Virtual server 7

2.3 Project server 9

2.3.1 Concept 9

2.3.2 Partitions 9

2.4 Test server 12

3. Code Repository 13

3.1 Interaction between Server and Client 13

3.1.1 Peer-to-Peer structure 13

3.1.2 Object Diagram 15

3.1.3 Activity Diagram between Server and Client 16

3.1.4 Structure Diagram between Server and Client 16

3.2 User and uses case 18

3.2.1 Software Development Company 18

3.2.2 Software Architect 19

3.2.3 Developer Team Leader 19

3.2.4 Developer Team Member 20

3.2.5 QA 21

3.2.6 Manager 21

3.2.7 Customer 21

3.2.8 Server Administrator 21

3.2.9 Interns and students 21

3.3. Repository server 23

3.3.1.Concept 23

3.3.2 Activity 25

3.3.3 Structure 27

3.3.4 Interfaces 31

3.3.5 Relationships: 32

3.4. Clients 33

3.4.1 Concept 33

3.4.2 Activity 34

3.4.2 Structure 36

3.4.3 Views 41

3.5 Critical issues 45

3.5.1 Files name conflicts 45

3.5.2 File caching issues 45

3.5.3 Huge file sets transfer 46

3.5.4 Security issues 46

4. Services and policies 47

4.1 Services 47

4.1.1 Check-in/check-out service 47

4.1.2 Metadata 47

4.1.3 Notification 48

4.1.4 File caching 48

4.1.5 Update version 49

4.1.6 Back version 50

4.1.7 Track access 50

4.1.8 Query 51

4.1.9 Navigation 51

4.1.10 Communication 52

4.2 Policies 53

4.2.1 Ownership policy 53

4.2.2 Check-in policy 54

4.2.3 Versioning policy 54

4.2.4 Notification policy 55

5. Appendices: 56

Prototype 56

6. Conclusion 57

# 1. Introduction

## 1.1 Executive summary

This OCD comes up with the Source Code Repository, who is the mainly part of the Software Development Collaboration System. The purpose of this Source Code Repository is to support management of large sets of document files and share files between users. It has a collaborated Server-side and several Client-sides.

Client-sides make insertion and extraction of files to and from the Server, and the users interface in Client-sides will display information about files properties and relationships in Sever-side Repository. In addition, users at Client-sides can require for making queries into Server-sides Repository to search for specific files contain information users input.

This architecture document first talk about the context of Source Code Repository, SDCS, and introduce other two servers briefly. Then it focus on Repository Server and Clients, discuss all parts about this server and clients, including concept, users and uses cases, activities, packages, and critical issues and so on.

This Source Code Repository has several critical issues like security issues, huge file to transfer, run over file cache and so on, these critical issues will discussed later in detail.

## 1.2 Brief architecture

The Source Code Repository has a united Server-side and more than one Client-sides, as figure shows below. This OCD document will introduce the communication between them and processes run in them use one-client and one-server module, which is clearer to describe.

The Repository server stores a huge set of files, Vault Client can insert and extract to or from it by communication with Server using WCF.

The server may be a virtual server not exists as the Figure 1.1 shows.

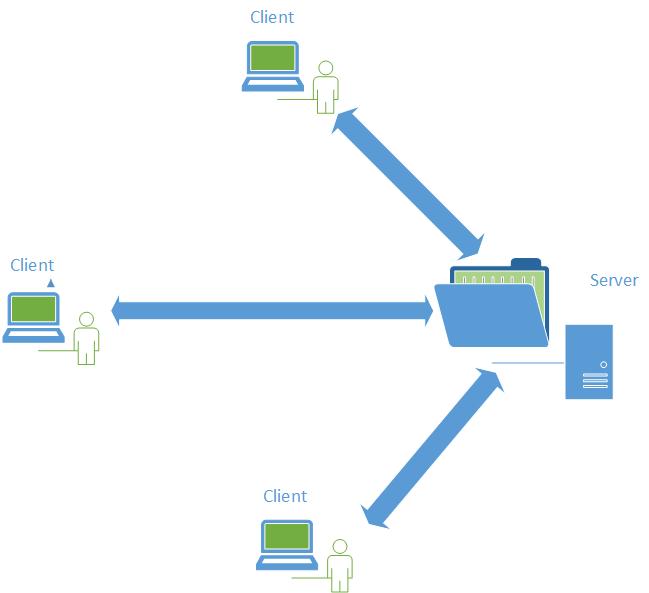


Figure 1.1 Brief Architecture

## 1.3 Requirements

* The Source Code Repository client shall support insertion and extraction of files and their metadata to and from a remote Source Code Repository Server using WCF.
* The Source Code Repository Client shall have a GUI build by WPF to provide information views to users.

The information view include a categories (projects)/files view show that all categories or projects and files in Server Repository; a file view which has two text boxes to show text file content, metadata file contents, parents of selected file and children of it; a query view which make query processing more clear and easy to use for users; a check-in view make users can select text files and create their metadata to check-in, users can also use this view to edit metadata stored in Server.

* The Source Code Repository Client shall provide clickable list of files for users to view selected file or change current file.

Files in categories/files view, and files in parents panes and children panes can be chose and enter file view.

* The SCR shall provide a Navigation tool to traverse metadata files and build a virtual graph of parents and children relationship.
* The SCR Client shall provide a check-in view to browse local file sets and check-in files into Repository. In this view, it should also have a metadata editor to help user create a metadata file associate with source code file user want to check-in. User need to input some key values like children dependencies, projects it belongs to, then the Client will build this metadata automatically.
* The SCR shall accept a text query message and return the fully qualified names of every file containing that text in a clickable list.
* The SCR shall accept a metadata query message specifying metadata tags and return the contents of each element for every file in the file sets.
* The SCR shall support editing metadata files by supplying a set of tags names and values.

# 2. Context

This Repository is one member of a server federation, Software Development Collaboration System. In this section, we will have a brief discussion about these members’ functionalities and their interactions.

We will also discuss the virtual server, how it implement in SDCS.

## 2.1 Context Diagram

SDCS supports maintenance of project baseline, testing, scheduling and collaboration and development. The four members shown below implement these separately.

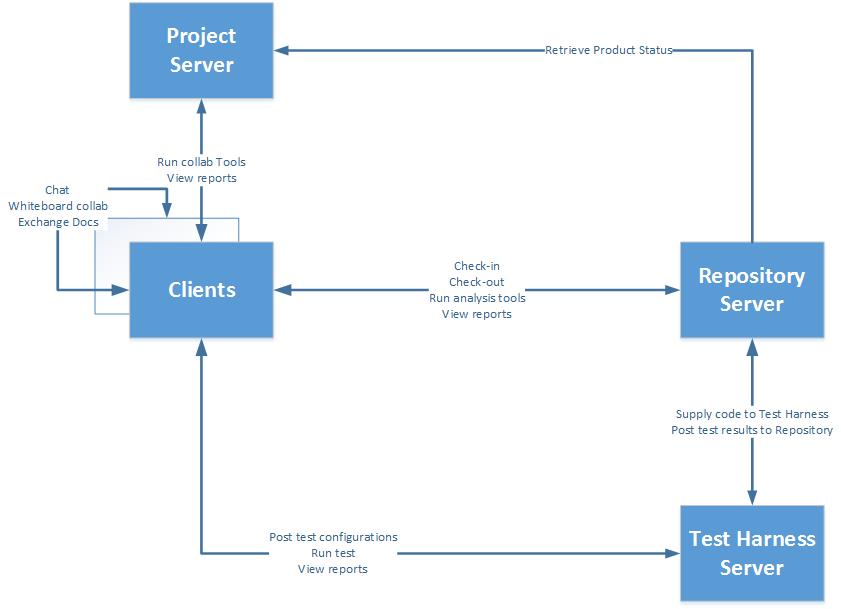


Figure 2.1 Context Diagram

Repository Server

Repository Server holds all projects’ baseline, source code, test drivers, test results and documentation. It also provides analysis tools like Query Handler to help users analyze the Repository storage.

Repository retrieves product status from Project server, Work packages, scheduling, and all information about specified user’s development. The virtual server instance only has this user’s accessible project data. Repository also pass notification message to the Project Server.

Repository supply source code to Test Harness to test, Test Harness Server run certified tests on certified code then post test results to Repository.

Clients are able to check-in files and metadata to Repository baseline and check-out files and metadata from it. Clients can also use analysis tools provided by Repository.

Project Server

Project Server provides management information to Repository, as we talked above, it also provides collaboration tools to clients, clients can use this tools to chat with each other, see notification on Scheduling, post information on Whiteboard, get knowledge from Wiki and so on.

Test Harness Server

Test Harness play a role to test code user checked-in, it will judge is this file work successfully separately and in project. Then post results to Repository.

Client

Client is the interface provided by Software Development Collaboration System, users do activities on client-side to implement all functionalities of SDCS.

About Project Server and Test Harness Server, we will have a concise discussion later.

## 2.2 Virtual server

This Software Development Collaboration System is designed as distributed virtual server. All servers can be run in virtual environment, server contents can be distributed across several server instances. Virtual server is not defined by boundaries. It may have multiple servers on one machine or multiple machines implementing one server.

Virtual server implementation:

Virtual server is easily replicated. The virtual server instances provide a cloning facility that allow authenticated user to create an instance on local, which means the SDCS will create an instance that only has this user’s accessible data by using copy constructor to create a subset of servers and download this instance to user’s local desktop. User can operate server functionalities locally, check-in files and test locally, may be a project virtual server, team virtual server, or company virtual server. After user modified local virtual server instance, this modified server instance can be checked-into the source server with new products user added. Note that not all cases need to create this local virtual server, local virtual server support test and check-in locally but will occupy a huge resource to download server data to local desktop.

Virtual server uses:

**Each project has virtual SDCS.**

It has Repository to manage all certificated project products, code baseline, test driver and test results, and documentations. It has Project Server to manage this project’s data, work package, schedule and so on. It also has the Test Harness to test check-in file.

**Each team has virtual SDCS.**

It provides local management for each team. The Repository in this situation holds all projects’ product developing by this team. Project Server stores all projects’ management information. And Test Harness has all tester for these projects.

**Company has virtual SDCS.**

Company’s virtual servers need to hold and manage company’s reusable code base. It must have Repository to store reusable code. It may have Project Server and Test Harness; it’s up to the company’s responsibility in software development.

Figure 2.2 gives us a clear structure about layered virtual server.

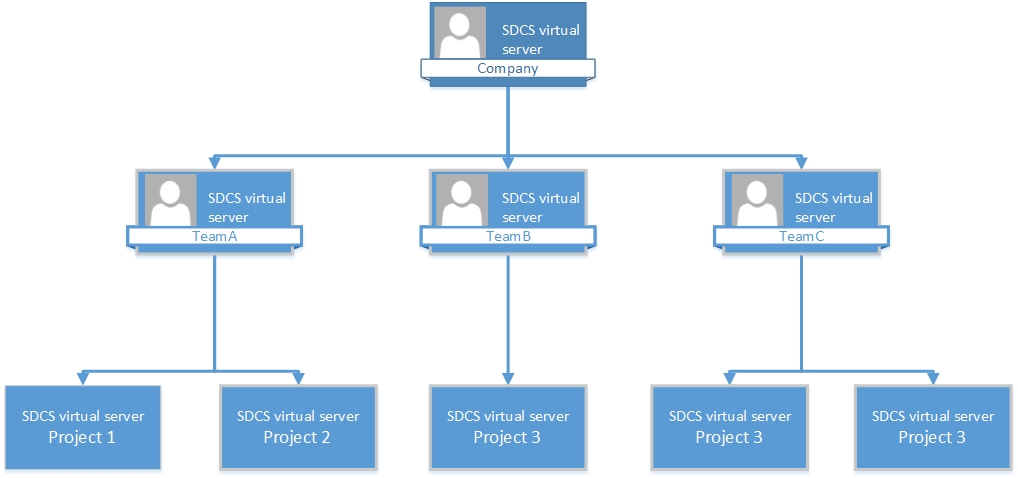


Figure 2.2 Virtual Server Module

## 2.3 Project server

#### 2.3.1 Concept

Project Server supports work scheduling and all collaboration tools. Everything of value concerning definition of work and scheduling goes on Project Server, it stores and publishes project management information for developers to plan conveniently. It also supports collaboration communication through virtual meetings like Chat application, Whiteboard, Wiki and so on.

#### 2.3.2 Partitions

The Project Server has these core packages shown in Figure 2.3 below:

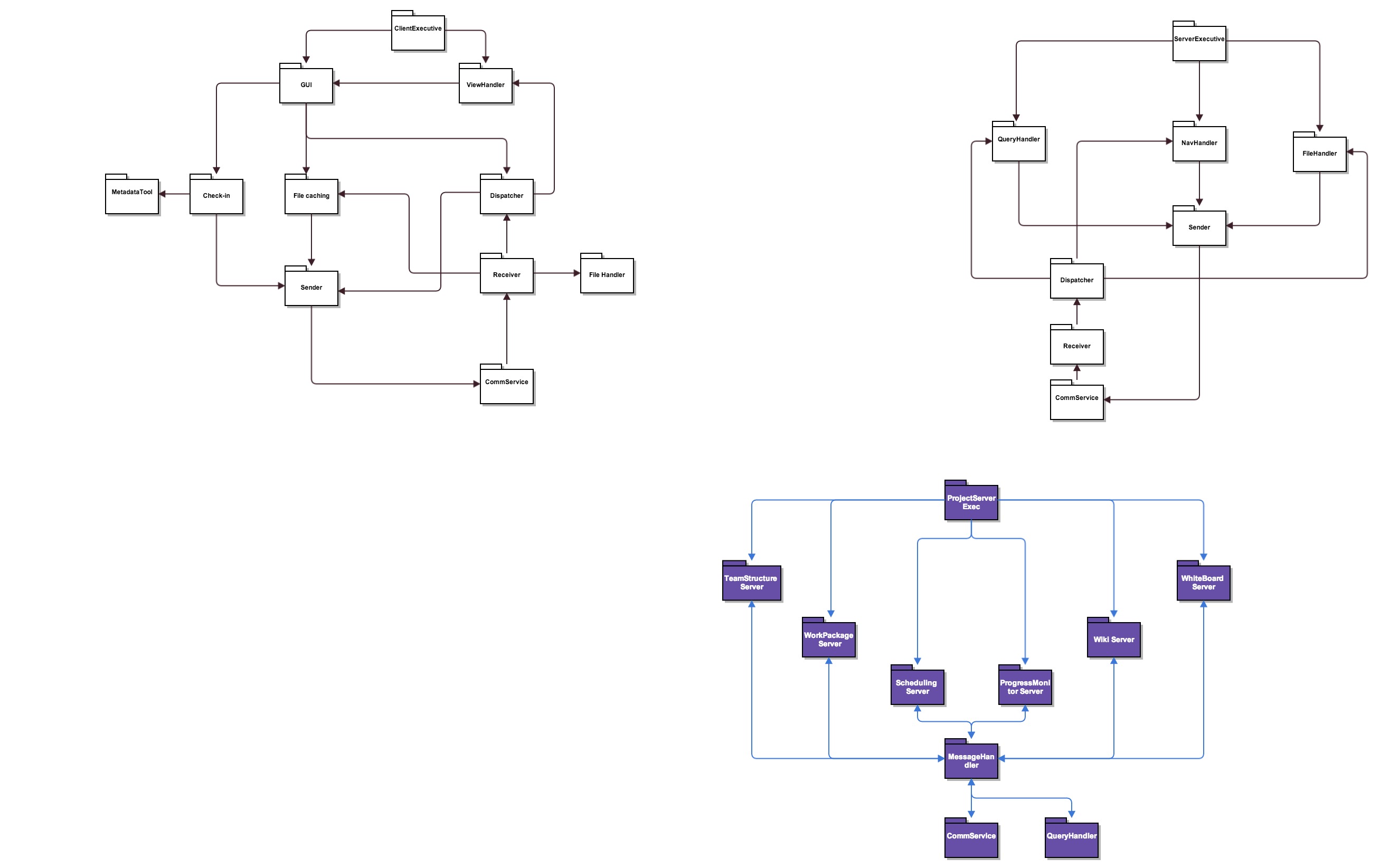


Figure 2.3 Project Server Package Diagram

The main responsibilities of each package are discussed below.

ProjectServerExecutive

ProjectServerExecutive is the controller of the Project Server. It determines the sequence of the packages in Project Server to active. It is also the start of the Project Server.

Whiteboard Server

Whiteboard is a non-real time communication application, like a forum. User can post their thoughts, notices on whiteboard. Whiteboard Server is used to manage this application, dequeue messages and put them on this application, or modify, delete content when users request.

Wiki Server

Wiki is a web application which allows users to add, modify, or delete content in a collaboration with others. Wiki here help developers share knowledge of development, note taking of progress, etc. Wiki Server provides functionalities to support Wiki application, using mark up language to write down content. Wiki also need to save former content and inform users who have modified the content.

ProgressMonitor Server

ProgressMonitor Server monitors the development progress. User can see projects progress accessible to this user. The team leader can see all projects under team then modify plan to develop them, and company administrator can manage all projects under it with help of ProgressMonitor server.

ProgressMonitor carries out this functionality by check how many dependencies do not exist in the Repository currently.

Scheduling Server

Work schedule managed by Scheduling Server, it mentioned in former parts that update notification and merge notification will pass from Repository to Scheduling. Scheduling application aims to help users know what are the next steps, what kind of functionalities needs to be implemented now and which files need to be checked-in, etc. It also has a long-time schedule help developers to make their plans.

WorkPackage Server

This package aims to manage work packages stored in Repository. It integrates information about each project in Repository, clearly points out work package’s owner, who has access to check-out and modify package. The relationship of packages stored in Project Server is like the structure in Repository.

Each time SDCS initiate, Project Server receives the user information and find work packages this user have access to, reply message to Repository and construct a virtual server which only contains this user’s work package.

Each time Repository update version in its storage, Repository sends message to WorkPackage server, and the information and relationship stored in WorkPackage server also updates follow the Repository.

TeamStructure Server

Team structure is the higher level of the work package structure. TeamStructure Server stores team relationships and assign projects to teams. The structure is shown below.

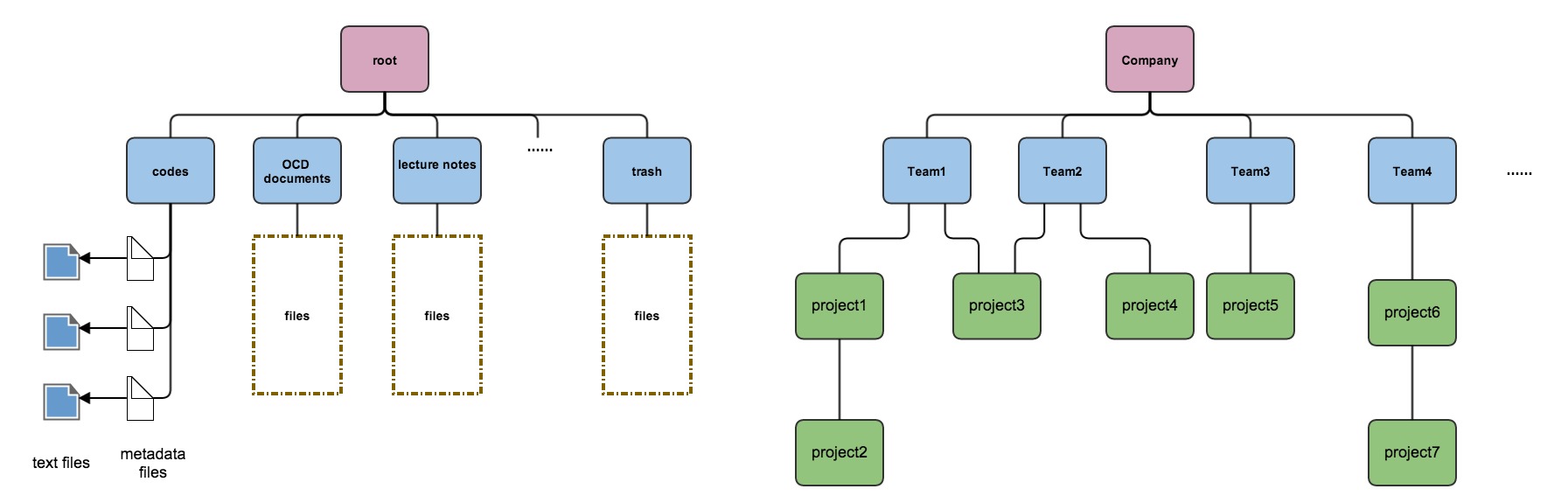


Figure 2.4 Team Structure

MessageHandler Server

MessageHandler Server is like Receiver and Sender in Repository, it plays a role to handle messages, creating messages and send them to other members in SDCS via communication service, or receiving messages from others and dispatcher messages to sub server to process.

CommService

CommService package shall provide communication channel between Client and Servers. It creates channels with http or tcp bindings for messages transfer.

## 2.4 Test server

The Test server provides a test harness supporting continuous test and integration. When user checks-in a new package or a modified new version of existing package into Repository client also posts a message with test configuration to the Test server. Test Harness then runs a sequence of tests against modified package and subsystem it belongs to.

The Test Harness Server should support topdown testing, which means, when check-in a package Test Harness runs top test driver for current project baseline test project recursively. If any fail, Test Harness start subtree of the current baseline and run unit test driver to test package with more details.

Everything of product value is extracted from Repository and tested on Test Harness. Test Harness generates certified test results and store them on Repository with a link to the component version. Figure 2.5 shows the structure of Test Harness Server.

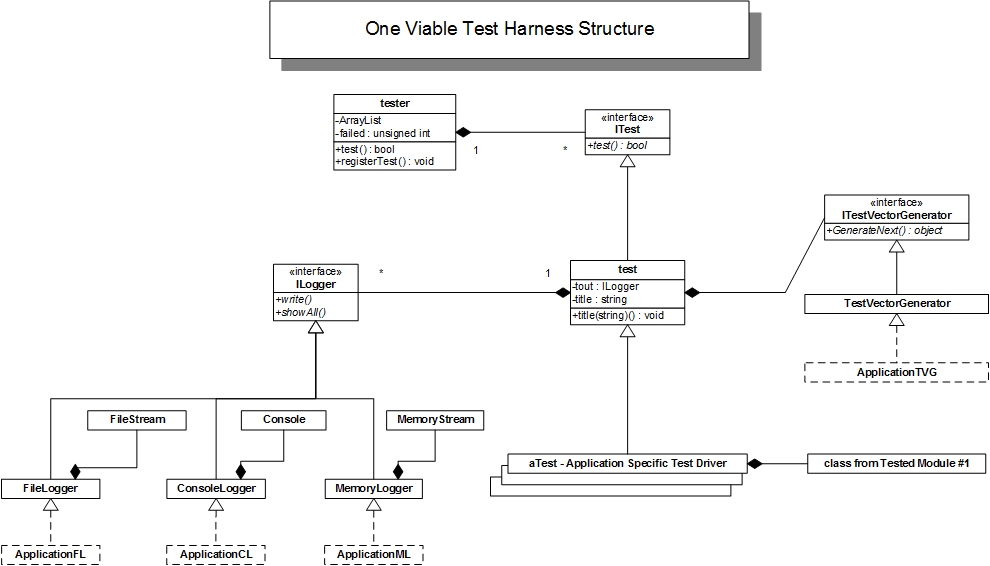


Figure 2.5 Test Harness Class Diagram

# 3. Code Repository

## 3.1 Interaction between Server and Client

This part will provide a concise introduction about the interaction between the Code Repository Server and Clients, including the Peer-to-Peer communication structure, brief package partition and task activity diagram.

#### 3.1.1 Peer-to-Peer structure

The Code Repository server and client is the major part we will discuss in this Operational Concept Document. The server and client communication service is a Peer-to-Peer asynchronous Message-Passing structure, it support central or distributed locater services. Peers interact, sending and receiving messages from each other. The common Peer-to-Peer structure diagrams are shown below.

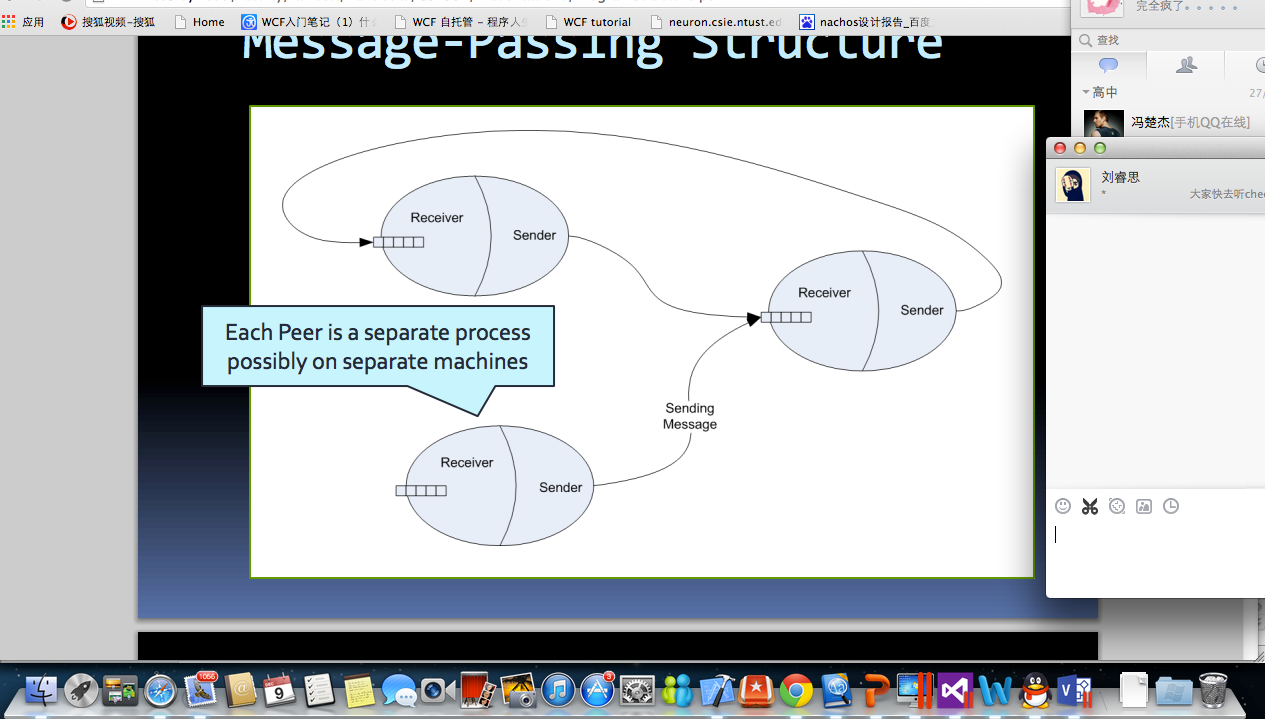


Figure 3.1 Peer-to-Peer structures

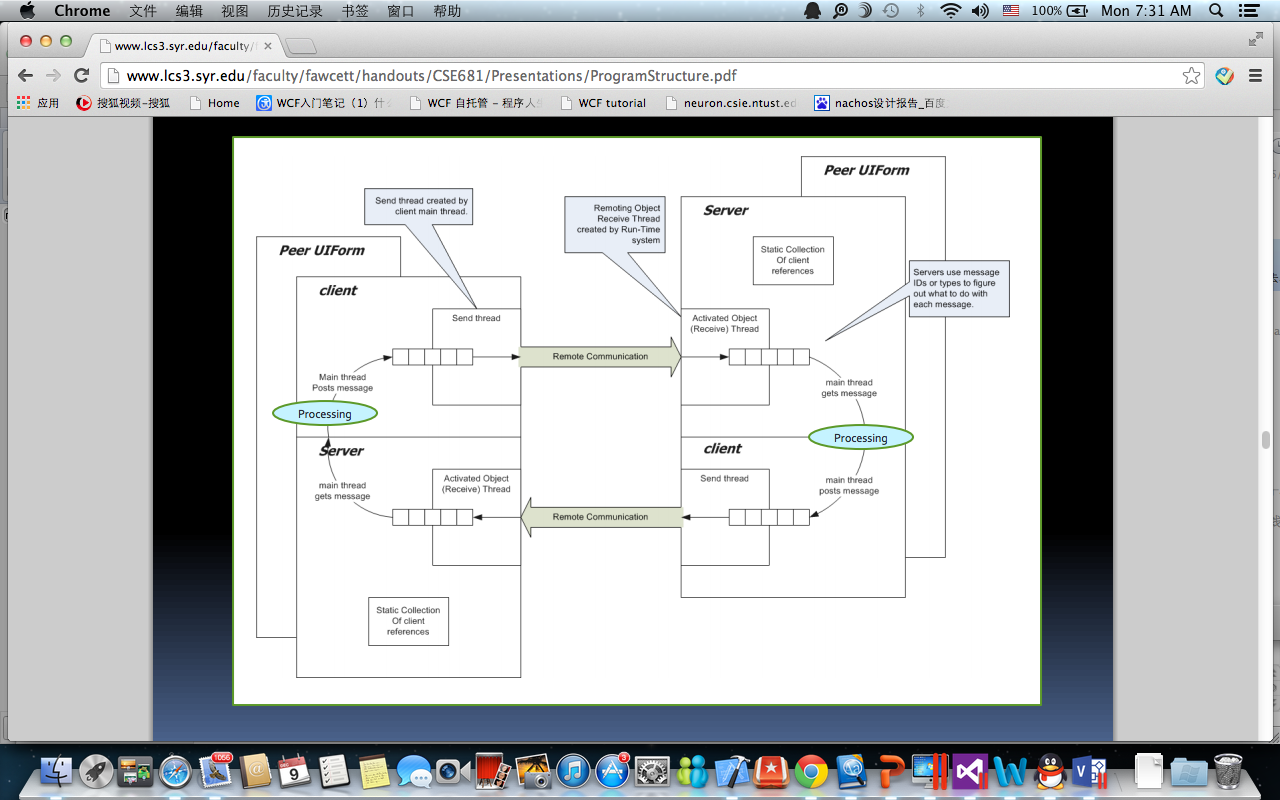


Figure 3.2 Peer-to-Peer structure (2)

The Processing procedure in client-side maybe View Handler or File Caching or other client task processing, while the processing in server-side maybe Query Handler, Navigation Handler and so on.

#### 3.1.2 Object Diagram

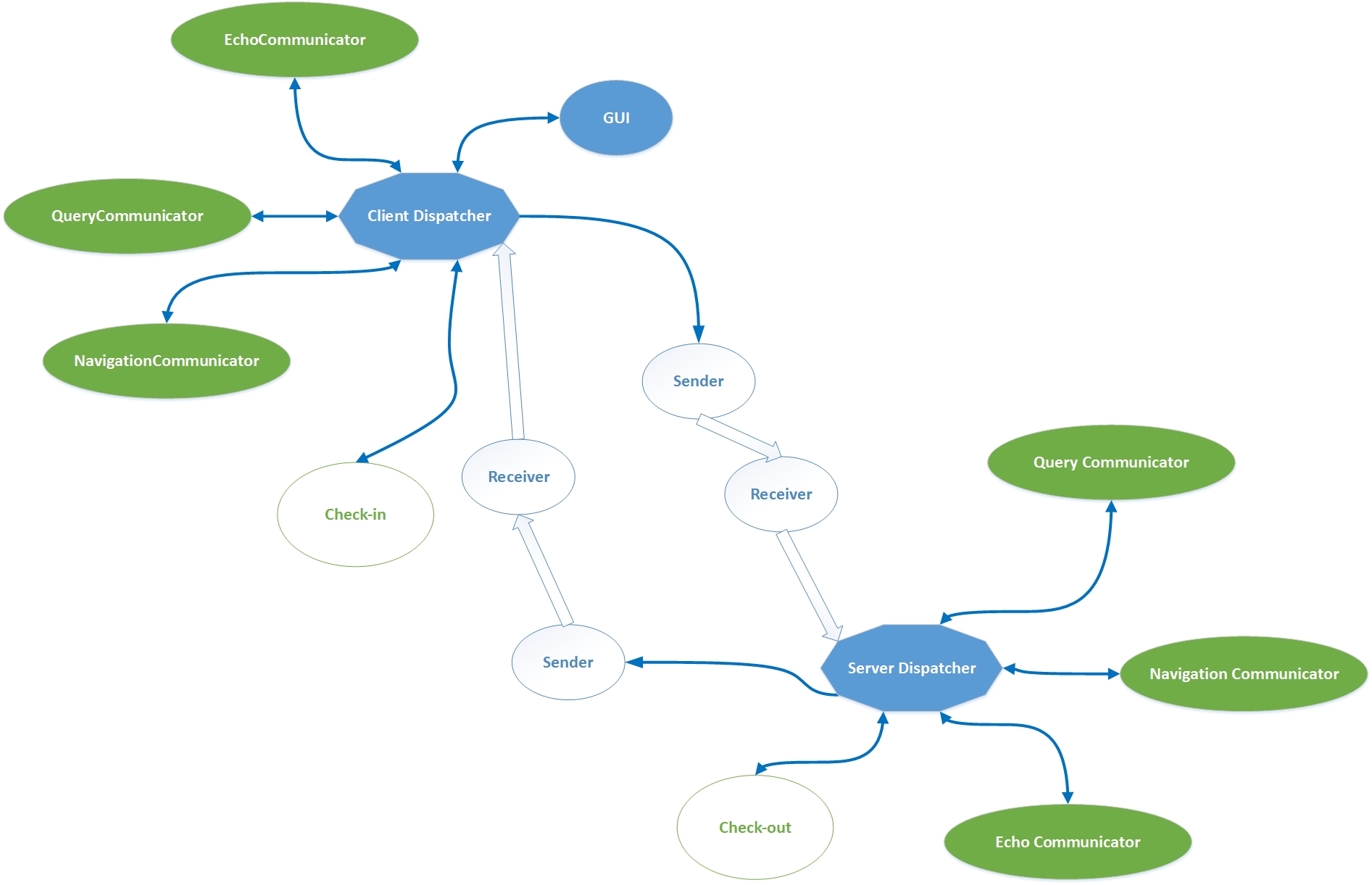


Figure 3.3 Object Diagram

This object diagram gives us a clear view how Repository Server and Client work to handle messages, the Client send messages by Sender, Server receive messages by Receiver, then distribute messages to different communicators and call functions to process messages. For example, the query request messages go to the Query Communicator and call queryProcessing function to process. Reply messages go to the similar way back to client-side.

#### 3.1.3 Activity Diagram between Server and Client

The Figure 3.4 shows the general activities in server-side and client-side. In next two sections we will talk the Repository Server and Client separately, in these sections we will also talk about these activities in detail.



Figure 3.4 Interaction Activity Diagram

#### 3.1.4 Structure Diagram between Server and Client

The united Package Diagram is shown below, this diagram gives us a concise general understanding about packages in server-side and client-side. These two parts are combined by Communication Service packages; each of them has this package to implement communication. We will talk about these packages responsibilities and interaction between them in next two sections Package parts.



Figure 3.5 United Package Diagram

## 3.2 User and uses case

This part describes the potential users who use the Remote Document Vault and uses case.

#### 3.2.1 Software Development Company

Software Development Company is an organization that is responsible for developing lots of software. Company may develop several software projects at the same time, and the company needs to build a platform to manage all development. This Code Repository Server and client system is that kind of platform for a company to process and manage its software projects. Company can distribute projects to several teams or individual developer to construct software. The codes for developing software sometimes may repeated, Software Company should have a codes library also.

Uses case: Assign permission to the teams and individuals who need to develop projects on this Code Repository.

The company administrator has the highest authority to access whole Repository content, but it should also assign the permission to the teams and developer who are responsible work on these projects’ baseline. It should distribute all projects to these teams and developers, like team A has access to project1 and project2; developer B has access to project3. Then the company only needs to monitor the progress of development.

Uses case: Manage the Repository content, like remove projects have been developed completely.

If a project in Repository has been developed completely and builds and tests successfully, the company needs remove this project from Repository to make room for new project need to be developed. Company shall extract all project data from server and delete these data in server.

Uses case: Build a reusable codes library for its developers.

The projects this company responsible for may use some codes repeated, software company can use this Repository to provide all reusable codes for its developers, which will make the d

evelopment more efficient.

#### 3.2.2 Software Architect

Software Architect is a [computer programmer](http://en.wikipedia.org/wiki/Programmer) who makes high-level design choices and dictates technical standards, including software coding standards, tools, and platforms. Software Architect is responsible for the design of software not programming, he/she will build the architecture of projects and lead teams to implement those, Software Architecture always work on several projects concurrently.

Uses case: build structure of projects and make plan of development.

Software Architect needs to build a structure of a project and store the script in the Project Server, and this structure script may be a metadata file. The Project Server will translate it into a map and post the map on Whiteboard informing developers which packages need to add into Repository. Developers will flow the Software Architect’s instructions and build the project. This structure metadata will also be a standard to check the development progress and analyze progress in Progress Server.

Uses case: Use defined dependencies and find files in their P/C relationships virtual graph.

Software Architect can focus on specified project material and files related to it by accessing to one file in this project. Software Architect can make queries into remote repository or browse in categories/projects view to focus on one file. The relationship and categorizing are implemented by metadata file information, and each metadata in repository must points to a text file.

#### 3.2.3 Developer Team Leader

Developer Team should work on one or more project together, it breaks down software into modules and its members implement each module separately. The team leader should check all modules of a project, and he/she needs to check all files checked-in to Repository and navigate to their dependencies.

Uses case: use file cache

The client provides user a file cache to store files and dependencies recently used. For a team leader, file cache is more useful because he/she needs to check-out lots of files in a project baseline and modify them. By using file cache, it avoid transferring files from server to client every time, the leader can do his/her work locally.

Uses case: update version and return version of a project.

Team Leader always has the highest authority of a project, so that if a member of team has checked-in a file successfully, team leader can see the notification remind him/her to update the project version, and the team leader can choose to update project linking to the newest modified package. Team leader can also return a project’s version if he/she found the newer version of project goes into a biased way. Team leader can use Code Repository to manage team’s file and inspect architecture of projects.

#### 3.2.4 Developer Team Member

Developer team members are developers working on the same software project. They may work in the same location or may not. If they work in same location, this Server-Client service makes it more convenient to do their work, else if they work in distance, this Repository will definitely useful for them. In addition, they shall do their works on the client, they can use the graphic user interface to do their work so that save the trouble to enter the command line. The client for them shall provide ways to check-in/check-out files and communicate with other members or their leaders on another machine.

Uses case: Sharing codes and integrating modules

Team member can check-in their codes to the Server and create associate metadata files. Metadata files aim to create relationships with each other and store basic information. Members can also check-out files from Repository server, members who work on different levels of modules should check the interfaces by scanning files in specified categories and relationships.

Uses case: use client to check-in and check-out files.

Team members can simply click “check-out” button to check-out files from file cache or Repository, and “check-in” button to check-in files into Repository and their file cache. For check-in step, client will create file’s metadata file automatically only need user to enter some key value. They can browse projects’ packages on their client and make query into Repository. They can also see files navigation relationship on client views.

Uses case: test locally.

Every time when team members check-in a package, it should be tested by Test harness. Members may want to test it locally instead in the remote server. So that members can download a server instance to their local desktop, then they can do the test locally. After they modified the server instance, the modified one will merge with the source server, everything developers checked-in to the server will be added to the Repository.

#### 3.2.5 QA

A software quality analyst is responsible for applying the principles and practices of [software quality assurance](http://en.wikipedia.org/wiki/Software_quality_assurance) throughout the [software development life cycle](http://en.wikipedia.org/wiki/Software_development_process).

Uses case: Processing batch

QA should analyze a large set of files for quality assessment. Batch query, batch check-out and batch check-in processes are required by QA. Files shall be transferred with chunking avoid of occupying too much resource in Server-Client communication. The Server-Client mechanism may also need a pre-empt rule to assure uses.

#### 3.2.6 Manager

This Document Vault can be used not only in software field but lots of fields need to share large set of files and build the relationships of files. Managers refer to administrators to get access to all files in Server repository and revise it relationships by editing metadata files. Managers can also making queries into file set in Server to find specified file to edit and download.

#### 3.2.7 Customer

Customers who deal with large sets of files may need this Repository. They can upload their local files into the Server in categories and structures to save local space and make manage file procedure more convenient.

#### 3.2.8 Server Administrator

Server administrator is always a character in Software Company. Server administrator needs manage Server-Client connection and check the repository size for storing file sets, administrator should add Clients and determine when to add more space for repository if the repository is crowded.

#### 3.2.9 Interns and students

For intern and students, they may need to log in Repository server to download files for study. In this case, these users only have authority to download files; they are not allowed to make any change in Repository.

## 3.3. Repository server

#### 3.3.1.Concept

Repository Server is a dependency-based storage server, which holds and manages developing projects’ baseline, a collection of source code, documentation, test drivers and test results.

Repository server shall provide access to versioned files, packages and all its dependencies. Note that a replication of Repository server with specified client only provides files this client user has access to. User can check-out files from Repository project’s baseline to modify and check-in files into project’s baseline.

Repository shall holds all versions in linked structure so that if the newer version is inappropriate found in future it is convenient back to the earlier version.

Repository server also has some analysis tool, like Query Handler and Navigation Handler, Query Handler can make query into its storage to search for text files and metadata file, and Navigation Handler can help build and update a virtual map of its storage structure.

Repository shall transfer messages between it and clients or other servers, Project Server and Test Harness Server. Clients will pass request messages to the Repository Server, like query request, show file request, and Repository should pass result reply messages back to clients.

It will also pass notification messages to the Project Server when need. Because this Repository Server section will mainly discuss the communication between Repository and clients, so the two situation of transfer notification from Repository to Project will have a brief discussion below.

One situation is that when user checked-in modified file, the file’s parents files should be updated with their owners’ permits. Notification passed to Project Server and post notice in Scheduling to inform their user they can update version.

Another situation is that when owner is a group, when one member starts open check-in, which means open the check-in process until all children dependencies of first checked-in file have been checked-in, other members may check-in same children dependencies with different contents in this process, the files should be merged if that. In this situation, Repository should pass a notification to the Project Server to post in Scheduling to let the members know their files have been merged and they can check and chat to modify the merged file.

In addition, users can post notification when users have some important information want to inform other users. Users who have authority can send messages containing notification content in specified format to Project Server, and then server will post that notification on Scheduling or Whiteboard.

All this responsibilities of Repository can be implement by its packages. Message transfer processing is executed by Sender, Receiver and communicators. Query processing is implemented by Query Handler package. The discussion next will give reader a clear describe about this.

#### 3.3.2 Activity

This parts breakdown all the Repository Server-side uses into activities and use activity diagram to describe what happens in every step.

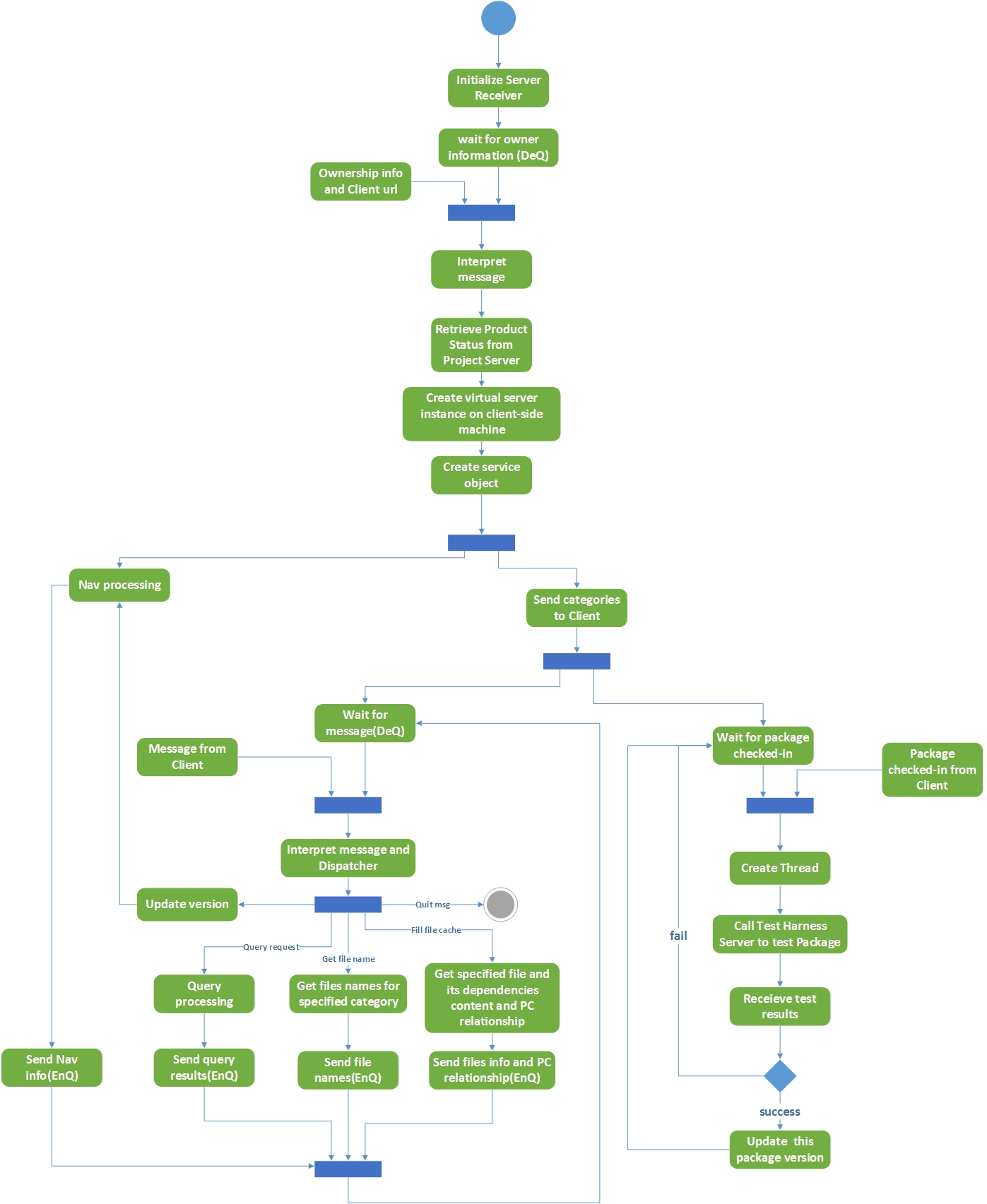
1. Activity Diagram

Figure 3.6 Activity Diagram of Server

2. Description of Activity Diagram

Initialization

Initializing Receiver instance when Repository Server starts, then server wait for messages from clients, which contain their ownership and client urls. After receive the information, the server retrieve product status from Project Server and create a virtual server instance for owner that only contains this owner’s accessible projects in Repository Server. Send reply message contains categories information in virtual server.

Message handling

Dequeue message and interpret it, then call dispatcher to process client’s request. After processing, create reply message and send it to the client-side communicator.

Query processing

When dequeued message requires for making query into Server repository, do the query processing according to users demands (will be discussed in Client-side), and store the results to create reply message.

Echo processing

When dequeued message requires for getting files in specified category or getting files information and its dependencies information, do the echo processing and create reply message contains results.

Navigation processing

When dequeued message requires for update version of project baseline in Repository or when start the Repository Server, do navigation processing map the PC relationship of newest version.

File handling

Receive the checked-in package and call Test Harness Server to test package and project baseline, and retrieve test results. If the test is successful, update checked-in package’s version, but do not update its parent packages until their owners require update.

#### 3.3.3 Structure

This part describes the Repository Server packages’ responsibilities and uses a package diagram to show the interaction between every package.

1. Responsibility of Packages

The Repository Server has 8 core packages:

* Server Executive
* Query Handler
* Navigation Handler
* File Handler
* Sender
* Receiver
* Dispatcher
* Communication Service

The responsibilities discussed below:

Server Executive

Server Executive is the controller of the Server-side. It determines the sequence of the packages in Server-side to active. It is also the start of the Server-side.

Query Handler

Query Handler package offers a service for users making queries into Repository Server. It contains metadata query and text query, and the results will be sent to client through query communicator by message. The text query processing can implement search one text string or several text strings in one or several categories, and the results are categories names and file names. The metadata processing can implement query for one or several tags in one or several categories, and the results are categories names, file names, tags and its values.

Navigation Handler

Navigation Handler aims to create/update map of parents/children relationships of all files in Repository by searching metadata files’ child dependencies list. The metadata for each file that has no parents shall identify its categories as parents.

File Handler

File Handler aims to check-in and check-out packages. It will receive packages checked-in from client-side, and call Test Harness Server to test packages and project it belong to, if the check-in process is successful, update this package (do not update its parents packages) to the new version. File Handler also send packages to the client-side when its require check-out.

Sender and Receiver

Sender provides functions to connect with client and put messages from Server into client-side receiver queue. Receiver, to the contrary, dequeue messages and call dispatcher to process.

Dispatcher

Dispatcher package provides service manage messages and communicators, distribute messages to different communicators and call functions to process requests.

CommService

CommService package shall provide communication channel between Client and Server. When a client request for a connection, Communication package should create a channel between this client and Server and create a service object in Server to handle this client’s requirement. The message and information send and receive shall pass by the channel. When finish the requirement and disconnect, the CommService should remove the service object and reassign the channel.

2. Package Diagram and Interactions

The package Diagram below gives a clear understanding of interactions between packages.

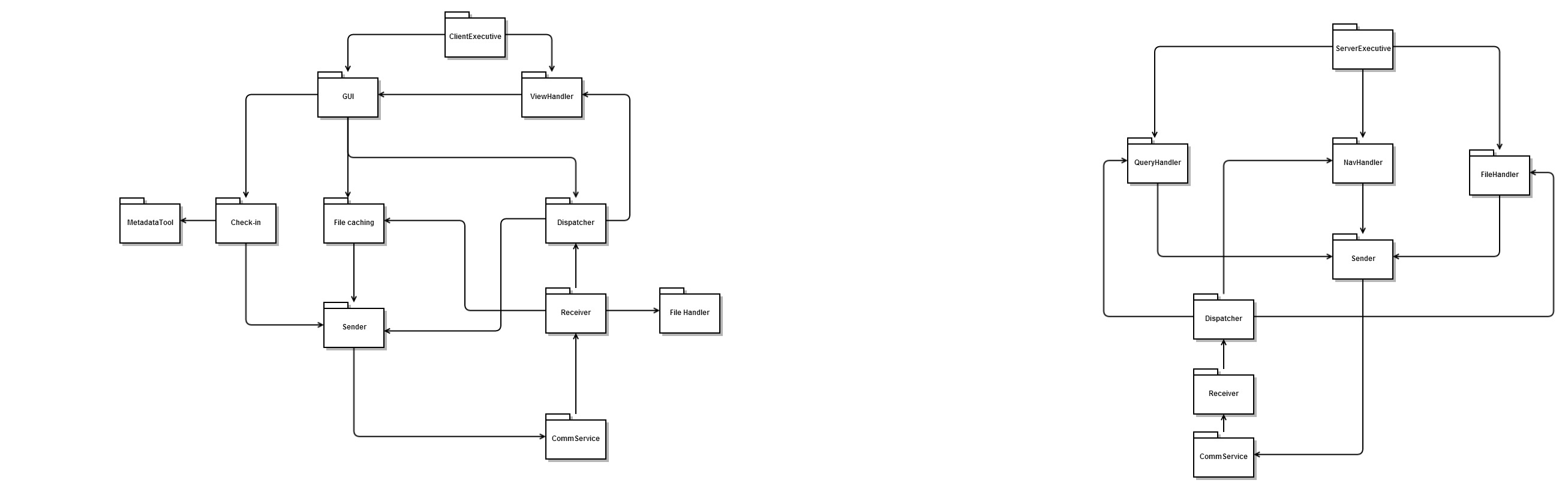


Figure 3.7 Package Diagram of Server

Interaction of Server-executive

Server Executive is the bridge and controller of the Repository Server, when the Client connection has been created, the demands will be sent to the Server Communicator, and the Executive shall decide invoke which package to implement users requirements. The replies will be sent to the Client after processing.

Interaction of Dispatcher

Dispatcher is used to assign tasks to different process, it interpret message from receiver and decide call which package to implement user’s request, call Query Handler to make query into Repository, call Navigation Handler to find PC relationship and call File Handler to check-out package and so on.

Interaction of CommService package

CommService package is the bridge between Client-side and Server-side. It will build a channel between them and create service object for clients. It will call Receiver package to receive messages from client-side or file checked-in and create messages then send by Sender.

#### 3.3.4 Interfaces

This part talks about the interface Server communication use, including message formats, interface contract.

1. Messages formats

Messages transferred between clients and servers should have information about target communicator, source communicator, message content, and resource name. There is an example, when user request to make query into Repository, client will send a message to Repository to implement that, and the message should be “queryCommunicator, ClientService, requestQuery, clientUrl”. These strings are DataMembers in DataContract. “queryCommunicator” is the name of target communicator – query communicator in Repository server, “ClientService” means that this message aims to implement client service, “requestQuery” gives server command to process query, and the clientUrl stands for this message pass form address clientUrl. clientUrl should be a http address.

2. Interface contract

Contracts are simply interface declarations, in this WCF communication model, it has two types contract: Service contract and data contract.

Service contract specifies what operations the service support. This communication service has several [OperationContract] attributes to implement its service, posting message to target communicator and transferring files in block.

Data contract describe what kind of data type can be transmitted from clients and servers. Data contract should have [DataMember] attributes to compose messages as we mentioned above.

#### 3.3.5 Relationships:

Figure 3.8 illustrates how the Repository Server manages dependency relationships between the resources it holds. Each source code file has an associate metadata file, and the children dependencies was embedded in the metadata file. The Repository’s Navigation Handler will build a virtual map, a dictionary container, which will store the metadata files’ parents relationship.

When user check-in a modified file, a new version of its metadata file is created and added to the repository contents with the modified file. If the test result of this modified file is successful, server will recursive find its parents metadata file and update the parents metadata file and parent source code file’s version eventually, this part will be discussed in version policy in detail.

When user asks to renew its file cache, server will open the aimed files metadata and find its children files recursively, then transfer all files to the client-side.

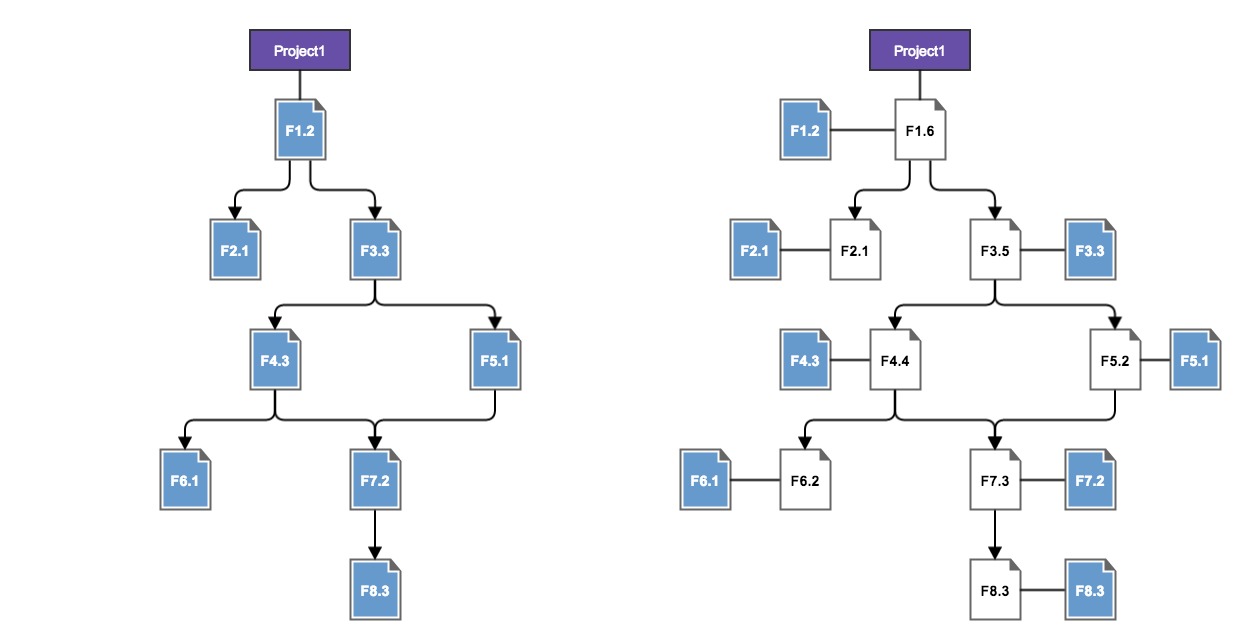


Figure 3.8 Virtual Navigation Graph

## 3.4. Clients

#### 3.4.1 Concept

Clients as a member of Software Development Collaboration System, it plays a role to provide local development platform. User can write source code files and test files local then check-in files into Server.

Clients also provide communication functionalities to help developers communicate. Users can chat with each other or discuss on Whiteboard, and see schedule plan displayed by Scheduling server in Project Server, users can also check Wiki application to find material they need and so on.

The main functionality of Clients is connecting with Repository Server and sending request messages to server. Clients have graphic user interface to help users do this. They can browse files in Repository projects’ baseline, check-in check-out files, make query into Repository and cache files and metadata locally to minimize network copies, all of these will be described in next parts.

These functionalities could be implement by packages discussed in partition part, and message passing could be implemented by communicators, Sender and Receiver, which is similar to server-side.

#### 3.4.2 Activity

This parts breakdown all the client-side uses into activities and use activity diagram to describe what happens in every step.

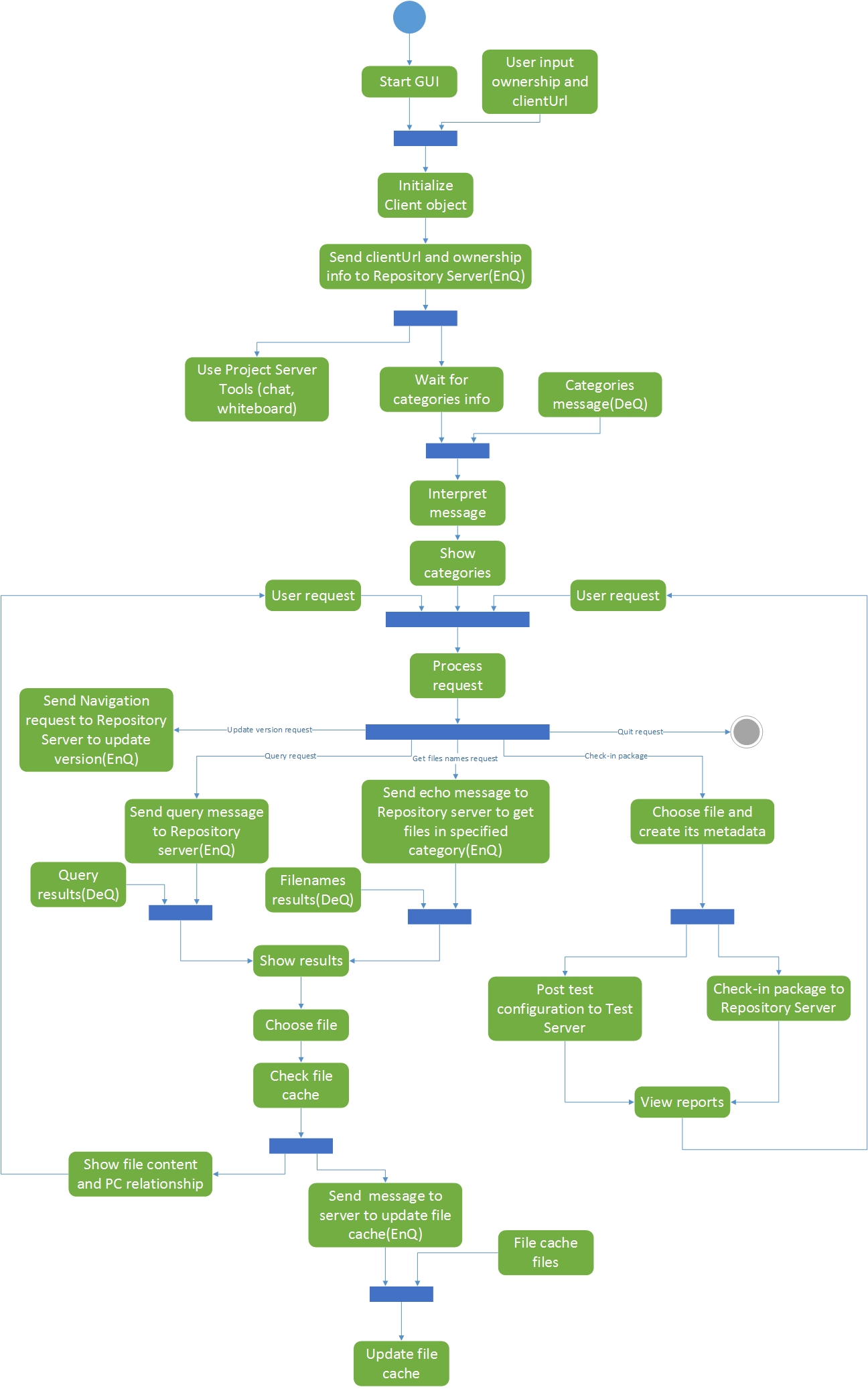
1. Activity Diagram

Figure 3.9 Activity Diagram of Client

2. Description of Activity Diagram

Initialization

Start GUI and get user’s ownership information and its client url, create message contains ownership information and url to Repository Server to request connection. Then get categories of files in repository user have access to.

Message handing

User can request Repository Server to process functions by creating request message, query request, echo request to get files in specified category, navigation request, and update file cache request. The server reply messages are received and dispatcher to change GUI views.

Check-in processing

User can choose file or files (with its dependencies) and create the metadata then check-into the Repository Server, in this procedure, the client also send message with test configuration to Test Harness Server to ask for test. Then user view see the reports of check-in.

File caching

Clients have local file cache, which can avoid unnecessary file transfers. When user selects a file whose dependencies are not stored in file cache, client will send message containing absent file name to server to update its file cache, and server will transfer file and all its dependencies to the local file cache.

Collaboration activities

Client can use collab tools provide by Project Server to chat with other clients, or use whiteboard application, wiki application, etc.

#### 3.4.2 Structure

This part describes the Clients-side packages’ responsibilities and uses a package diagram to show the interaction between every package.

1. Responsibilities of Packages

The client-side has 10 core packages:

* Client Executive
* GUI
* View Handler
* Check-in
* File Handler
* File Caching
* Sender
* Receiver
* Dispatcher
* Communication Service

Client Executive

Client Executive is the controller of the Client-side. It determines the sequence of the packages in Client-side to active. It is also the start of the Client-side.

GUI

GUI provides views to display information and achieve user’s requirements.

* It provides a view to show categories and files information passed from Server. The categories and files are clickable make itself the current object and open another view.
* It provides a view to show XML and text file content stored in the file cache.
* It provides a view to show P/C relationship of current file stored in the file cache. The parent and children files are clickable, and click it make a file to be current file.
* It provides a view to check-out/check-in package and modify it. User can browse local files and choose package to check-in. The check-in package must contain metadata, and user can create it in this view. User can also edit metadata in Repository Server and check-in it in this view.
* It provides a view to make queries into Repository Server, and it can show results in a clickable file list for users changing current file.

View Handler

View Handler is used to invoke GUI views. View Handler obtains user’s requests or messages received from Server-side to invoke different views in GUI.

Check-in

Check-in package implement check-in service, user can check-in package into Repository’s project baseline. User shall check-in package with its metadata file which has its description information and dependencies. Check-in process will call Metadata Tool to create XML file and make sure each source code has its metadata file.

File Handler

File Handler to process checked-out packages, the package checked-out will be received by Receiver, and Receiver invoke File Handler to store this package in local. User can modify this package and check-in it.

File Caching

File caching is used to manage file cache. File cache is a local storage to store files and metadata avoiding unnecessary file transfers. When user wants to download a component to see its contents, which depends on other files, it request the Repository to send a list of files on the component’s dependency map graph. Client will download all files in the list that it doesn’t have currently in file cache and add them to the cache.

Note that when user modifies a file and checks-in a new version into Repository successfully, the new file is also added to the cache.

File caching implement show file contents locally, when user change current file, client can find file and its dependencies in local and show its contents to the panels.

Sender and Receiver

Sender provides functions to connect with Server CommService package and put messages from Client into Server-side receiver queue. Receiver, on the contrary, dequeue messages and call dispatcher to process.

Dispatcher

Dispatcher package provides service manage messages and communicators, distribute messages to different communicators and call functions to process requests.

CommService

CommService package in Client-side has the similar functionality in Servfer-side, which is providing communication channel between Client and Server. The CommService shall generate messages in different communicators for different requests; send message by Sender to the Server-side, and process reply message in different communicators.

2. Package Diagram and Interactions

The package Diagram below gives a clear understanding of interactions between packages.

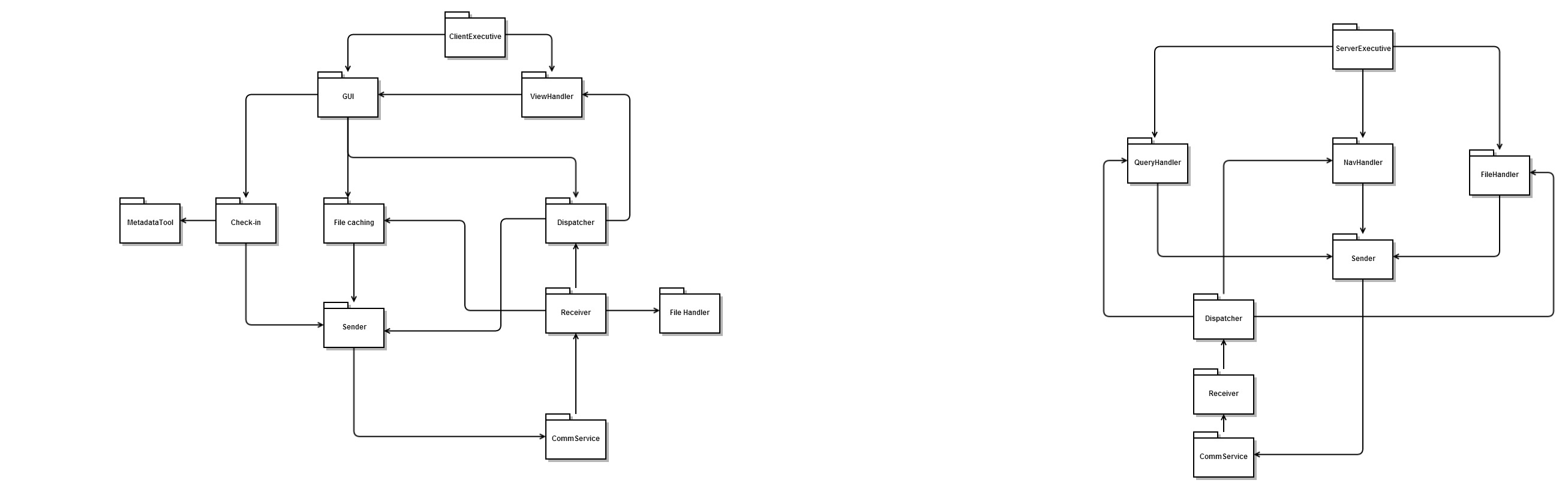


Figure 3.10 Package Diagram of Client

Interaction of Client Executive

Client Executive package is the bridge of Client-side packages, when Client requests for connection with Server, the CommService package shall create a channel between them and an object to handle this client’s requirement. CommService package will pass the messages and information from Server to Client.

The Client Executive package shall call the function in GUI and View Handler to create and change views. GUI package will feedback users input and requirement to the Client Executive, Client Executive then call functions in View Handler to change view or Dispatcher to prepare messages.

Interaction of Dispatcher package

Dispatcher package here has similar responsibility with Repository’s Dispatcher. The GUI obtains user’s requirements and call Dispatcher to change views or send messages. The Receiver get replied messages from Server and call Dispatcher to process different tasks.

Interaction of Check-in package

Check-in package will be called when user require check-in file in GUI check-in view. It will call Metadata Tool to create XML file and call Sender to send the source code and its metadata to Repository.

Interaction of CommService package

The CommService interaction in Client-side is similar to the Repository Server-side. It call Receiver package to dequeue reply message from Repository Server or receive checked-out files, and it call Sender to send messages created or checked-in files.

#### 3.4.3 Views

This part gives us a clear view of graphic user interface views providing to users.

1. Categories (projects) and files view

When user starts the Client GUI, user should register and connect with Server. Then the categories (project) in Repository user has access to will show in the Categories (projects) panel. Choose a category/project then all files in this category/project user has access to will show in Files panel.

Categories (project) and files view is shown by Figure 3.11.

2. File Information and File Navigation view

By clicking a file in Files panel, user will get file information and file navigation. Information tab displays the file contents of selected file; source code file and metadata file will show in two panels. By clicking edit button users can enter the check-in view to edit metadata file in metadata editor.

Navigation tab shows the Parents/Children relationship of selected file. One panel for its parents and another panel for children, the files in navigation tab are clickable, so users can change the current file by click a file.

The “Edit Metadata” button should be changed to “check-out” and it should implement to check-out this file and its metadata and modified locally.

File Information tab is shown by Figure 3.11 and File Navigation tab is shown by Figure 3.12.

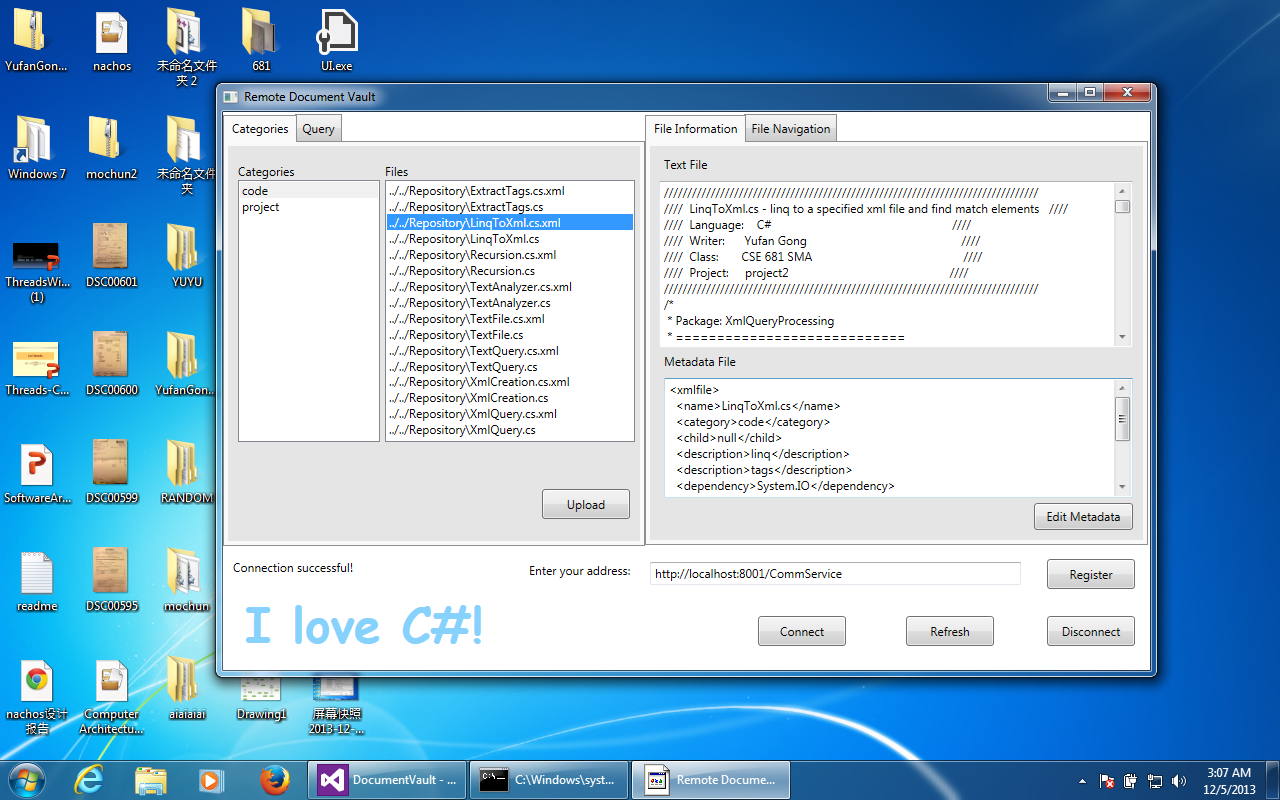


Figure 3.11 View(1)

3. Query Process view

Query tab offers user functionality to make queries into Repository.

Input the text strings and select text file will make query into source code in Repository, the check box all strings give users choice to query files contain all strings input or at least one string input. Users can input tags and select metadata file also, which will make query into metadata files contain all tags input.

The categories text box render user chooses categories making queries into.

Result for source code query will show the files names and its categories, result for metadata query will show the tag names and its value, files names and categories.

Figure 3.12 and Figure 3.13 show Metadata query and source code query.

By clicking filename in result, the File Information and File Navigation view will change to the selected file, which means change the file to current file.

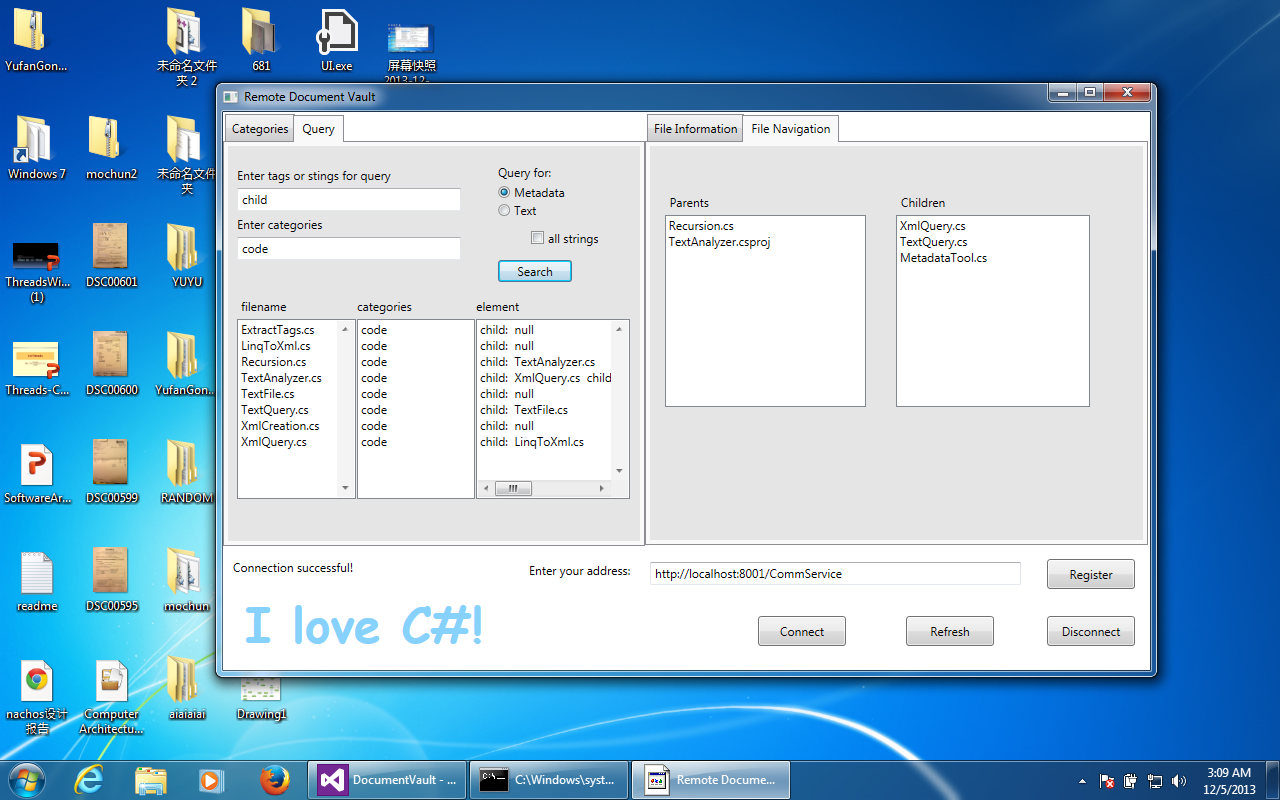
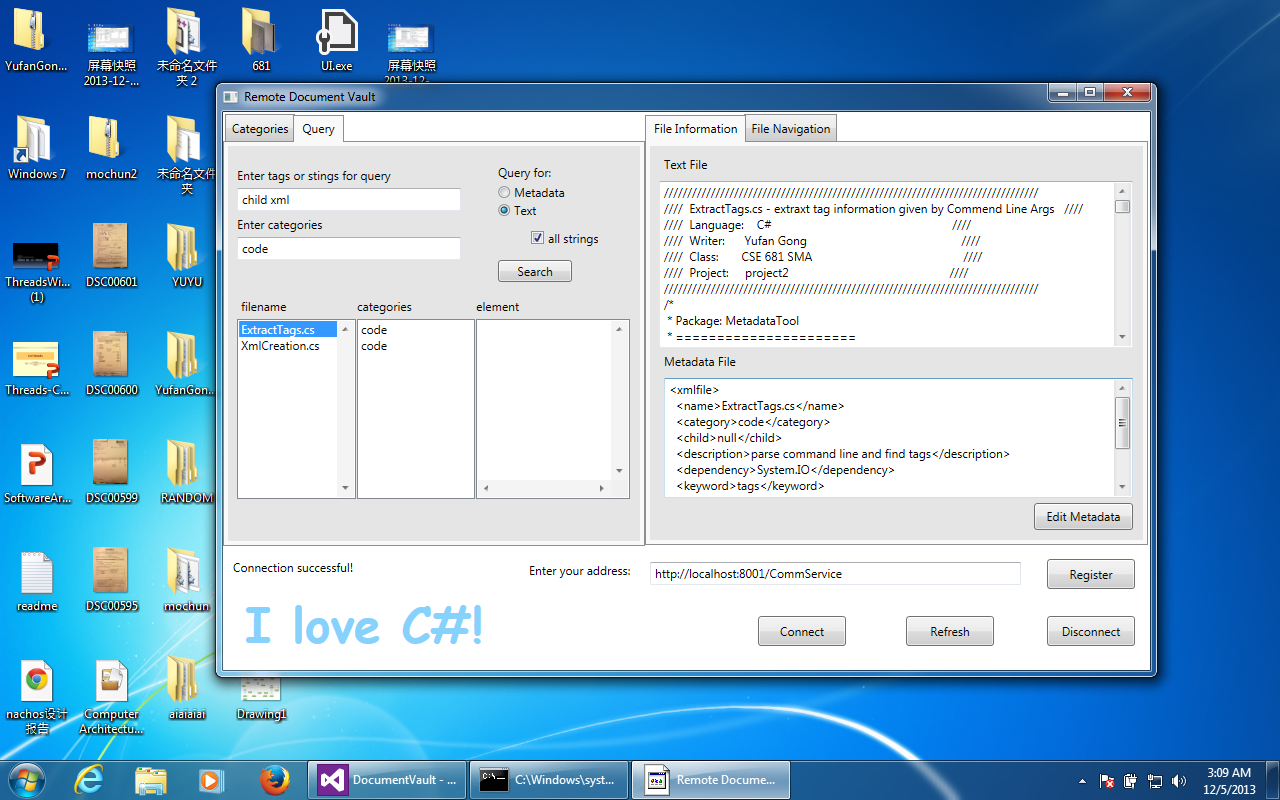


Figure 3.13 View(3)

Figure 3.12 View(2)

4. Check-in view

By clicking upload file button in start view or Edit Metadata button in File Information tab will enter the Check-in view.

Users can choose a file by browsing in local file system and then edit its metadata file below. If users choose edit metadata file from Information tabs, users can directly edit metadata in metadata editor. The tags names already given, users need to fill in tags’ values. For category tag, user must select categories existing in Repository except administrator. For children dependency tag, user can select files already existing or enter new files names.

User must check-in source code file with its metadata, which will be discussed in check-in policy.

Check-in view is shown by Figure 3.14.

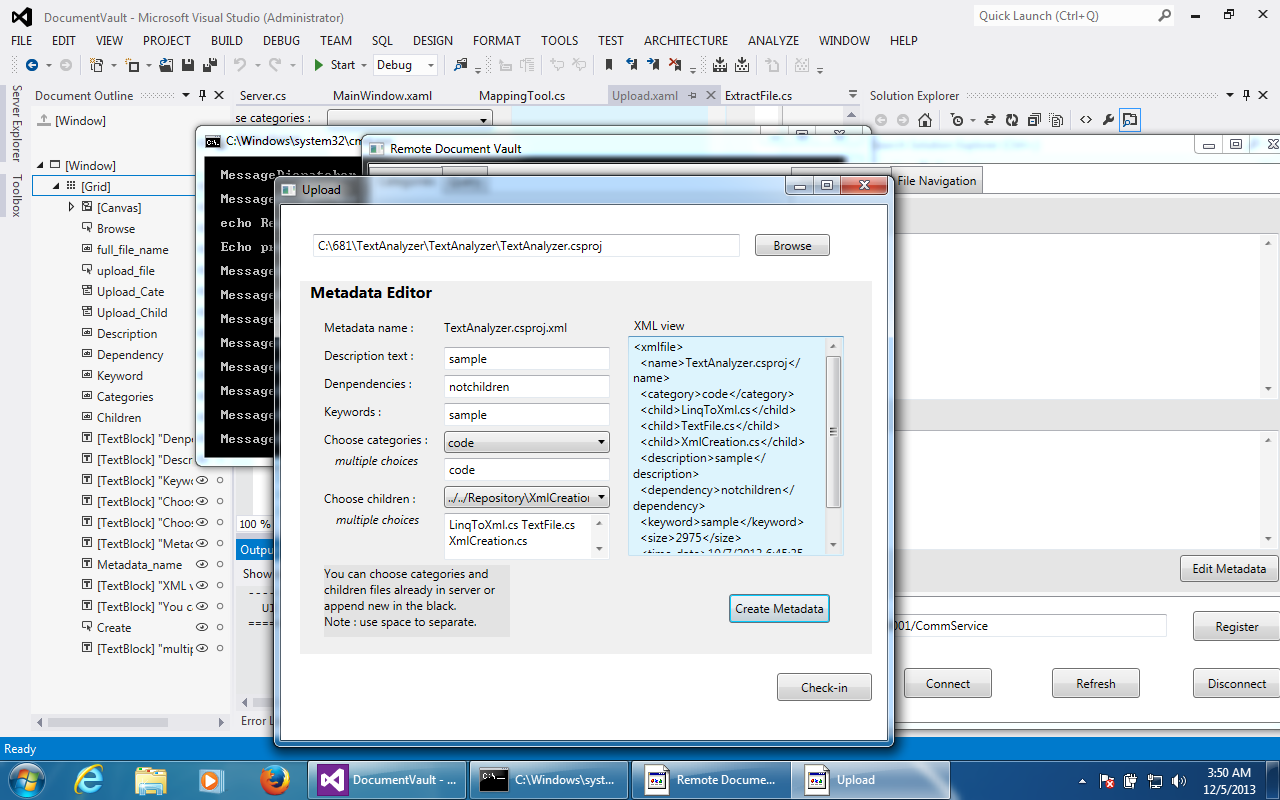


Figure 3.14 View(4)

## 3.5 Critical issues

This part discusses critical issues that may have influence on the Code Repository. Ignoring these critical issues may cause some dangerous situation and make the Clients insertion, extraction, query and navigation process failed. Some of them have been listed below and the solutions of them have been discussed.

#### 3.5.1 Files name conflicts

There are two files name conflicts situation:

Files name conflict when start open check-in.

When start an open check-in procedure, the single user or users in this group owner could keep checking-in packages until all dependencies have been checked-in and close the open check-in. In this situation, users may check-in same file with different content.

Solution: During an open check-in process, if files with same name have been checked-in, the Repository will extract these files and merge them automatically or manually. By posting a notification to Project Server Scheduling to notice all users in these group that the files have same name should be merged, or tracking access to find the users who have checked-in the files then send them messages shown in their clients to notice them they need to merge their files on Server.

Files name conflict with Repository Server system files

Another conflict is that when a user check-in a file that has the same name with Repository Server’s system file, which is in the Repository package to implement functionalities of Repository. The file checked-in with same name may override the system file cause the Repository cannot work normally.

Solution: The Repository Server needs to build a map to store its own system core files, these files need to be protected strictly, which means no user could has access to these core files except server administrators. In addition, whenever users check-in files, Repository Server will compare the file name with files in the core system packages, if they have same name, the check-in procedure will be denied except the users are administrators.

#### 3.5.2 File caching issues

File caching incoherencies

The file caching cannot always keep up with the versions in Repository simultaneous. All clients of this virtual server are adding files to Server and updating its versions, so that local file cache maybe out of date easily.

Solution: The client could request to server to send all metadata interest to client that have been added since the last 24 hours ago. When the coherency is important to the client, client could request server to send all metadata since last request, which is an expensive but effective way to keep the coherency.

Run over the storage of file cache

Another issue about file cache is how to manage the cache contents. The file cache is much smaller than Repository Server, so that we cannot keep caching files into it, it will run over the storage and cause some errors.

Solution: The standard way of dealing with this is to use a “least recently used” algorithm to discard files when the cache size grows beyond some critical values.

#### 3.5.3 Huge file sets transfer

When the client user require for uploading or downloading a huge set of files, the WCF can not connect Server with Client long-playing, which will impede other clients to connect with server.

Solution: The WCF communication package should divide file set into blocks and store request. Which means the file set will transfer with chunking automatically without re-request. This blocking method can fairly and efficiently allocate resource for each client service object.

The service objects for clients shall be put into garbage when clients finish their calls and ask for disconnection. It will refresh the thread resource in server.

#### 3.5.4 Security issues

The file transfer process now doesn’t have an encryption transmission mechanism. If the developer are developing some confidentiality software or the company use this platform need to prevent their competitors to filching their production, the normally file transfer is not save.

Solution: Using encryption transmission to transfer files. Files will be transferred by stream between clients and server, so that it is not hard to add an encryption package link to Sender, on the other side, it must add another decryption package link to Server. All files transfer between client and server is encrypted so that if any other individual or organization without authority obtains these files in transfer procedure, files will be encrypted, without decryption, it can not be read and write.

# 4. Services and policies

## 4.1 Services

The Code Repository can provide users many services, such as check-in and check-out packages, using file cache to avoid unnecessary file transfer, update package’s version in Repository and so on, we will have a discuss in this parts about these services.

#### 4.1.1 Check-in/check-out service

The Code Repository server and client system provides users service to check-out packages from Repository and check-in packages into Repository.

For check-out operation, user and extract a package from Repository, then modify it locally or check-in a new one replace older one. User can choose a file in file list, click check-out button to download it. In older to maintains the integrity of project baseline and the progress of project’s development, user can not check-out one more package before they check-in a new version of the package he/she checked-out. We will have a detailed discussion in policy part about this.

For check-in operation, user can check-in a new package or modified package in check-in view. The package checked-in will be added to the project’s baseline and tested by Test Harness Server. The test process will be processed topdown, checking is this package work successfully in project’s baseline, if any file, test will be processed in subtree of project to find the problem of this package. The test results will be post on Repository and Client. User can see the test results when the package be checked-in, if success, Repository will post a notification to Scheduling Server to notice upper owner they can update Repository now.

#### 4.1.2 Metadata

When user check-in a package into Repository Server, there is a metadata tool in client-side to create a metadata file and check-in this metadata automatically with its source code file. User need to input the project name its belongs to and its children dependencies, then click “Create metadata” button to create it locally, without creating a metadata file, user can not check-in a package.

User can also edit a metadata file stored in Repository, in this case, the file name and projects value won’t change, user can modify other values like children dependencies and update this metadata into Repository. Editing metadata file won’t trigger the test process.

When the checked-in package test result is successful, the upper owner can update the Repository relationship, which means, render the former package’s parents package depend on the new version. This procedure processing automatically by searching the newest children dependencies in Repository and create a new metadata file associate with this parent package and add this new metadata into Repository. The newer metadata file will link to the older one in Repository.

#### 4.1.3 Notification

When user checked-in modified file, the file’s parents files should be updated with their owners’ permits. So that notification passed to Project Server and post notice in Scheduling to inform their user they can update version.

When owner is a group, and one member starts open check-in, which means open the check-in process until all children dependencies of first checked-in file have been checked-in, other members may check-in same children dependencies with different contents in this process, the files should be merged if that. In this situation, Repository should pass a notification to the Project Server to post in Scheduling to let the members know their files have been merged and they can check and chat to modify the merged file.

Also, users can post notification when users have some important information want to inform other users. Users who have authority can send messages containing notification content in specified format to Project Server, and then server will post that notification on Scheduling or Whiteboard.

#### 4.1.4 File caching

File caching is a service supplied in client-side, client implement caching by storing locally file it has received from Repository at earlier time.

When user chooses a file make it the current file, client will search in file cache. If the file already exists in file cache, it will check is that all dependencies of this file existing in file cache and send a message to the Repository including all dependencies name. Repository then will make a list of all files and its dependencies recursive then download these files to the client file cache. If the selected file doesn’t exist in file cache, client will send this file name to the Repository, and the Repository will traverse the virtual dependency graph and make a list of all dependencies, then download them to client’s file cache.

When user check-in a package into Repository, this package will also be added in its file cache. This file caching service allow client to navigate through local metadata, when the current file cache can not satisfy the user’s require, which means a reference found in one metadata doesn’t exist in file cache, file cache makes a request to Repository to download this component and all its dependencies. This would save a lot of network traffic and server load and make client’s navigation view more responsive. The file cache would be update to the newest version at some time, midnight for example, as we mentioned in critical issues above. Also, it would remove some files and metadata not be used recently.

#### 4.1.5 Update version

Repository needs an update version service to renew its storage, to update its project’s base line for development. When user checks-in a modified file successfully, it become a new version of older file in Repository, its metadata links to dependencies (may change dependencies also), but the older package’s parents don’t have a link to it currently. After the owner of its parent request update version, parent would point to the newer one instead of former. This update procedure is a bubble up process that Repository will keep updating until the top one updated. Every time it updated one file version, Repository will send a notification to the Scheduling Server in Project Server to inform upper owner he can update his files version. Note that when update a file’s metadata, the newer metadata will have a link points to older one, which is convenient for undo and find the older version.

This down top process is shown in Figure 4.1.

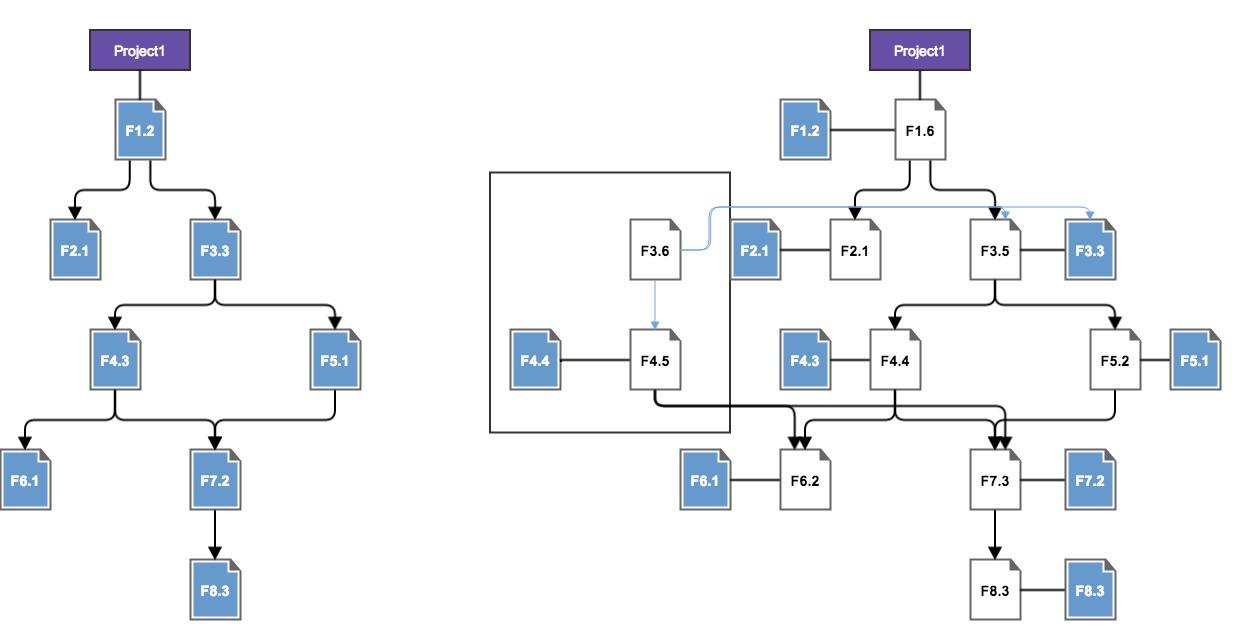


Figure 4.1 Update Versioning

#### 4.1.6 Back version

When users realize that the newest version go to an inappropriate way against the aim this project has. The Repository project baseline can easily back to the former version.

If the whole project baseline has been updated, only the top user of this project can choose to back to former version. If the project baseline have updated to some level, only that level’s user can choose to back to former version. Other user who realizes that project goes to a biased way can send a notification to Project server to notice upper user he should choose to back version.

#### 4.1.7 Track access

Track access is a service provided to guarantee the security of Repository project’s data. This service not trying to control access, just track access, so that we can know which component has been modified by which user.

Track access service needs at least two hash tables and two metadata files to store the access information. One hash table would use the components of project baseline as key, userids who have accessed the component as value. Another hash table, on the contrary, use the userids as key and a list of components this user accessed as value. So that we can know who have accessed a component and which component a user has accessed. We need to add function make those hash tables would not store information literately, which means only log the recent access.

We also need two Xml files to store information in hash tales stable and persistent. So that when Repository shuts down, it will write its hash tables contents to Xml files. When it restarts, it will retrieve the contents from Xml to its hash tables, this process need to create a new thread to guarantee the Repository starts quickly.

#### 4.1.8 Query

The Code Repository would provide user a query service, which means, user can search files and metadata files in Repository. User first clicks query tab enter query view, if he/she wants to make query into text files, he/she should enter one or more strings in the query condition text box, and he/she should choose a category or project to make query into, then checks query text files, he/she can also choose to query files contains all strings entered or at least one string by choosing a radio button. The query results will be sent to client, which shows files categories (projects) it belongs to and file name.

If user wants to make query into metadata files, he/she needs to enter one or more tags, choose a category or project to make query into and check the metadata query. The query results came back will contain the file name, categories or projects it belongs to and the query tag name with its values.

The query processing will process recursively if user chooses a category that has a substructure, like several projects. The file names in query results would be added into a clickable list, user can selected a file to show it content and metadata content, this procedure also comes to the file caching as we mentioned above.

#### 4.1.9 Navigation

Navigation service provided by Repository is to draw a virtual graphic map of metadata. This map gives us a clear view about the relationships in Repository (Figure 3.8). Children dependencies have been embedded into metadata, and parents dependencies would be stored in a container like map, use child file name as key, current file name as value, so the current file is its children dependencies’ parent.

The Navigation service also gives user a service to find current file’s children and parents. When user selects a file as current file, its children and parents file will be added into Navigation panel. The Repository traverse the virtual relationship graph and find its dependencies send back to client. This functionality also implemented in file caching locally.

#### 4.1.10 Communication

The SDCS provide user many ways to communication, not only communicate between clients and Repository (pass messages and transfer file), user can also communicate with each other by using collaboration tools.

Users can see notification posted on Scheduling to point what next step is of developing current project baseline, and is there any update desired by Repository. These notifications in Scheduling server give user a scheme to work on their development. User can also chat with each other to discuss the development, check Wiki web application to get knowledge and information they need, see Whiteboard to communicate with other user in a non-real time way and so on.

These communications implemented by using communication services, each member (Clients, Project Server, Test Harness Server, Repository Server) has interfaces to send and receive messages.

## 4.2 Policies

#### 4.2.1 Ownership policy

Ownership describes who has access to project’s baseline stored in Repository. When a user request to connect with server, server will create a virtual server only include this user’s accessible project’s data. And this user can only check-in/ check-out from these accessible file. Ownership is a rule-based policy so that ownership must be an element in every file’s associate metadata.

Ownership may be a hierarchical distribution. For example, a software architect of a company may have an ownership of all project’s data in Repository, he/she can see all source codes, documentation and test results in Repository and check the progress of project’s development. Company’s projects may be distributed to several teams to develop, each team leader has an ownership that accessible to all projects’ data this team responsible for, while the team members may only have access to part of these projects, which they are responsible to develop.

Ownership can be single owner, group owner and no owner, sensible policies discussed below.

Single owner:

Single owner means each item in this owner’s accessible file set only has one user. Only one user can check-in the modified files into its baseline. Since only one user are responsible for all changes to the items, no check-out is required.

Group owner with lock sub-policy:

Any member of the owning group can check-out, modify, and check-in an item owned by the group. If this member checks-out an item intend to modify it, this item shall be locked and other member cannot check-out or check-in this item until this member checks-in this item.

If one member of this group owner start an open check-in process, other members can also check-in dependencies of this item until all dependencies checked-in. The open check-in policy will be discussed in detail next.

No owner with lock sub-policy:

This no owner situation equivalents to group ownership when there is only one group encompassing all members of the Repository. Any authenticated member of this Repository may check-out, modify and check-in items. In this situation, policies are same as group ownership, user can check-out an item then this item will be locked until this user check-in the modified file.

#### 4.2.2 Check-in policy

When user choose to check-in an item, all dependencies of this item must exist in Repository, if user wants to check-in an item whose dependencies is not all existing, user will start an open check-in process.

When start an open check-in, users can continue check-in its children dependencies until all children have been added into the project baseline. It will create a notification and send it to the Scheduling Server notice other members and users have access to this project that they can only check-in dependencies of the open checked-in file. When start this open check-in process, other users cannot check-in package except these children dependencies. If other users check-in other packages into project, the test results must be incorrect because the project baseline is not integrated. So that in this procedure users can only check-in dependencies and these dependencies will not be test until close the open check-in.

Once all dependencies have been checked-in to the project, Repository will close the open check-in automatically and send a notification to Scheduling Server inform users that open check-in is closed. Then it will call Test Harness Server to test project check are the file and its children dependencies work successfully, then post result to clients.

When open check-in start, and an user want to check-in a file, server will compare the name of file with children dependencies waiting for checking-in. if the file is a children’s name, this file will be checked-in to Repository, otherwise, this check-in request will be refused.

#### 4.2.3 Versioning policy

The version of a package is stored in its metadata same as ownership. It is a rule-based policy. The version element generate automatically created by server. It will traverse the virtual graphic and find the latest version, the new checked-in file will have a larger version number than that one.

Bubble up update:

The versioning policy is that version cannot be updated automatically by server when user checked-in a package successfully, which means, the version of this package will be updated, but its parents and all upper packages will not unless server get the commands from their owners to update these package. And these owners won’t get notification that they can update packages’ version unless the children packages of these packages have been updated.

#### 4.2.4 Notification policy

The notification policy means to stipulate what kind of notification can be post by whom or what events.

When the Repository has received a modified file and test successfully, this file is a new version of the older one. At this time, Repository will post a notification to Scheduling Server to inform upper user who own that new packages parents package to update version.

When during an open check-in process, more than one file with same name have been checked-in, Server will send notification to Scheduling server, or by tracking access of those files to find the developer who is responsible for them and send notification to their client-side to let these developer manual merge files.

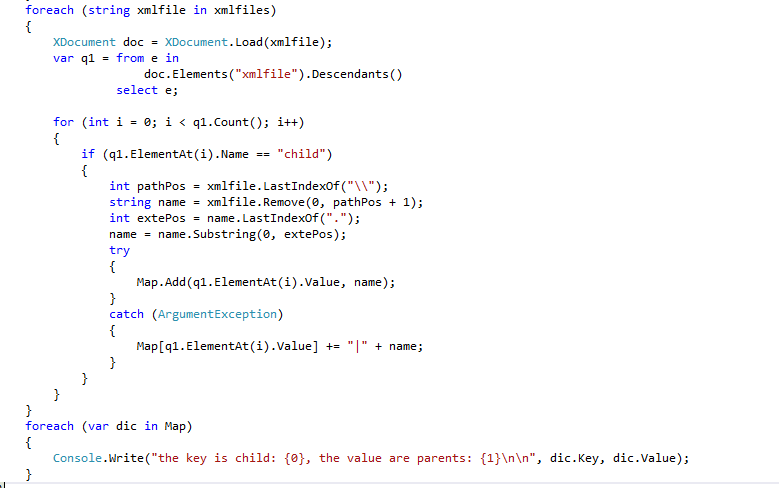
When the Team leaders want to inform team members some important information, like change the development plan, the team leader has right to send a notification to the Scheduling server and provide all team members information.

# 5. Appendices:

## Prototype

This prototype shows the mapping tool, the way to build the virtual navigation graph.

## 



# 6. Conclusion

The Source Code Repository can be used to transfer messages and files. It also has a Navigation tool to build relationships of files in it. The Repository servers and clients implement users share data and manage files more conveniently and efficiently.

In this Operational Concept Document, we have discussed the users who are potential to use the SCR, functions it can implement. We have also divided the SCR client and server’s architecture into several packages to show the interactions between them, in addition, we have illustrate the communications between Client-side and Server-side. By demonstrate its activity diagram, we make the procedure of its running clear. We have also discussed some critical issues may occur when the SCR is connecting, and the solutions provided can be used to handle some of critical issues.