CSE 681-Software Modeling and Analysis

Project #5

QUALITY ASSURANCE TOOLSET

OPERATIONAL CONCEPT DOCUMENT (OCD)

VERSION: 1.0

Arunkumar Manikantan (899234415)

Date: 05-10-2010

Instructor-Dr. Jim Fawcett

Table of Contents

1. Executive Summary

Quality assurance, or QA for short, consists of means of monitoring and evaluating the various aspects of a software project to ensure that standards of quality are being met.

Quality Assurance encompasses the entire software development process, which includes processes such as requirements definition, software design, coding, source code control, code reviews, change management, configuration management, testing and product integration. In order to analyze the quality of a project's evolving baseline, a toolset is required to inspect the various aspects of it. This document explains the architecture of a Quality Assurance Toolset consisting of the following tools.

- Risk Analyzer
- Rule-based Defect analyzer
- Tracking tools – requirements database, bug tracker, change logger
- Pretty Printer
- Server-based tool holster
- Agents to automate QA analyses
- Configuration tool for configuring inputs
- Logging tool

The primary users of this tool will be developers, project leads/managers, test team, Quality assurance team, Maintenance team, Software Architect and the customer. Developers could use this toolset to ensure that they write quality code so that the testing team consumes less effort. The Project leads/managers can use this to monitor the quality of the project which is the most important aspect of a product from a customer's perspective besides the functionality. The Quality assurance team uses this toolset the most to save their time and effort as the name of this toolset suggests. Maintenance team could use some of the tools in this set while they are about to make changes or fix latent errors. A software architect could use while working on a subsystem of an existing system. The customers could use this sometimes to evaluate the quality of the delivered product and ensure it meets their needs.

A brief description about each tool, the uses of the tool, the context where it is used, the activities, the structure and critical issues associated with each tool are described in detail in the coming sections supported with diagrams.

The next section describes the overall architecture of the system integrating the above stand alone tools into an environment suitable for quality analysis and the communication mechanism between the client and the server machines. The architecture describes the concept of the toolset system as a whole. The tools could be distributed across machines and the server-based tool holster manages these tools. The holster knows about all the tools, where they are located using a configuration file and is responsible for deploying them when needed. Each client has a GUI that displays the set of tools available. When the user selects a particular tool, he is provides with an interface of the specific tool for interaction. When a set of files are selected as input, the tool name and the file set is sent to the local executive which is then sent to the tool holster on the server through a remote executive. The tool holster spawns a new process for the selected tool and sends the file set to that process. After the tool executes, the result is sent back to the client through the executives.

All the communication between machines is through XML messages which could be achieved by using a WCF communication mechanism. This structure also supports legacy applications and tools by using a wrapper around them so that they need not have any knowledge on the message passing protocols that are prevalent across the system. Some other possible cases relevant to the tools and the tool holster are discussed in the next section.

Some critical issues associated with the entire system that need to be addressed are identified and solutions are discussed. They address the following types of issues:

- How does the QAT Client know where to send its messages?
- How does the remote executive know to which tool to dispatch an incoming message?
- What are the means to communicate and act upon tool errors?
- How to record, save, and view actions and results of the various tools?
    - Logged results for one run of one tool
    - Summaries of logged results for all tools in some period of time
    - Errors and actions for each of the tools

2. Architecture of the System

The system integrates all the stand-alone tools into a single environment and is mechanized with software agents using ensembles (file sets) for input and saving their results in XML-based log files. In this document, the term ensemble refers to input filesets for each tool. Each tool is partitioned into an interface and a set of core services, activated by messages from its interface. The left side portion of the diagram below depicts this simple scenario.

Figure 1 - Application of Pluggable Remote Executive Architecture

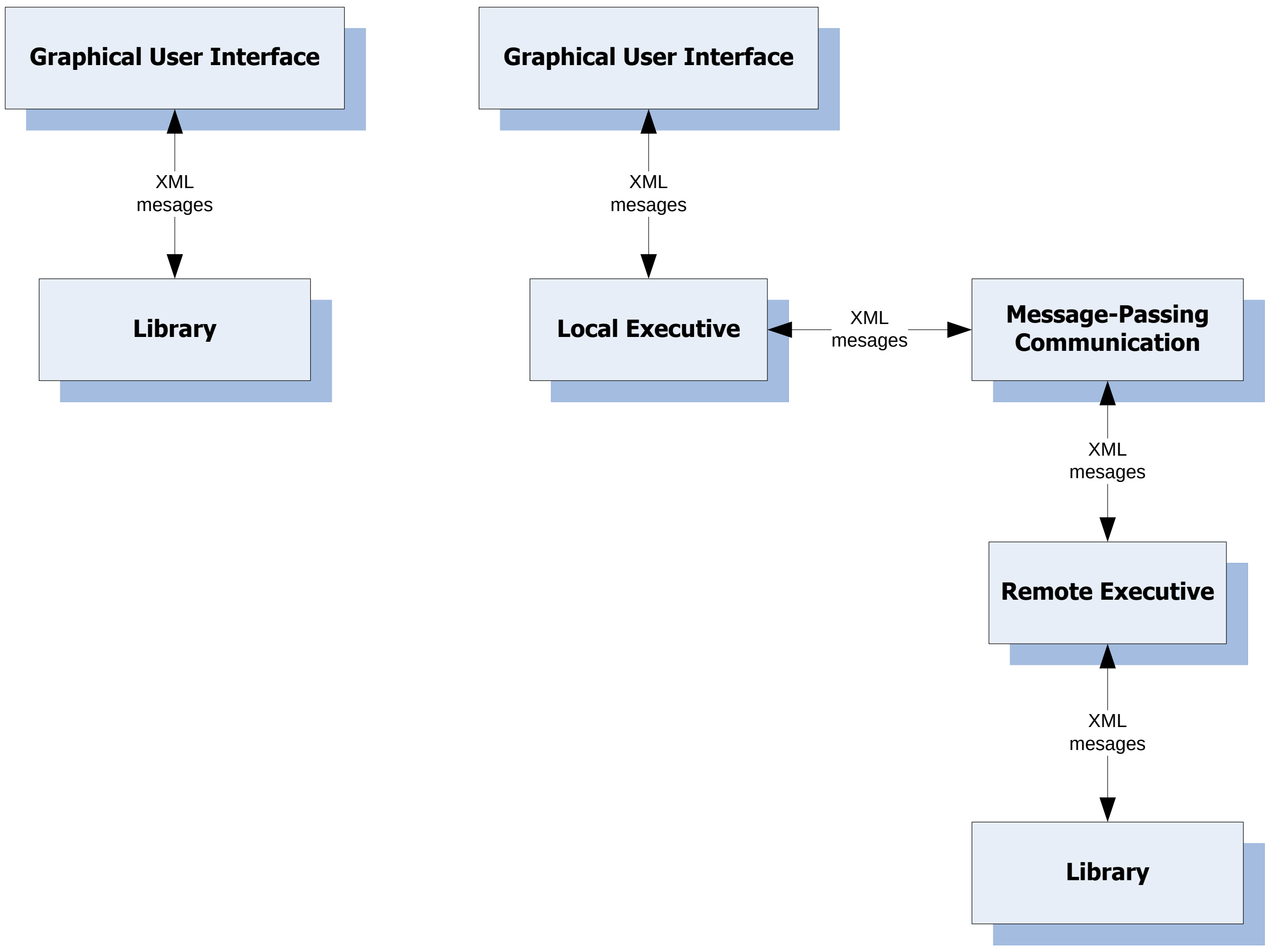


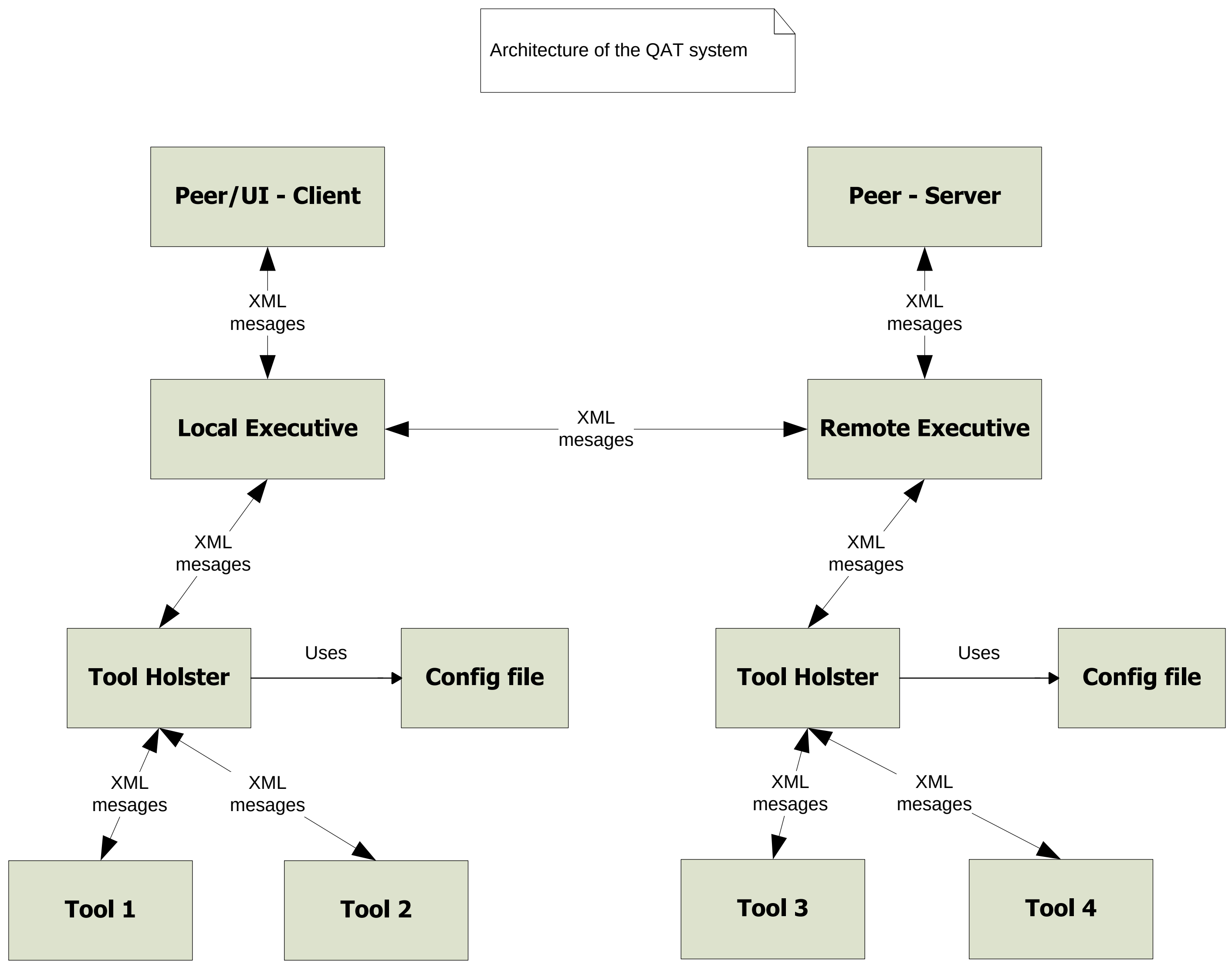
Fig1: Referred from Pr#5 Requirement

This structure also supports usage of the tool, by more than one user, when resident on a server machine across a network, each with its own interface. This scenario is depicted on the right hand side of the above diagram. The local executive provides a primary QAT Client interface that provides a selection of tools. When a tool is selected, the UI pops up an interface of the selected tool that supports definition of an ensemble or the selection of an existing ensemble. The remote executive serves to dispatch received messages to the appropriate tool library for analyses, and to return the logged results to the client.

2.1 Architecture of the QAT system involving Tool Holster and Tools

The diagram below in this section represents the architecture of the QAT system. It depicts the system in the context of two peer machines communicating across a network using XML messages. The client machine has the GUI for the tool set and a Local executive. All the machines in the network will have its own User interface. The server machine has a Remote Executive that dispatches messages received from the Client. The GUI consists of all the tools that the user can use as part of the tool set. When a user selects a particular tool, a separate interface for that particular tool will pop up for the user to interact with that particular tool. Most of the tools use file sets as input. The entire system interacts with a Source code Repository to define input file sets. Before delving more into the system, let us discuss briefly about the Code Repository and Tool Holster

- Code Repository:
  The code base for a project is typically stored in a source control repository. A source code repository is a place where large amounts of source code are kept, either publicly or privately. They are often used by multi-developer projects to handle various versions and developers submitting various patches of code in an organized fashion.
- Tool Holster:
  A Tool Holster is a special tool that manages the entire tool set and is responsible for deploying individual tools. It is usually server-based even though there could be a tool holster in the client machine in case there are some tools in the local client machine itself. A tool holster uses a configuration file in order to keep track of existing tools and newly added tools. The entire system has a pluggable architecture wherein existing tools could be removed or new tools could be added. In these scenarios, changes are made in the Configuration file so that the Tool holster always has the right track of tools that are available. The configuration file usually contains the name of the tool, the path where it is located and a small description about the tool. The structure of the configuration file and some of the possible issues with the holster and configuration file would be discussed at a later point of time.


Architecture of the QAT system
Peer/UI - Client
Peer - Server
XML mesages
XML mesages
Local Executive
XML mesages
Remote Executive
XML mesages
XML mesages
Tool Holster
Uses
Config file
Tool Holster
Uses
Config file
XML mesages
XML mesages
XML mesages
XML mesages
Tool 1
Tool 2
Tool 3
Tool 4

As soon as the user selects a fileset as an input, the selected tool along with its input ensemble is sent to the Tool holster through the Local executive. There are two cases to consider in this case:

Tool Holster in the Client machine:

One of the possible scenarios is that a tool holster is present in the Client machine itself because some of the tools may be present in the local machine. As tools are distributed across machines, this is a possibility. In this case, the Local Executive sends the tool name and the ensemble to the Local Tool holster. The Local Tool Holster checks if the requested tool is in that machine and if so deploys it and sends the fileset to it. After the execution of the tool, result is sent back to the local executive. In case, the requested tool is not available in that machine, it sends an appropriate message to the Local executive. As a result of this, the local executive sends a message to the remote executive which then dispatches the message to the server-based tool holster.

This then finds the location of the tool using the configuration file and then deploys the tool accordingly. After execution, it sends back the results back to the local executive.

Tool Holster in the Server machine:

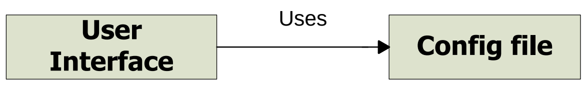
In case there is no Tool Holster in the local machine, the Local Executive send the tool name and fileset as messages to the Remote Executive which then dispatches them to the Server-based tool holster. This holster has information about all tools and their locations and hence deploys the appropriate tool together with its input. Once the execution of the tool is complete, the results are sent back to the local executive by the remote executive.

Deployment of Tools:

The Tool Holster can spawn a process for the tool and send the corresponding input to it. In case the tool is available in the local machine itself, then the tool holster present in the local machine is responsible for deploying the tool.

User Interface on all machines:

The User interface of the tool set on all the machines initially contains a list of all the tools that are available. To know the list of all tools, the UI uses a configuration File which contains a list of all the tools available across the network. This configuration files should be synchronized with the configuration file used by the Tool holster where all the tools are registered. How this is achieved is discussed in the Critical issues section.



2.2 Activities

The major functionality of this system revolves around the Local and Remote Executives and the Tool Holster where all the communication takes place using XML Messages. This section describes the major tasks that this toolset would perform and is generally applicable for those tools that take filesets as inputs. There are some tools like the tracking tools which doesn't deal with ensembles but have a different kind of interface and takes different types of input. The activities associated with those kinds of tools are described when describing about that tool functionality.
The major tasks would be in a sequence as described below.

2.2.1 User selects a tool

The GUI displays a list of tools that could be selected by the User. The user selects a particular tool of his choice.

2.2.2 An interface pops up for that tool

Each tool has its own interface and hence on clicking a particular tool, its interface pops up so that the user can interact with that tool. A model of that interface for some of the tools would be described in the coming sections when discussing more about each tool in detail.

2.2.3 Select a new/existing ensemble/fileset for input

The user can define a new fileset or an existing fileset for input. He/She would be provided with options to choose both. The input fileset is then sent to the tool for its execution.

2.2.4 User Selects Execute on the tool with fileset

Once the User defines a fileset as input, he selects an option to analyze/inspect the files depending on the tool from the interface

2.2.5 Request (toolname+ensemble) wrapped in an XML Message and sent to Local Executive

An XML Message request is formed with the tool name and the fileset containing the file paths and this message is then sent to the Local Executive which is the executive running on each machine.

2.2.6 Request passed on to Local Tool Holster if available

The Local Executive checks if a Local Tool Holster is available in that machine and if so passes the XML message to the holster.

2.2.7 If Tool Present in the local machine, holster deploys it

The Local tool holster checks if the requested tool is present in that machine. If available, it deploys the tool by spawning a new process.

2.2.8 If not, holster passes a message back to executive

In case the tool is not installed in that machine, then the holster sends a reply back to the executive denoting that the tool could not be found.

2.2.9 If tools holster or tool not available, message sent to Remote Executive

In cases where the holster itself is not available locally, or the requested tool does not exist (receives a message from holster for this- Step 8), then the Local Executive sends the message to the Remote Executive present in the Server.

2.2.10 Remote Executive dispatches to Server-Based Tool Holster

The Remote Executive after receiving the message just dispatches the message to the Server-Based tool holster that has the knowledge of all the tools.

2.2.11 Tool Hoslter checks the location of tool using config-file

The tool holster uses the configuration file to know the path of the tool as all tools would be registered with the server-based tool holster.

2.2.12 Tool Holster deploys tool with the ensemble extracted from message

The tool holster then deploys the tool with the ensemble extracted from the message. The tool retrieves the files from the code repository based on the paths in the fileset.

2.2.13 Results and Actions of tool are logged

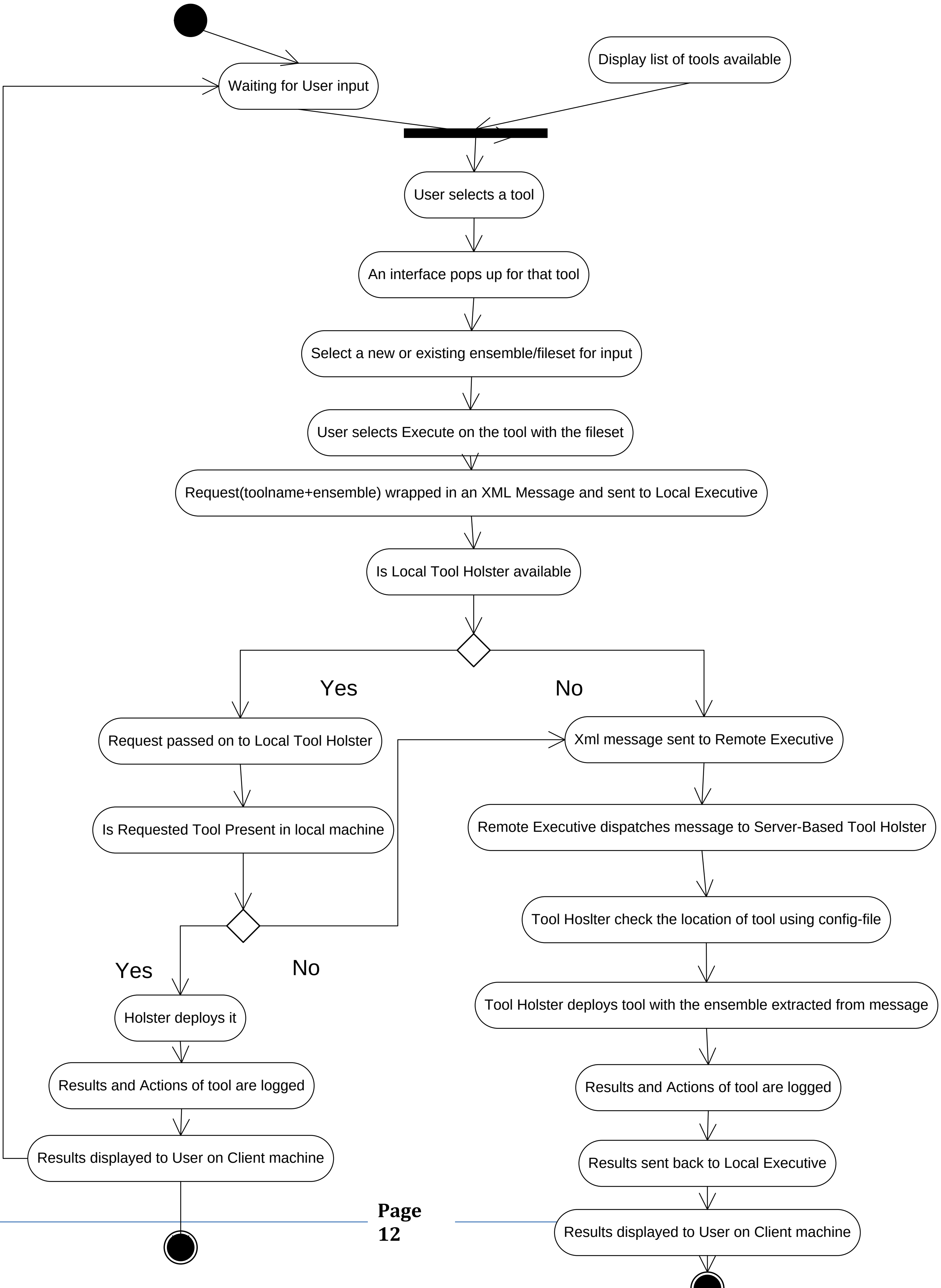
The Results and Actions of the execution of the tool are logged to be sent to the client and for history purposes so that at any point of time, a tool’s logging could be retrieved and seen.

2.2.14 Results sent back to Local Executive

The Results are then finally sent back to the local executive through the remote executive

2.2.15 Results displayed to User on Client machine

The Results of execution are displayed to the user on his screen. The user can quit or select another tool of his choice.


Display list of tools available
Waiting for User input
User selects a tool
An interface pops up for that tool
Select a new or existing ensemble/fileset for input
User selects Execute on the tool with the fileset
Request(toolname+ensemble) wrapped in an XML Message and sent to Local Executive
Is Local Tool Holster available
Yes
No
Request passed on to Local Tool Holster
Xml message sent to Remote Executive
Is Requested Tool Present in local machine
Remote Executive dispatches message to Server-Based Tool Holster
Tool Hoslter check the location of tool using config-file
Yes
No
Holster deploys it
Tool Holster deploys tool with the ensemble extracted from message
Results and Actions of tool are logged
Results and Actions of tool are logged
Results displayed to User on Client machine
Results sent back to Local Executive
Results displayed to User on Client machine

2.3 Partitions

Bases on the tasks and the activities within each task, the entire system is subdivided into 11 modules at a higher level as described below:

1) Client UI Module
2) Tool Interface Handler Module
3) Tool–UI Module
4) Local Executive Module
5) Fileset Handler Module
6) Fileset Store Module
7) Tool Holster Manager Module
8) Communication Module
9) Remote Executive Module
10) Logger Module
11) Log Storage Module

The responsibilities of each module, dependencies between them are described in detail below:

2.3.1 Client UI Module

This module is associated with the main graphical user interface that displays a list of available tools across the network to the user. This is achieved with the use of a Configuration file which contains a list of all the available tools. It can be considered to have details like tool name, path and a small description. This configuration file is essentially the same as one used by the server-based tool holster which contains similar details. It is very important that the configuration file used by the Client machines is in sync with the tool holster's config file. This is to ensure that the user always sees newly added tools and also does not see any removed tools.

2.3.2 Tool Interface Handler Module

This module is to manage the various tool interfaces that are part of the tool set. Whenever the user selects a tool from the main GUI, a separate interface for that tool pops up on the screen. The user then interacts with that interface. As there are multiple tools, it would be useful to have a separate module to map and manage these tool interfaces and this module would serve that purpose.

2.3.3 Tool UI Module

This module is associated with the interface of each tool and there would be a module for each tool in the toolset. This module is specific for each tool as interface would be entirely different for certain tools like tracking tools which does not take fileset as input.

2.3.4 Local Executive Module

This module accepts the tool name and the fileset and constructs an XML Message request. This message is then sent to the Local tool holster if one is available. This module has to check if there is a Local tool holster in that machine by some means. After the tool execution the local executive sends the results back to the UI module.

2.3.5 Fileset Handler Module

This module takes care of handling input filesets for each tool (apart from tools like tracking tools which doesn't take filesets as input).The user is allowed choose files from a remote code repository which is managed by this module. It is assumed that each machine knows the location of the code repository and hence connects to that machine when the user interface starts up. The selected filesets are sent to the Fileset Storage module to be stored for a later point of time. At any time, the user can define new filesets or use existing filesets. This feature prevents the user form defining the filesets each time when he is going to use multiple tools with the same fileset as input.

2.3.6 Fileset Storage module

This module is responsible for storing filesets already defined by the user. This is to enable the user to select existing filesets as input. The filesets are sent by the Fileset Handler module to be stored to this module.

2.3.7 Tool Holster Manager Module

This module is responsible for knowing all the information about each tool and deploying the requested tool. This uses a configuration file that contains all tool information like name, path and small description. Every tool a new tool is installed on a machine, it has to be registered with the tool holster for which an entry is made in the configuration file. Only if the newly added tool is registered with the server-based tool holster, then it could be used by client machines. In some machines, there might be a local tool holster responsible for managing and deploying tools available within that machine. A separate config file is used for that purposes. After deploying the tools and the tool executes, its actions and results are sent to the Logger module for storage purposes.

2.3.8 Communication Module

This module is responsible for establishing communication between the Client and Server. Once the client selects a set of files for analysis and a tool, a communication channel is set up using facilities of WCF. The communication takes place based on some predefined protocols and is based on message-passing. In message-passing technique, the sender encodes a request in an XML based message, the receiver decodes the message, acts on its contents, and may send back a reply, also encoded as a message. Using a message contract, WCF accepts a message object with public properties that it serializes into an XML string, sends to the receiver, and de-serializes the string into a message object on that side.

The module uses Queues on both the server side and client side. The requests from the sender are placed in the sender's queue and at the server side, the requests are put into the queue and are processed one by one. This is the same scenario for sending and receiving a response. Hence both the client and server have two blocking queues for their processing.

2.3.9 Remote Executive Module

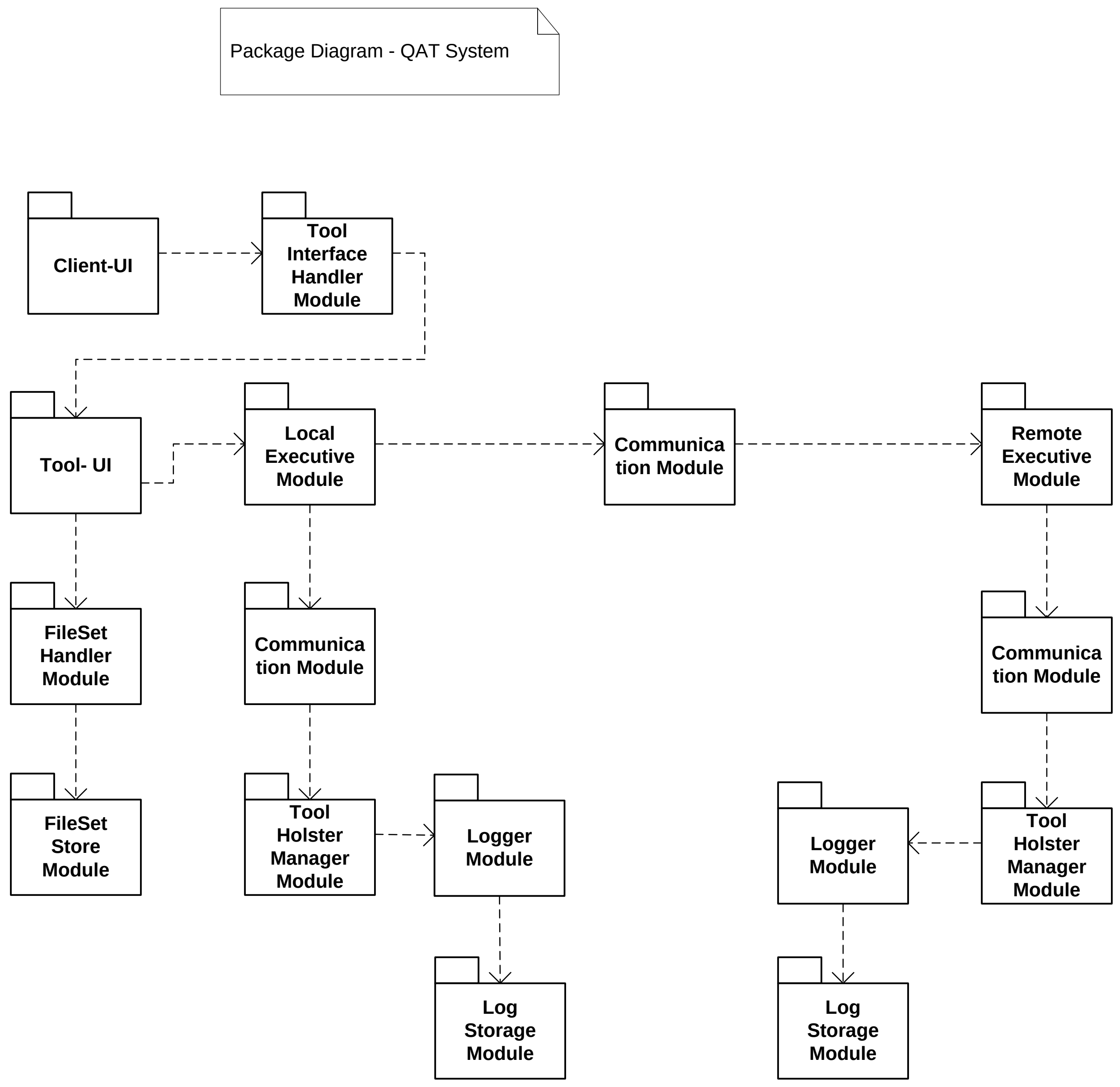
The Remote Executive module is responsible for receiving messages from the Local Executives of client machine and then dispatching it to the server-based tool holster. Its main job is to dispatch messages for which it uses a separate child thread operating on a queue.

2.3.10 Logger Module

Whenever a tool is executed, the actions and results of it should be logged somewhere so that the logs could be seen by the user after its execution or at a later point of time. The logger module sends the actions and results to the Log Storage module for storing them as an XML-based log. The logs are time-stamped and are queued up for storage. This module is available in all machines wherever there is Tool holster as there is a possibility that a local tool holster could exist in a client machine.

2.3.11 Log Storage Module

This module stores the actions and results of tool execution as an XML-based time-stamped log. It holds a queue for this purpose. The queue is necessary as many QAT tools may be running concurrently for one or more clients. Whenever the user wishes to see the logs of a particular tool, this module is contacted which returns back the corresponding logs. This module is available in all machines wherever there is Tool holster as there is a possibility that a local tool holster could exist in a client machine.


Package Diagram - QAT System
Client-UI
Tool Interface Handler Module
Tool- UI
Local Executive Module
Communication Module
Remote Executive Module
FileSet Handler Module
Communication Module
Communication Module
FileSet Store Module
Tool Holster Manager Module
Logger Module
Logger Module
Tool Holster Manager Module
Log Storage Module
Log Storage Module

2.4 Uses

The uses of this system are based on the uses of individual tools that are integrated into the system. Using all the tools, this system helps to analyze the quality of a project's evolving code baseline. The uses of each tool are described while discussing about each tool in their corresponding sections.

2.5 Critical Issues

2.5.1 How does the QAT client know where to send its messages?

It is essential for the QAT clients to know where to send the messages. This could be done by having the server broadcast a message with its location information to all client machines in the network. The QAT clients can store this information and use it for sending messages thereafter.

Another possibility is to have each QAT client machine query all the machines in the network to know which one is the server. This method is less efficient than the former one.

2.5.2 How does the remote executive know to which tool to dispatch an incoming message?

The remote executive has to dispatch the message received from client to a particular tool. But since the Remote executive doesn't have any information about the tools, it dispatches the message to another component which has complete knowledge about all tools. This component is nothing but the Tool holster. Hence the Remote Executive just dispatches the incoming message to the Tool Holster which then dispatches it to the appropriate tool.

2.5.3 How to Communicate and act upon tool errors?

There are possibilities that the tool doesn't execute properly and has crashed due to some reasons. There may be no log files generated. In this scenario the tool holster can set a timeout for each tool execution based on its average execution time for a specific number of files. The timeout could be large enough so that probability of tool not finishing its execution within the timeout is very less. When the timeout expires and there are no log file generated within that time, the holster can decide that the tool has some errors and hence can communicate a message to the client indicating that there has been an error and can ask the user to try once again.

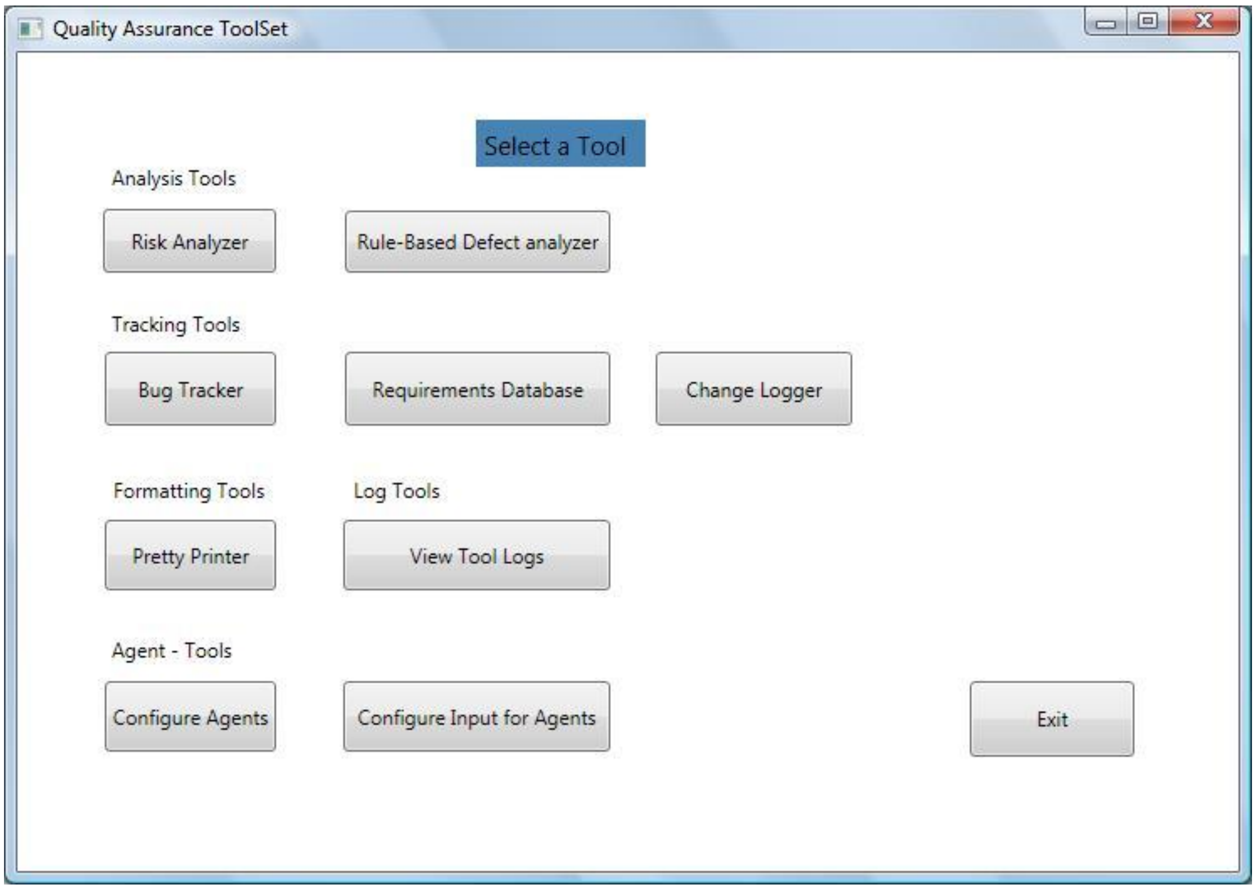
2.5.4 How to record, save, and view actions and results of the various tools?

It is essential to consider the following cases while discussing about this issue

- Logged results for one run of one tool
- Summaries of logged results for all tools in some period of time
- Errors and actions for each of the tools

The logs for each tool are time-stamped and stored in a folder corresponding to that tool name. Besides that, the folder with tool name can contain sub folders with dates to store logs of that tool run on that particular date. The organization can be at any level, creating folders for each month/week/day. This organization helps in easy retrieval of actions and results for a period of time. For every new tool, a folder needs to be created. For this, the logger while storing the logs can check if any folder for such tool exists. If not, it can create a new one for that tool.

2.6 Views



2.7 Usage of Legacy Tools and Applications

This system architecture also supports the incorporation of legacy tools that know nothing of the message-passing and data sharing protocols used by QAT, as shown in Figure below.

Figure 2 - Pluggable Remote Executive Architecture

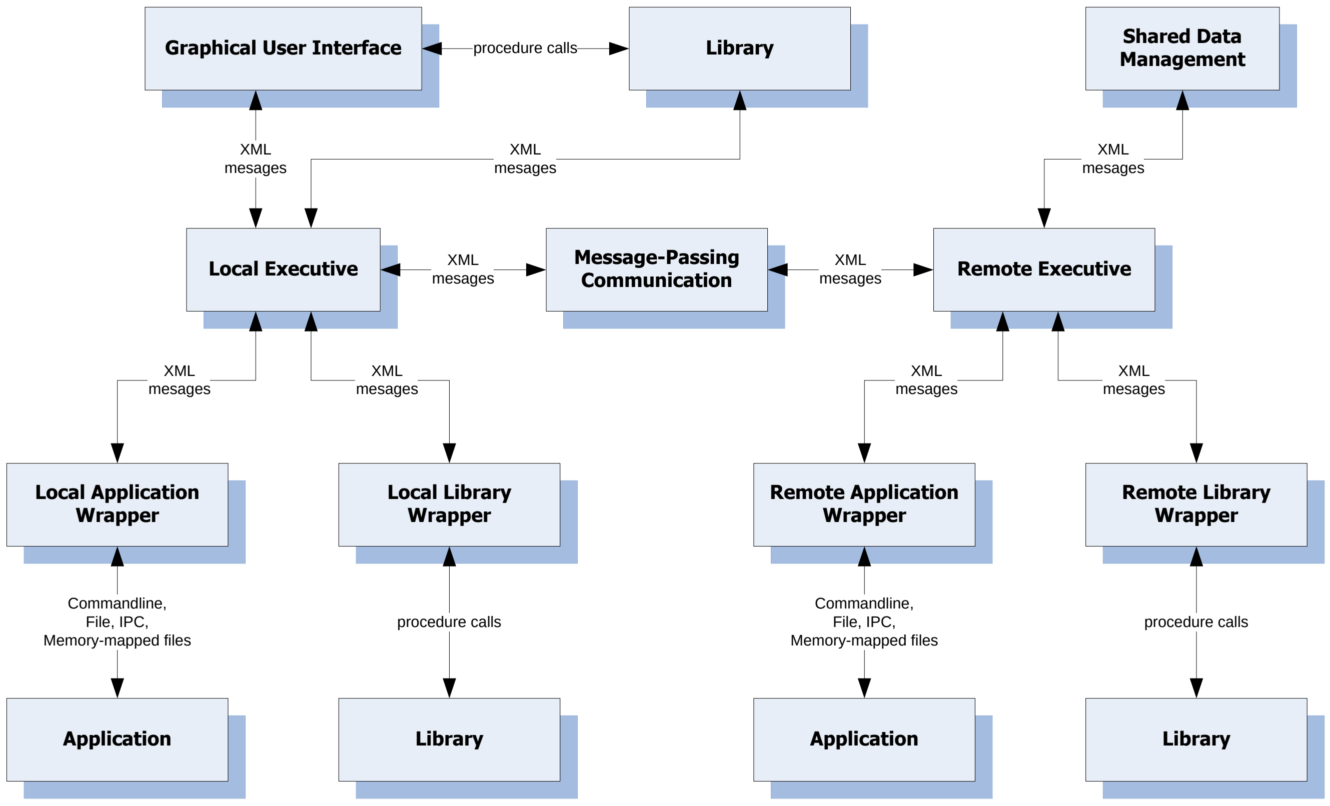


Fig: Referred from Pr #5 Requirement

Through the use of a wrapper that understands QAT protocols, and is specifically designed to work with the tool, incorporation of a legacy tool is possible. The nature of any specific wrapper is determined by whether the legacy tool is represented by a library or an executable.

In the case of a library, the wrapper consists of a message queue, dispatcher, and list of references for replies, as shown in Figure below.

Figure 3 - Library Wrapper

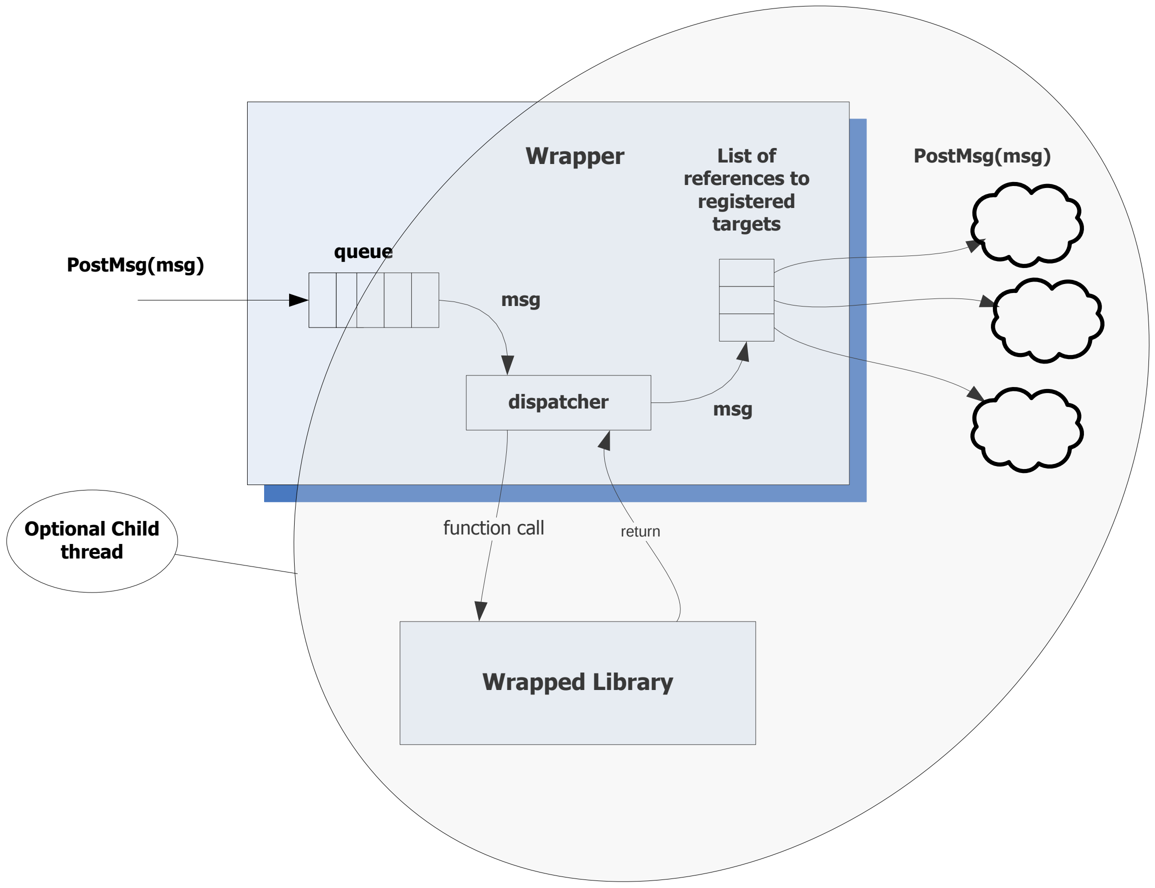


Fig: Referred from Pr#5 Requirement

The dispatcher is designed to map messages sent to the tool into calls into the libraries public interfaces.

Wrappers for libraries – maps messages into library method calls as shown in the above fig.

Wrappers for executables - differ only in that the dispatcher translates messages into command line arguments, spawns a process for the tool executable, and redirects its standard output to a file as shown in fig below.

Figure 4 - Application Wrapper

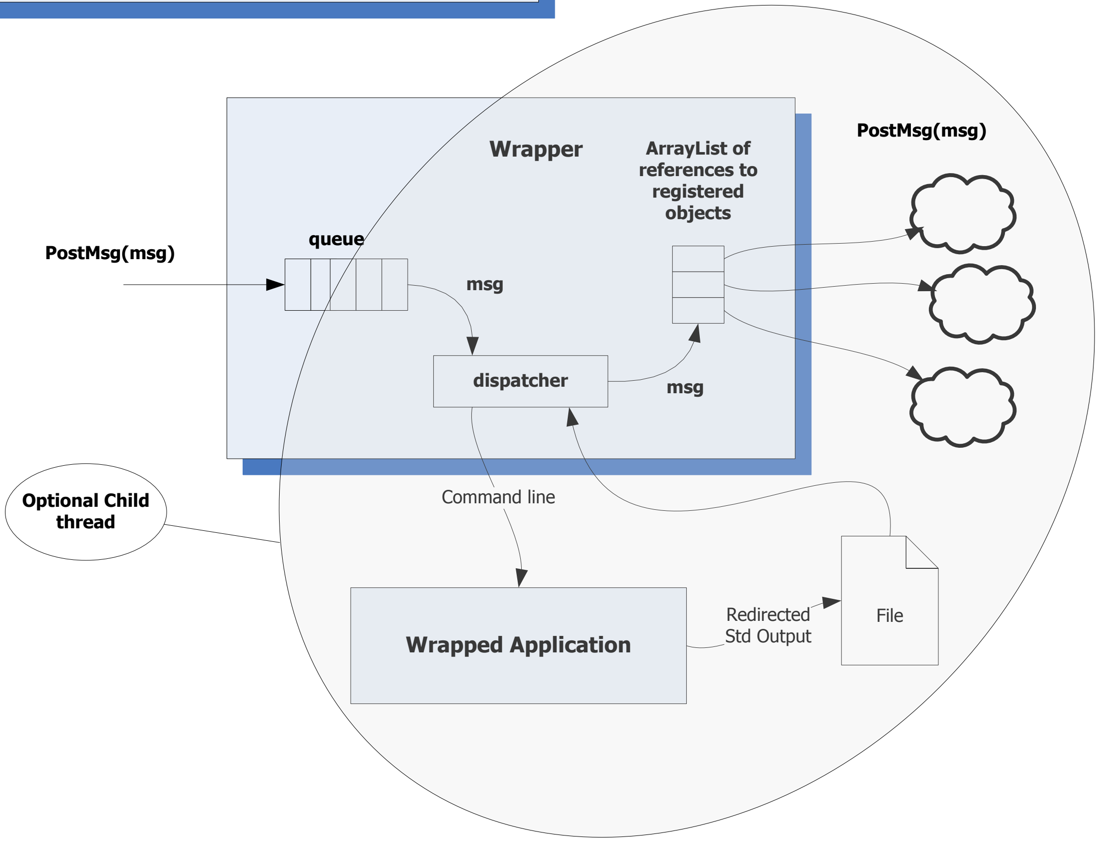


Fig: Referred from Pr#5 Requirement

3. Description of Users/Actors for the tools

Actors are those who use and interact with the tool. It is most essential to design and develop the tools in such a way that all the actors get valuable information out of it and can interact with it in a simple and effective manner. This section describes in detail the possible users/actors, who interact with the toolset.

3.1 Developer

A Developer is a team member responsible for development of the project. The tool helps a developer in increased understanding of a project, the risks associated with it and their impact.

3.2 Project Leader/Manager

A person who leads or manages a team and is responsible for scheduling the project, estimate time/cost needed for the development, monitor the progress of the project and notify customers of change in schedules and estimated time. This tool would be very useful for managers as risk analysis is an integral part of project management.

3.3 Test team

A team responsible for writing test plans and test cases to thoroughly test the functionality of various components in the project. This tool helps in determining the major components with high risks that need to be tested and establishing a testing priority based on each component's risk.

3.4 Quality Assurance Team

A team responsible for verifying the quality of the project that needs to be delivered to the customer

3.5 Maintenance team

A team that is responsible for maintaining the project after its delivery. It is also possible that a project is completely maintained by the customer itself.

3.6 Software Architect

A Software Architect is a person responsible for describing the architecture of the system and explains the concepts which bind the development team together. When the software architect works on a new sub-system of an existing system, or restructure an existing system, this tool can come in handy by giving statistical information of risk associated with the existing system which is essential and reduces further risk.

3.7 Customer

The potential buyer of the project after it is developed and tested.

4. Risk Analyzer tool

4.1. Introduction

Almost everything done in today's business world involve some kind of risk. This verdict needs more attention and focus in development of large software projects in an organization. Henry [2004] in his book defines Risk as,

"An event, development or state in a software project that causes damage, loss or delay"

Another definition of risk could be,

"Perceived extent of possible loss"

This tool that performs risk analysis helps in identifying and assessing factors that may jeopardize the success of a project or achieving a goal. It also helps in determining what components to test, the testing priority, and depth of testing. As a result of risk analysis, a project is better understood, can be better planned and managed and can be more profitable. Typically nowadays, the environment in which a large software project is developed is big involving many teams working on different parts of the project and possibily in different geographical locations. This tool is a Risk analyzer wherein an analysis of risk is done on a set of files.

4.1.1 Objective and key idea

Its main objective is to perform an analysis of risk, based on an existing model, of a set of source files defined by the client's user, and after analysis, would display on the client the file names ordered by risk. This helps in evaluating a measure of project risk for each file in the project baseline. The key idea of performing a risk analysis is based on an existing model. The idea behind this model is that risk incurred by the propensity of changes in one component to cause changes in another. These are quantified using notions of importance I and test risk T.

Importance measures how likely a change in file is to cause changes in its callers and this probability of change is measured as 'ɑ'.

A file's test risk is its quality factor plus the risk it derives from the possibility that a called file may have errors which, when fixed, cause changes in the original file.

These two terms are described more in detail in the coming sections.

After quantifying these, Project risk is Ri, for the ith file is defined as the product of its importance and test risk, Ri = Ii * Ti.

4.1.2 Obligations

The tool will be developed as an efficient and user-friendly one meeting all the stated requirements. The main obligations of the tool would be

- The input files are analyzed for risk based on the defined model.
- After the analysis is conducted by the tool, the results are displayed, in an efficient and easy-to-understand way.
- The output and the directory view for input would be a Graphical user interface, based on Windows Presentation Foundation (WPF)
- The output should display file names ordered by risk which are based on calculation of importance and test risk for each file.

4.1.3 Organizing Principles

The architecture of this system is organized into three layers:

1) Presentation layer
2) Processing layer
3) Data storage layer

All these three layers communicate using the communication layer implemented with the help of WCF facilities. This is one of the organizing principles.

4.2. Uses

This section is to brief out how the actors interact with the tool how it serves them, their expectations and needs out of it and the behavior of tool based on its actors. All these are described below.

4.2.1 Developer – Changes and code improvements, risk and impact during changes, user interface:

Important for changes and code improvements:

This tool is beneficial in situations involving unexpected new development or changes in an existing project. When a new developer joins the team, he/she may be entitled to work on a complex project and assigned to do some changes or code improvements. The tool facilitates in increased understanding of the project by listing out the risk information associated with each file in the project. This helps in determining files where code improvements could be made without much risk. Also he/she can understand the risk associated with each change that they need to do and their impact which is very important for the system to remain stable.

Risk and Impact during changes

When there are code changes to be made, before the actual implementation, it is very essential to analyze the risk associated with the change and the impact of it on other code parts. This tool facilitates in knowing that valuable information which can be used to decide on specific implementation strategies. If the change involves high risk it could be reported to the concerned manger and then to the customer and finally a suitable action could be taken. Depending on the risk associated with the change, the developer can carefully plan and schedule and give a proper estimate to the manager.

4.2.2 Project leader/Manager- Risk management, Decision making, Cost and schedule

Risk management:
Risk management is an integral part of project management which follows Risk analysis. This tool helps a manager in understanding the risks associated with a project and formulating risk plans accordingly in terms of cost and time. It's the manager's responsibility to assess the risks and decide on what actions to take to minimize disruptions to the project plan. The manager could decide on strategies that could be useful for controlling and managing risk based on the available information from this tool and should also ensure that the strategies are cost-effective. This tool enables managers to justify their decisions and enable more efficient risk management. Risk management serves as the basis for crisis prevention.

Decision making
This tool bolsters a manager in better decision making by carefully assessing the risk information obtained out it. They can decide on strategies to control risk and allowing incorporation of new changes into an existing project. Their decisions based on risk and its impact increases likelihood of successful project completion meeting the cost, time and performance objectives. The managers can also have the statistical information of historical risks associated with existing projects to decide on new projects and planning them.

Cost and schedule
One of the primary responsibilities of a manager is to carefully decide on the budget and schedule for a project. When there is no risk analysis done for a project, it fails to,

- Keep within budget
- Achieve required completion date
- Achieve required performance objective

Hence risk analysis is an integral part and based on the risk information, the managers can take appropriate actions, evaluate the expenditure and capital budget and sanction them accordingly and change project schedules. This way a project can be made more profitable satisfying all the customer needs and delivering them in time.

4.2.3 Quality Assurance Team – Estimate quality, reviewing test cases and plans

Reviewing test cases and plans
The Quality Assurance Team is responsible for ensuring of the quality of the final deliverable project. As part of this process, they may have to review test plans and test cases. By analyzing the risk information from this tool, they can verify if the test plan has been modified as per calculated risks and ensure if components with higher risks are tested

comprehensively and thoroughly. They can calculate the depth of testing needed for a particular file based on its risk information and make sure that it is achieved by suitable and sufficient test cases.

Estimate quality

The quality of a software project can be assessed to some extent based on risk associated with the project. A project with high risk for several of its components is definitely prone to future errors and instability. The risk information helps this team in taking appropriate decisions and actions in finding new strategies to test the project so that quality is not in stake.

4.2.4 Test team – Writing test plans/cases comprehensively

A major purpose of this tool is to help determine, what components or files to test, decide the testing priority based on risk information and know the depth of testing required for each component or file. It also helps in some cases in identifying what not to test. These information leads to a moderate or extensive change in test cases and overall test plan. A software tester can identify high-risk applications that should be tested more thoroughly. He/she can also identify specific components within an application that need to focused and tested comprehensively.

The results from this tool can be used during test planning phase to determine priorities for software under test.

4.2.5 Maintenance team – Fixing latent errors and incorporating small changes

This team is responsible for maintaining the project once it is delivered to the customer. Some of the important tasks that are part of maintenance are:

- To troubleshoot in case of errors/to fix latent errors.
- To provide support for the project.
- To incorporate small changes when required.

For fixing latent errors it is really important to analyze the risk impact on other files that are dependent on the file containing the error. This tool serves the purpose and helps in knowing the risk impact and take action accordingly and cautiously. This scenario is also applicable for making small changes in an existing code during the maintenance phase.

4.2.6 Software Architect – Restructure existing systems, architecture for new subsystems

One of the important jobs of software architects it to restructure an existing system due to some reasons or construct the architecture of a subsystem of an existing system. For any of these jobs, an increased understanding of the existing project and its historical risk information is essential and prove to be very useful. These are served by this tool.

4.2.7 Customer

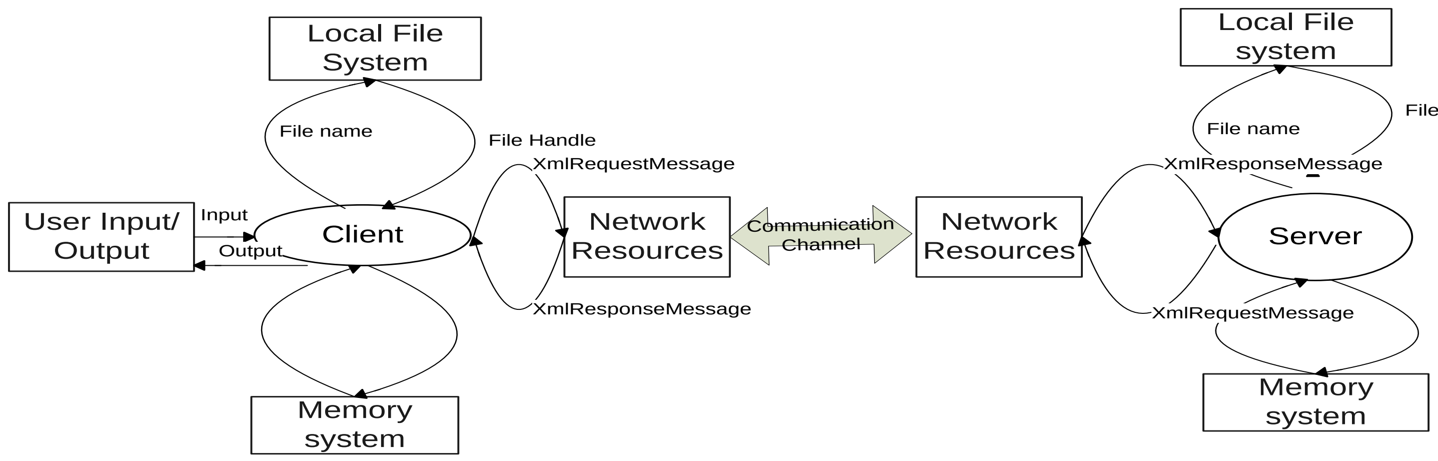
Once receiving the project that is being delivered, the customer has to maintain it for which he should be able to understand the structure of the project. Also for making some small changes after delivery, it should be possible for the customer to know the impact of the changes. This tool would support the customer in this scenario and makes his life easier by providing all the information that he needs to analyze the impact and to identify the context of the change. Also the customer can easily get to know the files which could be reused. This information would be pretty useful for their future projects and would add more value.

4.3. Extension or changes to the tool

The tool could be extended so that besides Risk calculation, it can be used to show some information on rough delay in estimated schedule of a project based on the risk value of its files. This could be done using some prediction techniques and historical risk information of previous projects. This feature could prove to be very useful for a Project Manager and save a lot of time.

4.4. Context

Context diagram gives a bird's eye view of the system and presents the environment in which the tool would be functioning. The diagram in this section depicts the environment in which this tool would be used.


Local File System
File name
File Handle
XmlRequestMessage
User Input/ Output
Input
Output
Client
Network Resources
Communication Channel
XmlResponseMessage
Memory system
Network Resources
Local File system
File name
File
XmlResponseMessage
Server
XmlRequestMessage
Memory system

4.5. Application Activities

The major implementation part revolves around calculation of importance and test risk for each file from which the Project Risk is calculated for each file. Importance and Test Risk are part of the model that is being used for Risk analysis. To serve this calculation, an iteration technique called Jacobi iteration is used which is applied for the set of input files. This iteration is made easier by performing it on a graph of files and their dependent files. Besides implementation, establishing communication and managing it is one of the most important activity of this tool. The major tasks that this tool would perform would be in a sequence as described below.

4.5.1 Get source files for risk analysis from client

The client selects a path containing a set of files that need to be analyzed for risk. Retrieve the file paths selected by the client and store it in a list.

4.5.2 Retrieve files from repository

The tool establishes a communication between itself and repository and retrieves the source files for analysis using the set of paths it has retrieved.

4.5.3 Perform Dependency Analysis

The set of input files need to be analyzed initially for dependencies between them. This is done by performing two passes over the list of files.

<u>Pass1:</u>

Detect all type definitions in each file by parsing them and store the data in a Hashtable where the key is the typename and value is the containing file name.

<u>Pass2:</u>

The list of files is parsed again to analyze if the files are using other types defined in other files. For this operation, the Hashtable constructed in Pass 1 is made use of and file dependencies are stored in an appropriate data structure. For example it could again be a Hashtable where the key is the filename and value is a list of all Called files.

Called files -> Other Files called from the owner file.

At the end of this Pass we have a list of all the files with the called files for each file stored in a Hashtable.

4.5.4 Construct a graph of Called files

In this step, we construct a graph of called files. The graph is implemented as an adjacency structure. It has a vector containing all the files are vertices. The called files for each file are represented as edges. The edges are implemented by having a reference from each vertex file to all the called files in the same vector. For this purpose, we could create a File object to represent each file as a vertex. The File object may contain the following attributes.

name ->denotes the file name
importance ->stores the importance value calculated for the file (explained in step 4.8)
smell_factor ->stores the smell factor value calculated for the file (explained in step 4.5)
test_risk ->stores the Test Risk value calculated for the file (explained in step 4.6)
project_risk ->stores the Project risk value calculated for the file (explained in step 4.9)

4.5.5 Calculate the smell factor β for each file

The smell factor β is required to be calculated in order to find the Test Risk of a file. The smell factor is calculated using the equation,

$$\beta_i = \frac{1}{N}\sqrt{(\frac{m_{1i}}{M_1})^2 + (\frac{m_{2i}}{M_2})^2 + \dots + (\frac{m_{Ni}}{M_N})^2}$$

$$\beta_i = \frac{1}{N}\sqrt{\sum_{j\in(1,N)} (\frac{m_{ji}}{M_j})^2}$$

Where m -> measured metric

M -> Boundary value for the metric

N -> Number of metrics used in the calculation.

The metric could denote anything related to code. For the implementation, we could use maximum values of cyclomatic complexity and total number of lines in a function in a particular file. These are calculated by analyzing the code in each file. The calculated values are updated in the corresponding attribute of each File object present in the graph.

4.5.6 Calculate Test Risk for each file using Jacobi Iteration technique

Jacobi Iteration technique is an iterative method to solve a system of linear equations. The iterative process is repeated on the equations until it converges to some value. The Test Risk is calculated using the following equation,

$$T_n = \beta_n + \sum_{AllCalled} \alpha_{mn} T_m \qquad T \in [1, \infty)$$

The above forms a set of linear equations when represented for each file. We have already calculated β in the previous step. 'α' is the probability of change which almost always takes the value 0.1 for our implementation. The iterative technique is started using some initial approximations. In this calculation, the initial approximation values for Test Risk T of a file would be the values of their corresponding smell factors. As using linear equations would be complex for an implementation, we could use a graph constructs to perform the same. This is the idea for constructing a graph of Called files in one of the previous steps. The calculation of Test Risk of a file depends on Test Risk of all the files it calls. The calculated values are updated in the corresponding attribute of each File object present in the graph.

4.5.7 Construct graph of Callers

In this step, we construct a graph of caller files. The graph is implemented as an adjacency structure as in the case of called files. It has a vector containing all the files are vertices. The caller files for each file are represented as edges. The edges are implemented by having a reference from each vertex file to all the caller files in the same vector. This graph is constructed with the help of the already constructed graph of called files as this is essentially the inverse of that graph.

4.5.8 Calculate Importance I for each file using Jacobi Iteration technique

The Importance of a file is calculated using the following equation,

$$I_i = 1 + \sum_{AllCallers} \alpha_{ij} I_j \qquad I \in [1, \infty) \qquad \alpha \in [0,1]$$

The above forms a set of linear equations when represented for each file. 'α' is the probability of change which almost always takes the value 0.1 for our implementation. The iterative technique is started using some initial approximations. In this calculation, the initial approximation values for Importance I of a file would be taken as 1. As already pointed out, using linear equations would be complex for an implementation, and hence we could again use a graph construct to perform the same. This is the idea for constructing a graph of Caller files in the previous step. The calculation of Importance of a file depends on Importance of all the files that it is called by. The calculated values are updated in the corresponding attribute of each File object present in the graph.

4.5.9 Calculate Project Risk for each file using Importance and Test Risk values

Once Importance and Test for each file is calculated, the final step of calculation is to find the Project Risk for each file which is essentially the product of its Importance and Test Risk. The Project Risk is denoted by the equation,

$$R_i = I_i \times T_i \qquad R \in [1, \infty)$$

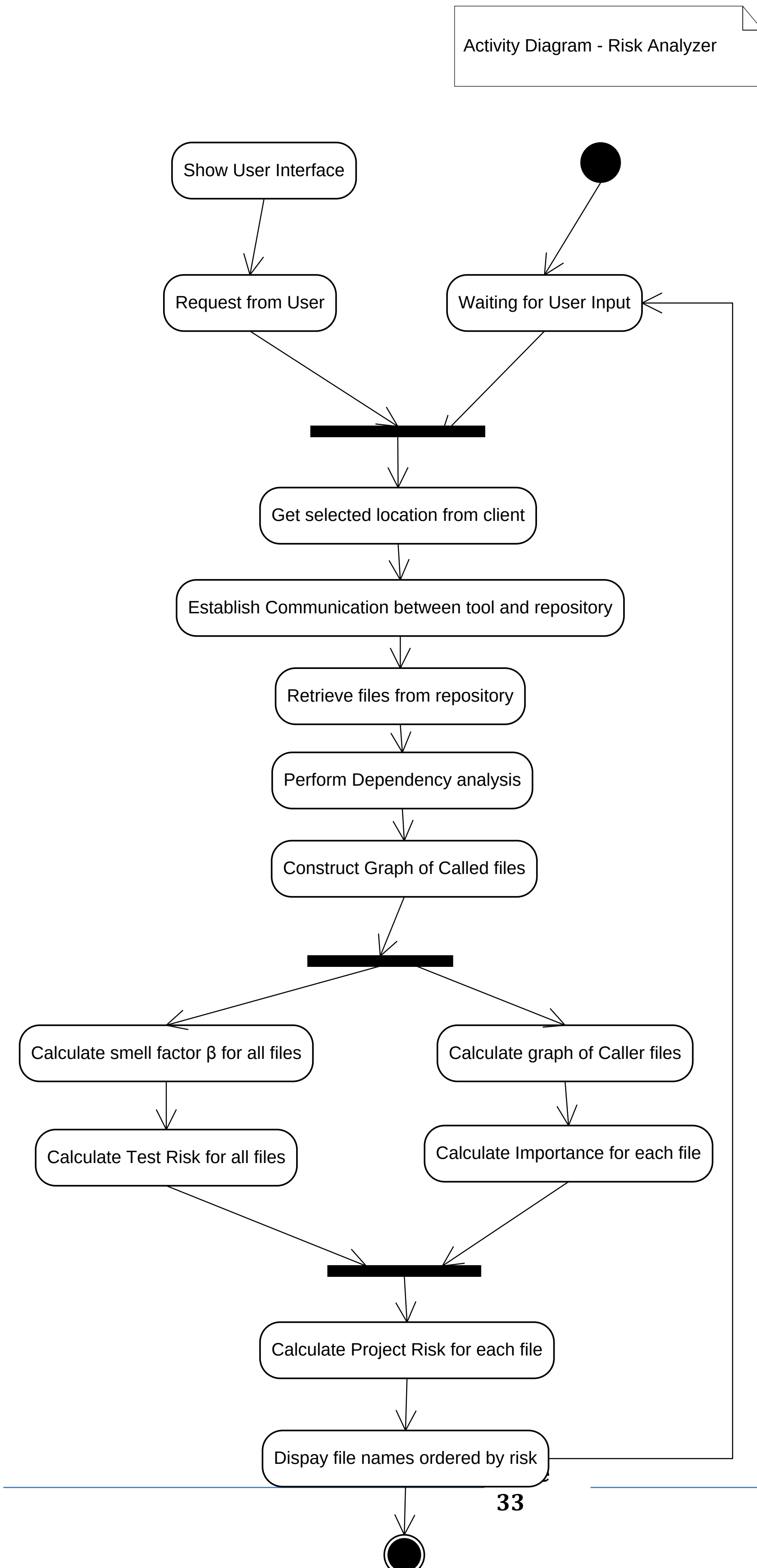
The calculated values are updated in the corresponding attribute of each File object present in the graph.

4.5.10 Send back result of analysis to client

The tool after its analysis sends back the list of File objects created with all information ordered by risk to the client. This is again sent by encoding the request in an XML-Based Message object.

4.5.11 Display files ordered by risk

The final step after receiving the result is to display the file names to the client ordered by risk value. The Project Risk value along with other values are available in a File object for each file and hence can be retrieved and presented to the user after sorting the values of Project risk for each file.


Activity Diagram - Risk Analyzer
Show User Interface
Request from User
Waiting for User Input
Get selected location from client
Establish Communication between tool and repository
Retrieve files from repository
Perform Dependency analysis
Construct Graph of Called files
Calculate smell factor β for all files
Calculate graph of Caller files
Calculate Test Risk for all files
Calculate Importance for each file
Calculate Project Risk for each file
Dispay file names ordered by risk

The activity diagram above conveys the activities that are part of this tool. This diagram's purpose is to represent activities because they indicate the functional details and provide the lower level information that group into tasks which again groups into modules at a higher level. The control flow between each activity and its sequence is well depicted through this diagram.

From the activity diagram, it can be inferred that calculation of Importance and Test Risk for a file can occur parallely but the Project Risk can be calculated only after the above two activities are completed. Also while doing the dependency analysis at server side, more than one thread could be spawned so that dependency analysis is carried out in parallel for more than one file. This is to improve performance when there are a large number of files to be analyzed for risk. The activity diagram above clubs activities that take place both at the client side and server side.

4.6. Partitions-Modules and Classes

Bases on the tasks and the activities within each task, the tool is subdivided into 16 modules as described below:

1) WPF Executive UI Module
2) Communication Module
3) ServerExecutive Module
4) File Manager Module
5) Parser Module
6) Semi-Expression Evaluator module
7) Tokenizer module
8) Type Definition Store module
9) File Dependency Store module
10) Graph Module
11) Code Metric Analyzer module
12) Risk Analyzer module
13) Importance Analyzer module
14) Test Risk Analyzer module
15) Project Risk Analyzer module
16) Risk Info Store module

Each module belongs to either Presentation layer, Communication Layer, Data storage layer or Processing layer. The responsibilities of each module, dependencies between them and the contained classes are described in detail below:

4.6.1 WPF Executive UI Module

This module is the Executive User Interface module responsible for accepting input from the User and displaying output to the user. The user is initially presented with a directory view from which a set of files are selected by the user for analysis. Once the user selects a set of files, they are analyzed by the tool and the final output would be the file names displayed to the user ordered by risk. This output lets the user understand the risk associated with each component and thus helps in deciding where to focus testing and code improvements.

The input/output is presented in a Graphical user interface based on Windows Presentation Foundation (WPF). This module is a client-side module.

Class and interfaces used:

1) SRAInput – Responsible for accepting input from the user and send it to the Communication module
2) SRAOutput – Responsible for displaying output on screen by obtaining it from communication module.

4.6.2 Communication Module

This module is responsible for establishing communication between the Tool and Repository. This module retrieves the files from the repository using the file paths it has received from the client machine.

4.6.3 ServerExecutive Module

This module is responsible for co-ordinating all the processing activities of the tool. It receives request information from the Communication module containing information specific to the processing requested and the list of files to be processed. This then invokes the File Manager module, Parser module and Risk Analyzer module for carrying out the Risk analysis.

Classes and interfaces used:
1) ServerExecutive- class that coordinates all processing taking place at the server side.

4.6.4 File Manager Module

This module is responsible for managing the list of source files that needs to be analyzed. This is a server side module. The files selected by the user from the directory view are retrieved and maintained in a list. This module handles errors in case it is unable to open or process some files. It ensures that the file streams are properly closed after they are processed
Classes and interfaces used:
1) CFileProcessor – To process each of the files in the list and send it to parser.
2) IFileProcessor – Contract for the services of the CFileProcessor.

4.6.5 Parser module

The Parser module is a container of rules each of which is a detector for some grammatical construction like class, struct, interface, enum, functions etc., This is written by Dr.Fawcett. The rules contain a set of actions which is executed when a particular rule is detected. For example when a class or interface definition rule is found out, appropriate action is taken.

The parser uses the services of Tokenizer module and Semi-Expression evaluator module for its operation. Besides the existing classes and interfaces, some additional classes are needed to achieve the required functionality.

Classes and interfaces used:

1) Parser – Main class used for parsing. Contains set of rules.
2) CSemiExp – Class that holds a semi-expression
3) CToker – Class that splits a file stream into tokens
4) IRule – An interface contract for grammatical rules.
5) ARule – An abtract class to define rules for some common operations.
6) IAction – An interface contract to specify actions for rule.
7) AAction – an abstract class to define actions for rules.
8) DetectNameSpace – rule for detecting namespaces
9) DetectClass – rule for detecting classes, interfaces, structs and enums
10) DetectFunction – rule for detecting functions
11) DetectScopeChange – rule for detecting scope changes

New classes and interfaces to be used:

12) DetectFileDependency – rule for detecting File dependencies.
13) DetectTypeDefinition – rule for detecting Type definitions
14) StoreTypeDef – action for storing type definitions
15) StoreFileDependency – action for storing File dependencies.

4.6.6 Semi-Expression evaluator module

This module contains the C# semiexpression detection class written by Dr. Fawcett. Semiexpression is a partial C# expression, and is a sequence of tokens that have meaning for code analysis. It is a minimal sequence of tokens that ends with the characters '{', '}', ';', or '\n' if the line in which the newline appears starts with the char '#'.This module uses Tokenizer module for its operation.

Classes used:

1) CSemiExp – Form a partial expression from a set of tokens.

Instances of class CSemiExp class provide facilities for constructing, reading, analyzing and editing semi-expression token sequences.

4.6.7 Tokenizer module

This module contains the C# tokenizer class written by Dr. Fawcett. This is a simple tokenizer that does not combine multi-character operators (such as "<<", "+=", "++") into single tokens. It supports reading words, called tokens, from a string or a file stream. Transitions from alphanumeric to whitespace or punctuator characters and back again, are treated as token boundaries. Quoted strings and comments are returned as single tokens.

Classes used:

1) CToker – to separate tokens from a file stream or string stream.

4.6.8 Type Definition Storage module

This module is responsible for holding the type definition information present in all files after they parsed in the first pass. This information is stored in a Hashtable where the key is the type name and the value is an object containing actual type and the file name which owns it. This module is used by the Type Relationship and File dependency storage module.
<u>Classes and interfaces used:</u>
1) CTypeInfo – A class to store the actual type and the containing file name for each type name identified.
2) StoreTypeDef – A class that maintains a list of CTypeInfo objects along with the type names in a Hashtable.

4.6.9 File dependency storage module

This module is responsible for holding all the file dependency information present in the source files after they are parsed in the second pass. The file dependency is identified by checking each token in the file against the Hashtable and comparing the file names. If the file name in differs then means a type from some other file has been used in the currently parsed file which conveys a dependency. This module interacts with Type Definition Storage module.
<u>Classes and interfaces used:</u>
1) CFileDependency – A class to hold file dependency information for each source file.
2) FileDependencyStore – A class that maintains the file dependencies in a suitable data structure like Hash table.

4.6.10 Graph module

This module contains the implementation of a generic directed graph using an adjacency list structure that does not duplicate nodes. This has been written by Dr.J.Fawcett. This module is essential as the calculation of importance and test risk for each file is based on Jacobi iteration technique and this will be applied by constructing a graph of caller files and called files respectively. The Importance analyzer module and the Test Risk analyzer module calls this module to construct a graph of called files and then to construct a graph of caller files (based on the already constructed graph of called files).
<u>Classes and interfaces used:</u>
1) CsNode<V,E> class – A class representing nodes that hold an instance of a V type and a collection of references to child nodes and instances of an E type for each edge to the child.
2) pair<CsNode<V,E>, E> -A structure holds a child node reference and an instance of the E type for its edge.
3) CsGraph<V,E> - A class that represents directed graphs and provides depth first search via its walk() method
4) Operation<V,E> - A class that provides a doNodeOp that CsGraph<V,E>.walk() applies to every node it encounters and a doEdgeOp that it applies when each edge is traversed.

4.6.11 Code Metric analyzer module

This module is responsible for analyzing the code of each source file and calculating some metric values. In our implementation, calculation of test risk of a file involves calculation of the smell factor of each file which depends on some measured metric values and their boundary values. As already mentioned, in this implementation, the metric values that would be calculated and used would be maximum cyclomatic complexity and number of lines of code in a function. The boundary values would be takes as 10 and 50 respectively. This module would be invoked by the Test risk analyzer module as part of its smell factor calculation which is necessary for calculating test risk of a file.

<u>Classes and interfaces used:</u>
1) CodeMetricAnal – A class that provides operations for analyzing code metric for each file that is passed.

4.6.12 Risk Analyzer module

This module co-ordinate the activities of the three modules – Importance analyzer, Test risk analyzer and Project risk analyzer. It serves as an interface for the ServerExecutive for calculation of Risk and internally manages all other calculations and store the resulting information in Risk Info Store module. The ServerExecutive just invokes this module which takes care of all Risk calculations with the help of other modules and finally the ServerExecutive uses Risk Info Store module to send the result to the Communication module.

<u>Class and interfaces used:</u>
1) RiskAnal – A class that invokes methods for all risk calculations present in other classes, creates a FileRiskInfo object and stores it for display.

4.6.13 Importance Analyzer module

This module performs the task of calculating the Importance of each file based on the importance of all its caller files and the probability of change factor which is always assumed to be 0.1. This module makes use of the graph of Caller files and performs Jacobi iteration on that graph. This module is invoked by the Risk Analyzer module. This module invokes the File Dependency Storage module and Graph module for its operations.

<u>Class and interfaces used:</u>
1) ImportanceAnal – A class that has methods to calculate importance value for each passed in file.

4.6.14 Test Risk Analyzer module

This module performs the task of calculating the test risk of each file based on the test risk of all its called files, the probability of change factor which is always assumed to be 0.1 and the owner file's quality/smell factor. This module makes use of the graph of Called files and performs Jacobi iteration on that graph. This module is invoked by the Risk Analyzer module. This module invokes the Code metric analyzer module for calculation of the quality factor of each file. This module also invokes File Dependency Storage module and Graph module for its operations.

Class and interfaces used:
1) TestRiskAnal – A class that has methods to calculate test risk value for each passed in file.

4.6.15 Project Risk Analyzer module

This module performs the task of calculating the project risk of each file based on the test risk and importance calculated for each file. This module is invoked by the Risk Analyzer module. This module invokes the Risk Info Store which contains the Importance and Test risk values of each file and uses them as a source for calculating the Project risk of each file.

Class and interfaces used:
1) ProjectRiskAnal – A class that has methods to calculate project risk value for each passed in file based on the values of importance and test risk that are already calculated.

4.6.16 Risk Info Store module

This module stores all the risk information associated with each file in a Hashtable or an ArrayList. This module is invoked by the Risk analyzer module for each file after the calculation of importance and test risk values. As a final step, this module is invoked by the ServerExecutive module to send the result of analysis to the client through the Communication module.

Class and interfaces used:
1) FileRiskInfo – A class that stores all the risk information associated with a file. The attributes would be,

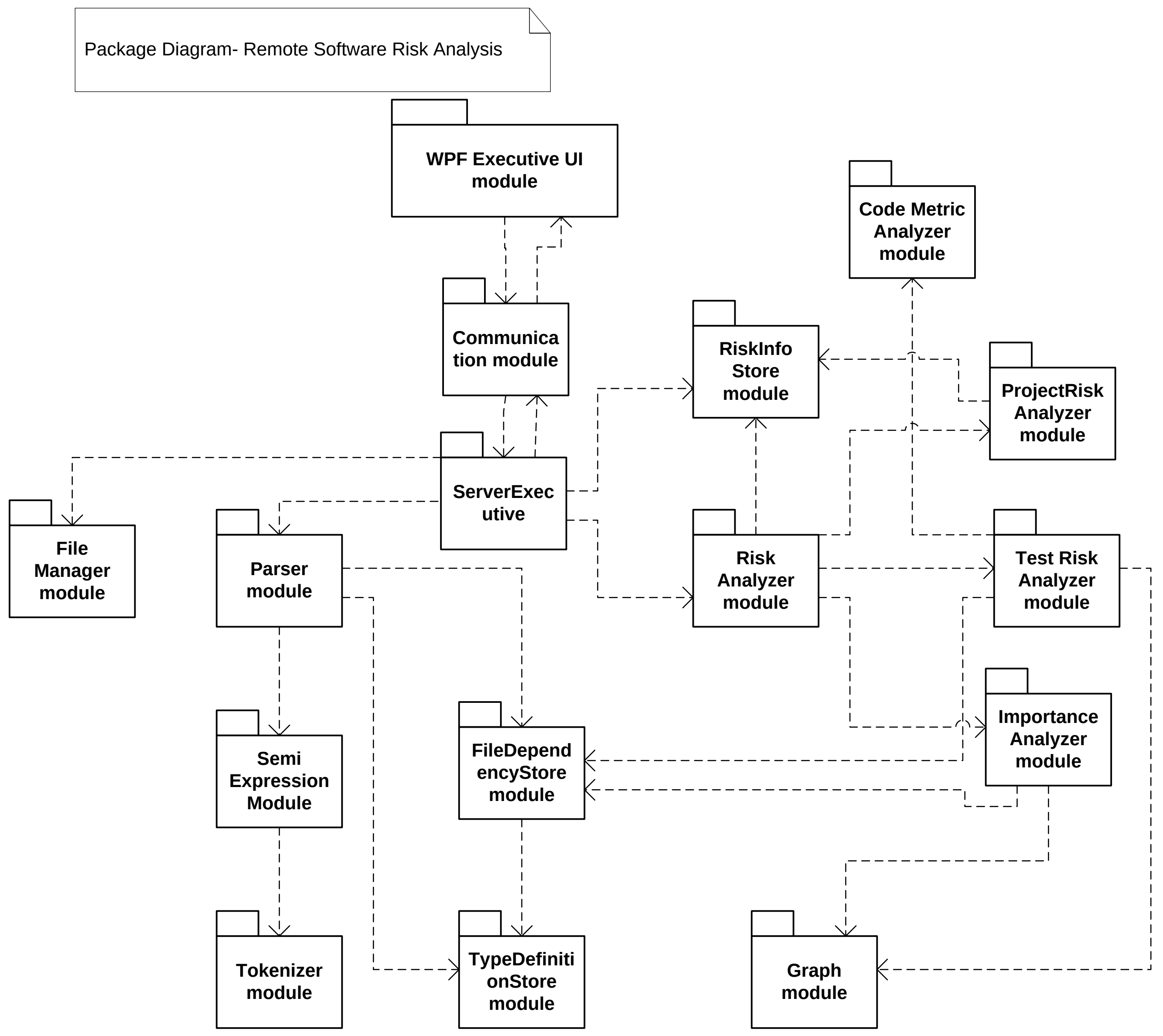
- ➢ importance – stores the importance value
- ➢ testRisk – stores the test risk value
- ➢ smell factor – stores the smell factor/quality factor value
- ➢ project risk – stores the project risk value of that file

2) RiskInfoStore – A class that stores a FileRiskInfo object for each file analyzed in a Hashtable. The key for the Hashtable would be a file name and value would be a FileRiskInfo object corresponding to that file

The above mentioned modules and classes are represented in the following package diagram (Modules are synonymous with packages) and class diagram respectively.

PACKAGE DIAGRAM

The below package diagram gives a high level view of packages and their dependencies:

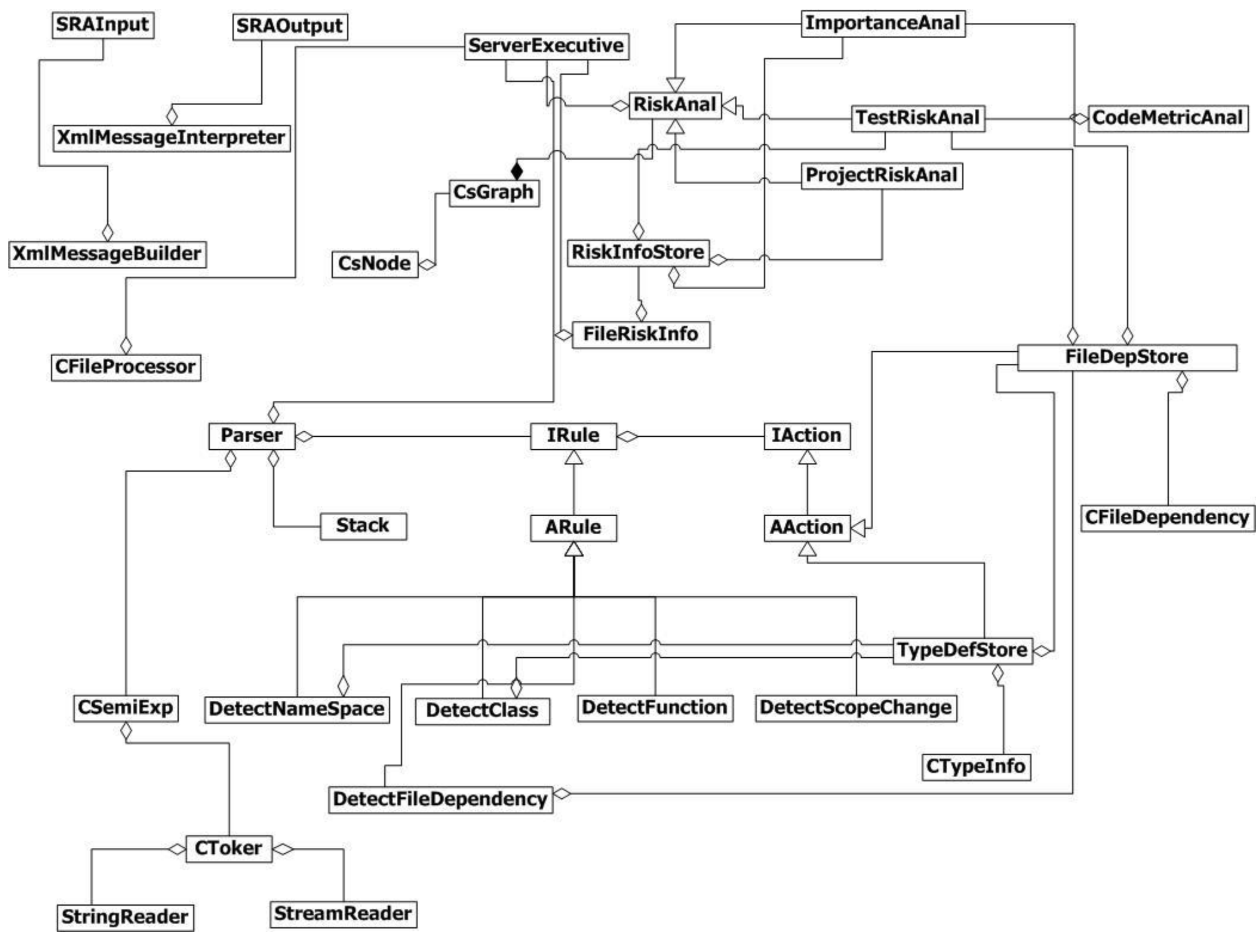

Package Diagram- Remote Software Risk Analysis
WPF Executive UI module
Code Metric Analyzer module
Communication module
RiskInfo Store module
ProjectRisk Analyzer module
ServerExecutive
File Manager module
Parser module
Risk Analyzer module
Test Risk Analyzer module
Importance Analyzer module
Semi Expression Module
FileDependencyStore module
Tokenizer module
TypeDefinitionStore module
Graph module

The package diagram helps in visualizing the system at a high level. Through this diagram, the complexity of the tool could be guessed based on the number of modules and how they interact between them. Always the first step in understanding a new system or tool would be to analyze and understand the package diagrams, which is why this diagram is important and is here. The modules also give a picture of functionality of the entire system or tool.

The modules are placed in one of the layers already discussed. The Communication module is placed in the communications layer. All the modules that deal with processing logic are placed in the processing layer. The WPF Executive UI layer goes into the Presentation layer. The RiskInfo Store module interacts with the data storage layer so that all the risk information is stored persistently for future reference and use.

CLASS DIAGRAM

The below class diagram gives a picture of various classes that constitutes each package.



This further explains modules at a lower level by depicting the various classes that form the whole tool and their relationships. The attributes and operations are suppressed for now as this diagram this just to understand the structure of each module at a higher level. The

finer inner level details would be a part of design and this would be very helpful for the design team to get a nice picture before actually defining attributes and member functions for each class.
The top left portion of the diagram depicts four classes – SRAInput, SRAOutput, XmlMessageBuilder and XmlMessageInterpreter that part of the Client and the communication module. All the other modules in the diagram are on the server side.
A small description about each class and its responsibility has already been provided while discussing the Partitions.

The RiskAnal class holds a reference to the CsGraph class as it needs reference to a Graph object to assist the calculation performed by other modules co-ordinated by this module. The three classes ImportanceAnal, TestRiskAnal and ProjectRiskAnal inherit from the RiskAnal class as RiskAnal class serves as a base interface for all the other three classes that perform risk specific calculations.

4.7. Critical Issues

Any tool that is being developed may have some critical issues that need to be addressed. This tool has some issues which need to be peeked at and are described below.

4.7.1 Dependency between layers

As already discussed, the modular structure of this tool is split into three layers – presentation layer, processing layer and data storage layer, each of them communicating with one another through a communication layer. One of the critical issues here is that each layer depends on one another directly and hence a change in implementation of any one of the layers will cause a change in another layer. This would lead to making changes often and testing the same which is an additional burden on the developers involving more cost and effort.

Solution:

A solution could be to apply the Dependency inversion principle which states that higher level components will not depend on lower level components. Instead both will depend on their abstractions. This means each layer will have an interface that provides a contract for the services provides by the layer and the other layer will use the interface to communicate with that layer. This is useful because when the implementation of layer changes, the depending layer need not change since it depends only on the interface.

Another solution is to use an MVC (Model-View-controller) architecture which is an architectural pattern. This pattern isolates the processing logic from the presentation layer thus allowing independent development, maintenance and testing of each. The model is the domain-specific representation of the data upon which the application operates. The view renders the model into a form suitable for interaction, typically a user interface element. The controller receives input and initiates a response by making calls on model objects.

4.7.2 Concurrent file access

This issue is faced when the client selects a file for analysis and while in the process of sending it to the server, some other process in the client machine is doing some modification to the same file. This results in concurrent file access as the file is being shared in this case and can lead to abnormal behavior or crash of one of the processes.

Solution:

As multiple threads are sharing the same file, some kind of thread locking mechanism should be applied in this scenario. For example a Mutex or a Monitor or some kind of locking variable could be used so that, access to the file is synchronized which means only one thread could access the file at a particular time.

4.7.3 Performance Vs Load

Performance is a critical issue that needs to be addressed with this tool since a large amount of data might possibily be sent across the network. Also the server may have to deal with multiple clients or a large number of files from a single client.

Solution:

To handle the former case, multiple servers can be used so that during heavier load times, another server could serve some of the client requests waiting in the queue. Even then each server will have a threshold on number of client requests that it could handle at a time. If the number exceeds that threshold, the requests are simply ignored and the requests need to be sent again possibily after some time. To handle the latter case, it would be a better idea to split the file into chunks of data and send it to the server so that huge data is not sent through the network. This may consume network resources at a higher cost but may improve performance to some extent.

4.7.4 Notification of newly added files/changed files to other users

This is another major issue that needs to be addressed. When there are multiple teams working on a project, each will have their own set of source code files. When a developer creates a new file in his local machine, there must be some means to notify other teams, especially users like Quality Analyst, so that they can do a risk analysis on the newly added file. Without a notification, the other teams and users who are part of the project would be unaware of that file which may lead to confusions at a later point of time.

Solution:

The developer should take the responsibility to send a notification to other developers and other users like manager, Quality analyst and test team. This is to ensure that all the people who are part of the project are aware of the newly added file and the Quality analyst makes sure that it doesn't create any impact on other files by doing a risk analysis. The notification could be through an e-mail or some other means of communication that reaches all the members and users of the project.

4.7.5 Security

Security is an important issue to be considered when it comes to sending data across network. As this system involves sending source files across the network, it is very important to make it secure to prevent hacking.

Solution:

An encryption mechanism could be used while sending the data so that, it is safe during its travel through the network. But encryption will again involve more cost and maintenance effort but since the source code files being sent are critical data, it is very essential to encrypt and send them.

4.7.6 Performance for large number of files

When there a large number of files to be analyzed, it becomes a burden on the server to process all the files, do risk analysis within a short span of time. This can't be achieved when the server processes one file at a time. The client may have to wait an indefinite amount of time in this case.

Solution:

The server spawns more than one thread and the files are processed parallely. The number of threads may be decided based on time taken and complexity. As the files are processed parallely, the intermediate results are stored in some queue and are merged again for further processing (if there is a second level processing). In this system, one pass must be made for type definition identification and another pass for type dependency analysis. Also the files are parsed again for calculating importance and test risk. As multiple passes are required, it is a good idea to make the server use more than one thread and process them parallely.

4.7.7 Risk analysis for the entire project when files are distributed

When the source code files are distributed across the client network, it is not possible to analyze the risk for all the files without assembling them in a single machine. This is a typical scenario where the Project manager or Quality analyst would want to analyze the risk of all the files in the project.

Solution:

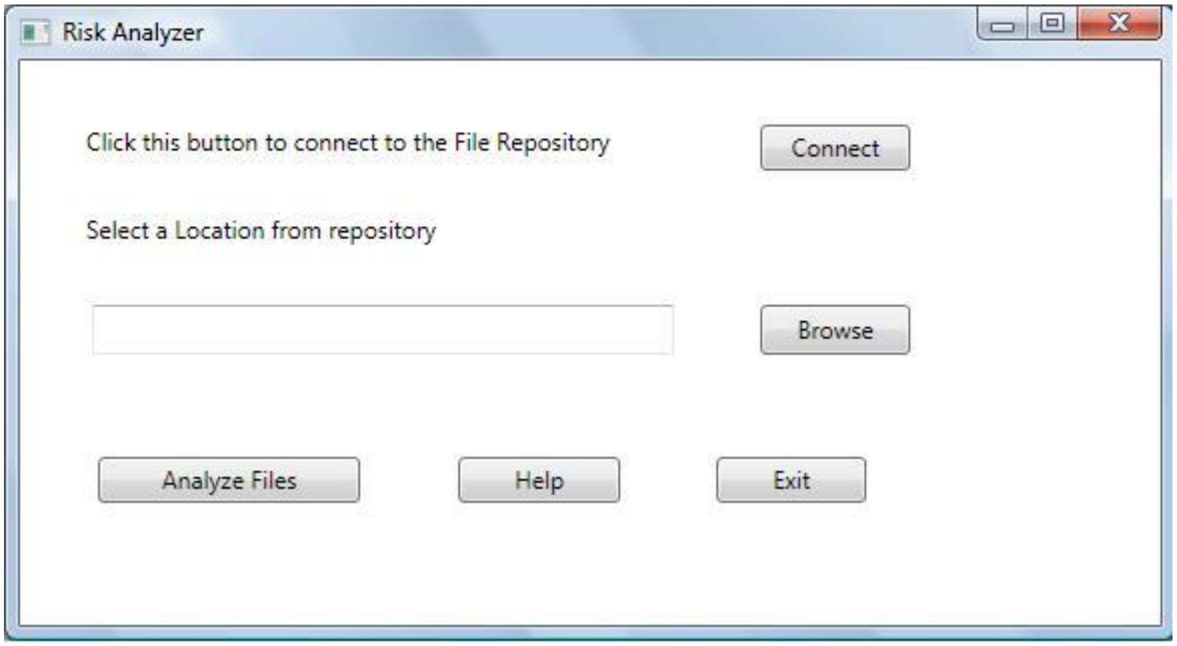
The Manager or the Quality analyst should manually gather all the source code files from developers into their machines and then do a risk analysis of all the files as a whole. As this introduces additional burden for these users, another possible solution would be to keep all the source code files in a single server, in a repository. The developers working on the code should work on the copy in the repository and should make sure that it is up to date. This allows a quality analyst or a manager to access all the files from a single location and do a risk analysis.

A problem noted with having a single repository for all files is that, when a developer wants to do risk analysis of certain files, the entire file repository has to be brought to the client from the server which would cause a lot of traffic in the network. Also the developer would not be interested in other team's code and may face difficulties in identifying his files for selection from a whole bunch of files. Hence it is better to have the files distributed and assemble them when needed by certain users at some point of time.

4.8. Views

This section describes the appearance of the Risk analyzer to its users and designers. The proposed WPF based graphical user interface is depicted in the following screenshots below:

4.8.1 Obtaining Input and Analyzing files


Risk Analyzer
Click this button to connect to the File Repository
Connect
Select a Location from repository
Browse
Analyze Files
Help
Exit

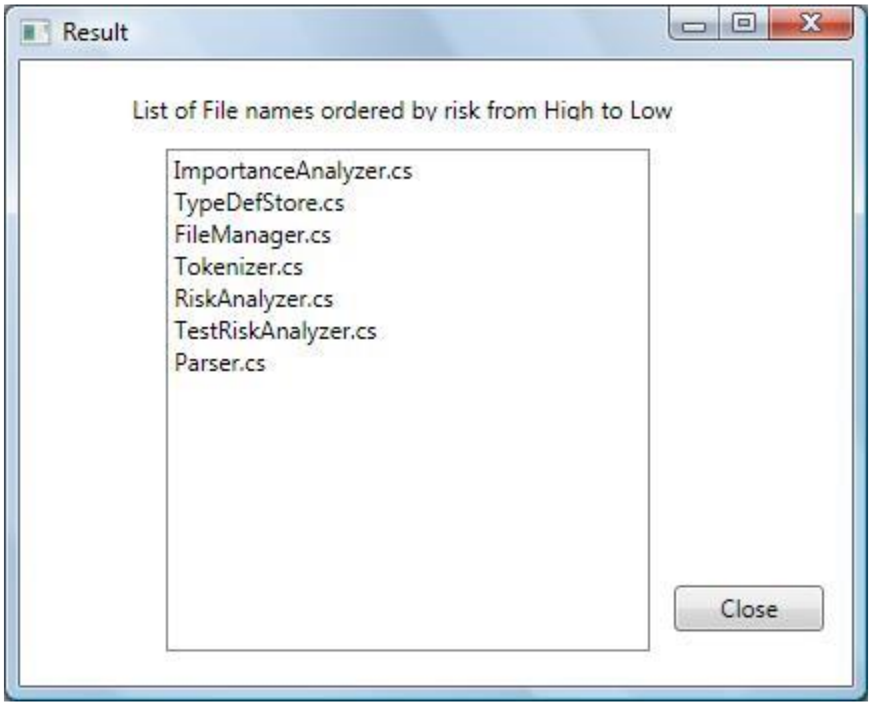
1) The User initially connects to the file repository

2) After this the user can select a remote location from the repository server that contains files for which risk need to be analyzed.

3) After selecting a remote location, he/she can click on Analyze Files button to deploy the tool and start analysis.

4) Help button gives a brief description about the tool for the user and Exit button closes the application.

4.8.2 Showing output – list of file names ordered by risk



List of File names ordered by risk from High to Low

```
ImportanceAnalyzer.cs
TypeDefStore.cs
FileManager.cs
Tokenizer.cs
RiskAnalyzer.cs
TestRiskAnalyzer.cs
Parser.cs
```

The output displayed in a new window just displays a lisbox containing all the names of files sent by the user but now sorted based on the risk information after analysis. The file which has the highest risk is at the top of the list followed by other lower level risk files. This output is simple and enables a user to understand which file has more importance and needs attention in terms of risk. The user can close the Result window by clicking on Close button.

5. Rule-Based Defect Analyzer Tool

5.1 Introduction

Defect Analysis, is done by the development team usually, in order to decide what defects should be treated, fixed, rejected or deferred to be dealt with later. Analyzing defects in the source code and deciding on how to treat the defects is an integral part of a software development. It is a continuous process that helps in achieving the desired quality for the final product. This tool helps in detecting defects based on predefined rule sets by running each of them against the source code. A software project should be simple and maintainable which is part of its design goals. This tool ensures that the source code which is analyzed is simple and maintainable by checking it against some rules and reporting the results accordingly. This tool also performs a quality analysis of individual files that quantify the important code smells defined.

5.1.1 Objective and key idea

Its main objective is to perform analysis of defects, based on existing pre-defined rules, of a set of source files defined by the user. This helps in evaluating the quality of a code baseline. Its objective also includes quantification of files by analyzing their code smells (quality factor). This tool does a batch mode quality analysis so that there is no human intervention. The key idea is to use a parser by building it using a set of rules for analyzing defects and running it on the set of input files and finally display the defects to the user.

5.1.2 Obligations

The tool will be developed as an efficient and user-friendly one meeting all the stated requirements. The main obligations of the tool would be

- ➢ Perform a rule based defect analysis of the project files and report rule violations for each file.
- ➢ Support batch mode quality analysis of individual files that quantify the important code smells defined in class.

5.1.3 Organizing Principles

The usage of Parser structure which allows plugging-in rules and actions as needed by the functionality is an organizing principle.

5.2 Uses

The following are considered as the uses of this tool:

1) Analyses code based on some rules and reports rule violations. This ensures that are source code are consistent in that manner. For example, some of the rules against which the source code should be checked are:

- Presence of manual and maintenance page
- Presence of test stubs

2) Apart from code consistency this tool helps in keeping the source code maintainable by detecting presence of manual and maintenance pages which is very essential in order to understand the code at a higher level and know other details.

3) As this tool involves analyzing code smells of each file, it thereby reports some of the quality factors/smells like the maximum cyclomatic complexity of the functions and maximum number of lines in the functions present in the file. There are many other code smells defined but these two are considered very importance and appropriate to our scope of analysis. By reporting these information, a developer can understand the complexity of his code and hence can take steps to make it simple.

4) To make a code selectable, there must be a way to easily run the code individually as a component. This is possible if the code has a test stub. This tool helps in reporting if the source code has proper test stubs.

5) When common defects are defined as rules and its violations are reported by this tool, it really helps in reducing the occurrence of common defects and hence the source code would be less prone to frequently/commonly seen defects

5.3 Activities

The tool performs two types of tasks:

1) Rule-based defect analysis

2) Batch-Mode Quality Analysis

The major activities that are part of the two tasks are discussed separately.

The activities of a Rule-based defect analysis and described and depicted below:

1) Show an interface for the user to select filesets

A directory view of source files is presented to the client. The client then navigates through the files and selects a set of files that need to be analyzed for defects based on predefined rules. The user can select new files or select existing filesets.

2) Get the fileset containing file paths in the code repository and retrieve files

The tool gets the file paths in the input fileset and not the actual files. The actual source code is in a code repository and hence the tool retrieves the files from the code repository.

3) Perform rule-based defect analysis.

The tool uses Parser that defines rules for some defects. Some of the rules that could be applied by this tool are

1) Detecting presence of manual and maintenance pages
2) Detecting presence of test stubs.

The first rule is applied to the source code by checking for a block of comments whose length is greater than 20 and containing keywords like Maintenance, Module Operations etc. ,When a source code doesn't satisfy these rules, it means it violates the rule and is displayed to the user as a rule violation for that particular file.

Similarly for the second rule, the source code is checked if it contains #if directive and a main method. If not a rule violation for that file is shown.

Some more rules could be applied so that they are checked for individual classes in each file (assuming there may be more than one class in a file sometimes).

4) Store the rule violations

The rule violations for each file are stored in an appropriate data structure. This is done in order to send back the results for all the files together to the client. The data structure might need to be modified to store rule violations for each class in a particular file when such a rule is applied.

5) Send the results

The results of rule violations are sent back to the module from where it obtained its input.

The activities of a Batch-mode Quality analysis tasks and described and depicted below.

Batch mode quality analysis essentially means the user can schedule the tool to be run at some time and it performs its operations without human intervention.

1) Show an interface for the user to schedule quality analysis

The user is presented with an interface from where he can define a new fileset or select an existing fileset. The user has to select a date and time so as to schedule the analysis.

2) Get the fileset containing file paths in the code repository and retrieve files

The tool gets the file paths in the input fileset and not the actual files. The actual source code is in a code repository and hence the tool retrieves the files from the code repository.

3) Perform Quality analysis when scheduled.

Quality analysis is used to quantify some of the code smells in the source code. In this context, we shall discuss code smells like number of lines in a function and cyclomatic complexity of a function. Both these code smells are used to ensure that the code is simple and maintainable. The tool displays all the functions with the corresponding number of lines and cyclomatic complexity. This helps the developer to restrict these measures under a certain threshold.

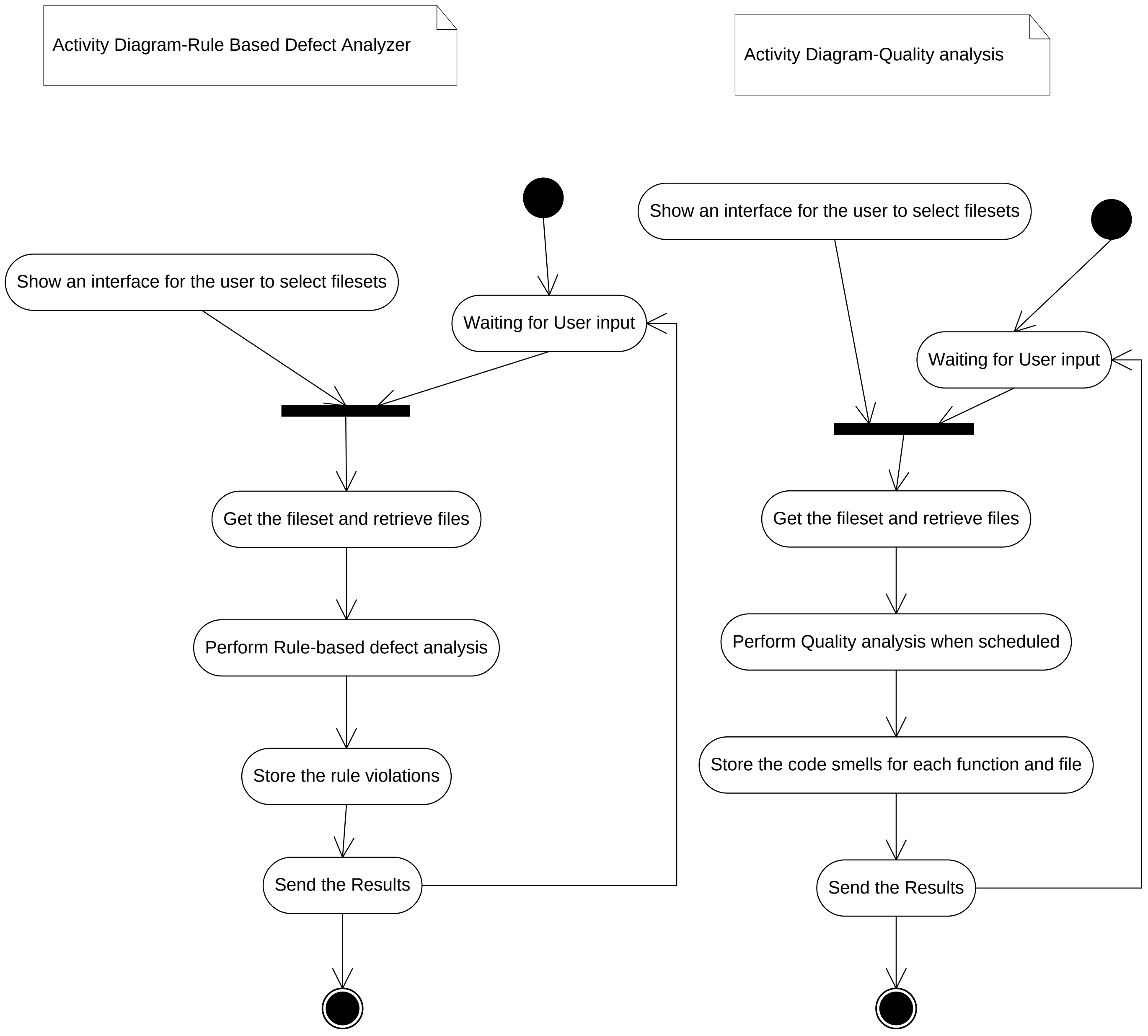
The number of lines of a function is calculated by keeping track of the number of lines when a function scope is entered and exited. Scope is identified by using a scope stack which keeps track of type, function and local scopes. Similarly within the scope of a function, the cyclomatic complexity is calculated by checking the number of control blocks and loop structures.

4) Store the code smells for each function and each file

The code smells calculated as above for each function is appropriately stored in a data structure and this information is stored for each file. A map would an ideal data structure to store this information which might need to contain another map as its value. The outer map contains a filename as its key and a map as it value. The value map contains function name as key and code smells as value. In case of multiple code smells, all of them could be encapsulated in a well-defined object.

5) Send the results

The results are logged into a text file and then sent back to the requested module.


Activity Diagram-Rule Based Defect Analyzer
Show an interface for the user to select filesets
Waiting for User input
Get the fileset and retrieve files
Perform Rule-based defect analysis
Store the rule violations
Send the Results
Activity Diagram-Quality analysis
Show an interface for the user to select filesets
Waiting for User input
Get the fileset and retrieve files
Perform Quality analysis when scheduled
Store the code smells for each function and file
Send the Results

5.4 Partition

Bases on the tasks and the activities within each task, the tool is subdivided into 16 modules as described below:

1) Executive UI Module
2) File Manager Module
3) Parser Module
4) Semi-Expression Evaluator module
5) Tokenizer module
6) Code Metric Analyzer module
7) Defect Analyzer module
8) Defect Store module
9) Code Metric Store module

5.4.1 Executive UI Module

This module is the Executive User Interface module responsible for accepting input from the User and displaying output to the user. The user is initially presented with a directory view from which a set of files are selected by the user for analysis. Once the user selects a set of files, they are analyzed by the tool and the final output would be displayed. This output displays all the rule violations for each file and for the quality analysis scheduled to run at some time, the results are sent back after the tool is run and the user sees the cyclomatic complexity and number of lines for each function. The tool notifies the user of functions that exceed the cyclomatic complexity or number of lines threshold at the end with a warning.

The input/output is presented in a Graphical user interface based on Windows Presentation Foundation (WPF). This module is a client-side module.

5.4.2 File Manager Module

This module is responsible for managing the list of source files that needs to be analyzed. The files selected by the user from the directory view are retrieved and maintained in a list. This module handles errors in case it is unable to open or process some files. It ensures that the file streams are properly closed after they are processed
<u>Classes and interfaces used:</u>
1) CFileProcessor – To process each of the files in the list and send it to parser.
2) IFileProcessor – Contract for the services of the CFileProcessor.

5.4.3 Parser module

The Parser module is a container of rules each of which is a detector for some grammatical construction like class, struct, interface, enum, functions etc., This is written by Dr.Fawcett. The rules contain a set of actions which is executed when a particular rule is detected. For example when a class or interface definition rule is found out, appropriate action is taken. The parser uses the services of Tokenizer module and Semi-Expression evaluator module for its operation. Besides the existing classes and interfaces, some additional classes are needed to achieve the required functionality.

5.4.4 Semi-Expression evaluator module

This module contains the C# semiexpression detection class written by Dr. Fawcett. Semiexpression is a partial C# expression, and is a sequence of tokens that have meaning for code analysis. It is a minimal sequence of tokens that ends with the characters '{', '}', ';', or '\n' if the line in which the newline appears starts with the char '#'.This module uses Tokenizer module for its operation.

5.4.5 Tokenizer module

This module contains the C# tokenizer class written by Dr. Fawcett. This is a simple tokenizer that does not combine multi-character operators (such as "<<", "+=", "++") into single tokens. It supports reading words, called tokens, from a string or a file stream. Transitions from alphanumeric to whitespace or punctuator characters and back again, are treated as token boundaries. Quoted strings and comments are returned as single tokens.

5.4.6 Code Metric analyzer module

This module is responsible for analyzing the code of each source file and calculating some metric values. This module is invoked by the Executive module. As already mentioned, in this implementation, the metric values that would be calculated and used would be maximum cyclomatic complexity and number of lines of code in a function. The boundary values would be takes as 10 and 50 respectively.

5.4.7 Code Metric Store module

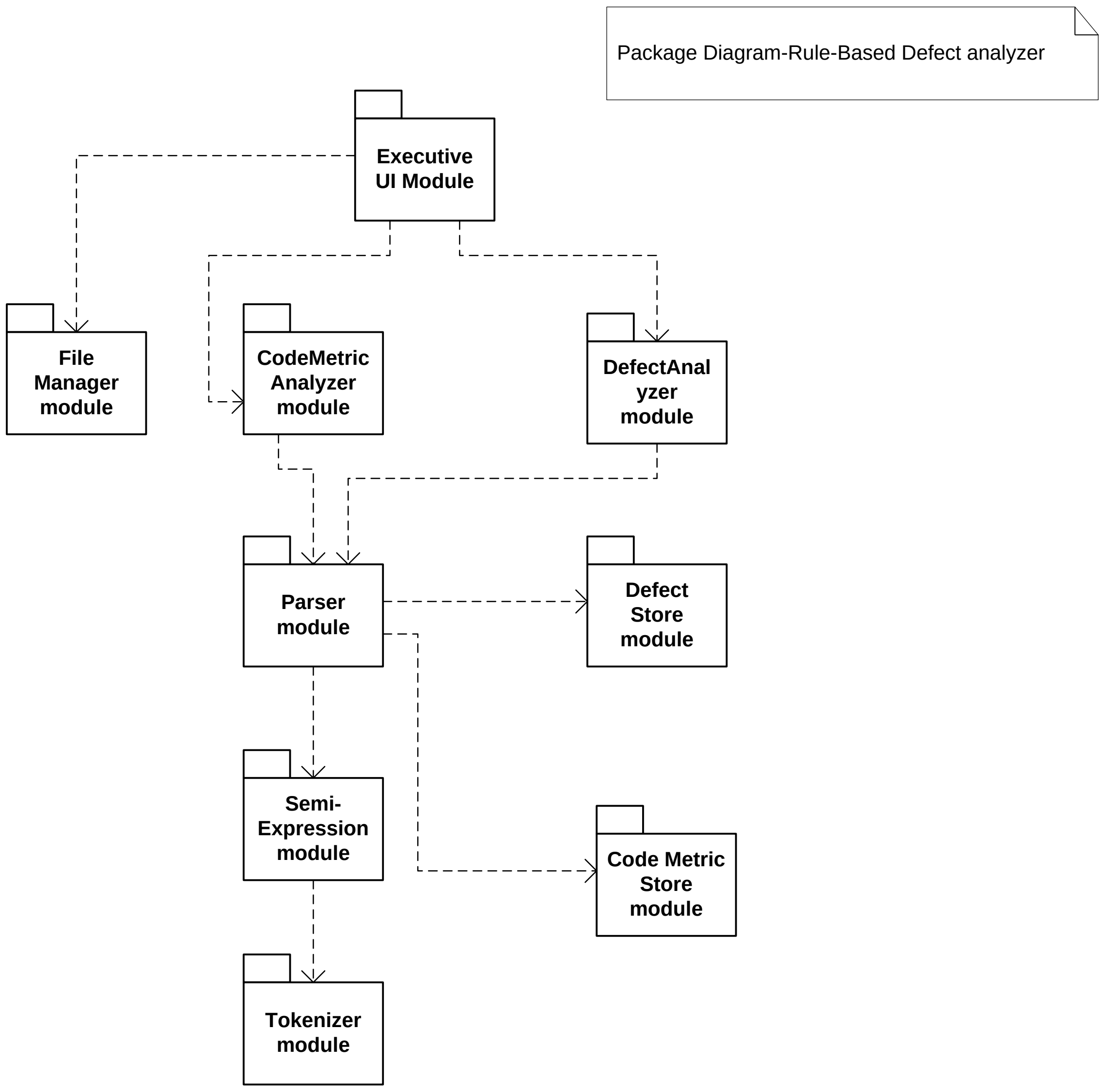
This module is invoked by the Code Metric analyzer module to store the code smell information. It stores the code smells for each function present in the set of files. This module stores functions that exceed the cyclomatic complexity and number of lines threshold separately to give a warning message to the user.

5.4.8 Defect analyzer module

This module is responsible for analyzing all the files for defects based on predefined rules by using a Parser. It configures the parser using some rules and invokes Parser module to carry out the defect analysis. This module is invoked by the Executive module.

5.4.9 Defect Store module

This module is responsible for storing the rule violations for each file that is analyzed. This module is invoked by the Parser module after analyzing the files and sends the results to them.


Package Diagram-Rule-Based Defect analyzer
Executive UI Module
File Manager module
CodeMetric Analyzer module
DefectAnalyzer module
Parser module
Defect Store module
Semi-Expression module
Code Metric Store module
Tokenizer module

5.5 Critical issues

1) Detecting Code smells

For batch mode quality analysis, detection of code smells is essential. There can be many code smells associated with a file. The document describes detecting two code smells which are number of lines in a function and cyclomatic complexity. Although these two are important metrics, there could be many other smell factors that affect the quality of a file. Some of them are *Long Parameter list, Duplicated code, temporary Field* etc. Hence deciding which code smell to detect is an important issue as a decision made on the quality depends on the factors which we detect.

<u>Solution:</u>

Some other important code smells should be considered which are considered essential to make a decision on the quality of a file. The code smell can be chosen based on how far it impacts the complexity and structure of code and performance.

2) Performance for large number of files

This is an issue for all tools that deals with input files. When a large number of files are analyzed, it may take a long time for computation and results to be sent back by the user. The burden on the server increases.

<u>Solution:</u>

The server spawns more than one thread and the files are processed parallely. As the files are processed parallely, the intermediate results are stored in some queue and are merged again for further processing (if there is a second level processing)

3) Analyzing file that has not changed

There might be many files which are not changed since it was last analyzed but the user sends it for analysis again. This consumes unwanted time for transporting that file and analysis.

<u>Solution:</u>

Some measures could be taken to make the code repository not return files which have not been changed since a very long time. The code repository keeps track of various revisions of files and hence can incorporate some mechanism to achieve the same.

4) Batch mode –how to execute? How to control?

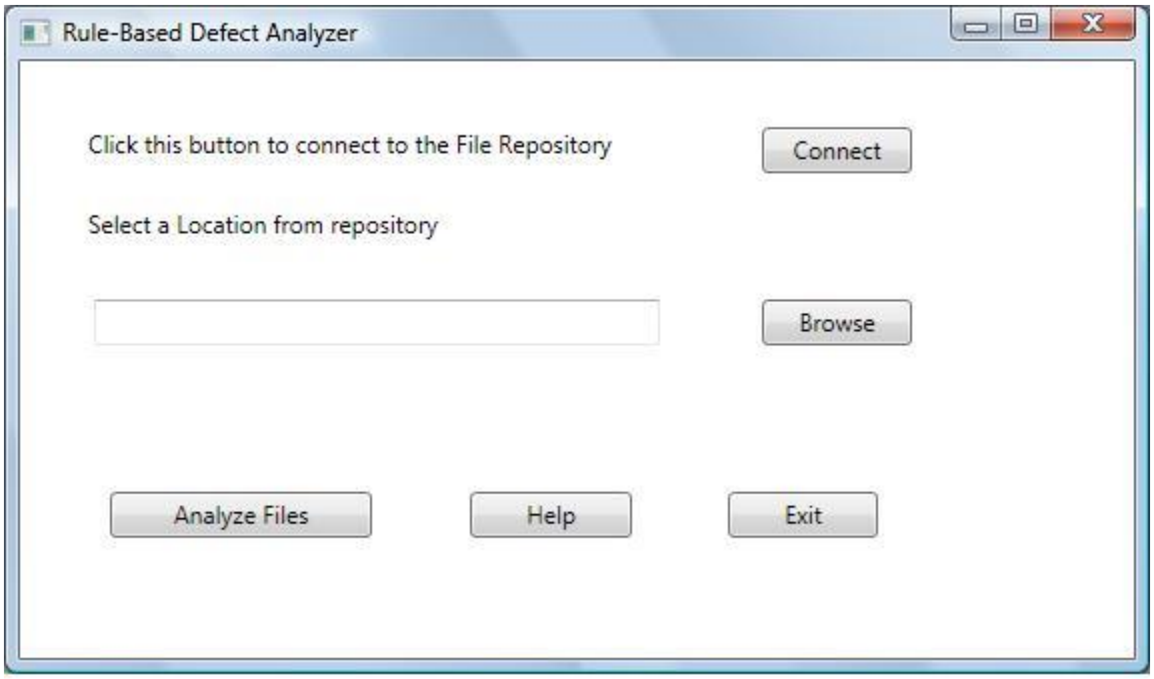
The quality analysis has to be done without human intervention and can be scheduled to be run at any date and time. How to achieve this and how to control this analysis is an important issue that needs to be addressed.

Solution:

This is similar to the working of an agent program which conducts scheduled QA analysis. A specific date and time and an input file set could be configured in an XML file for the quality analysis and a job or a batch file could be created to launch the quality analysis with specified input on the scheduled date and time. A simple tool could be developed to generate the xml and make the process easier.

5.6 Views

This section describes the appearance of the Defect analyzer to its users and designers. The proposed WPF based graphical user interface is depicted in the following screenshots below:


Rule-Based Defect Analyzer
Click this button to connect to the File Repository
Connect
Select a Location from repository
Browse
Analyze Files
Help
Exit

1) The User initially connects to the file repository to select a remote location using connect button.

2) The user then selects a remote location.

3) After this, the user can click on Analyze Files to start analysis of files by the tool.

4) Help displays a brief description of the tool and Exit closes the application.

5) The Results would possibly be displayed in a new window.

6. Tracking Tools

6.1 Introduction

Tracking tools like requirements database, bug tracker and change logger are essential during the development of large software project. A bug tracker is a tool designed to help quality assurance and to keep track of reported software bugs induced by various teams.

Change logger is a tool to maintain changes of source code by various team members which is very essential. Requirements database is a tool to store the requirements of a project gathered from customer. As requirements frequently change and new requirements are often put forth by customers, this tool would be very helpful to track and check if the requirements are met.

6.1.1 Objective and key idea

The main objective of the tracking tools is to track the progress of the project so that at any point of time it is possible and convenient to capture the state of the project which might be very useful for the Project managers to give updates to customers.

6.1.2 Obligations

The tools will be developed as an efficient and user-friendly one meeting all the stated requirements. The main obligation of the tools would be

Bug tracker:

- Keep track of above mentioned information at any time
- Provide an efficient way to input all the above details

Change logger:

- Keep track of changes with associated information
- Provide and easy and efficient interface

Requirements database:

- Keep track of requirements along with the status and other information
- Provide and easy and efficient interface for logging requirements

6.2 Uses

Bug tracker:

A bug tracker tool may be used to for the following:

- Ensure that defects doesn't get lost or stolen.
- Get an idea about the state/progress of the product
- Setting milestones
- Accepting feature requests
- Detect the potential of testers and developers
- History of bugs that can be referred later at any point of time
- Single source for all communication about bugs

A bug tracker may include details like the following:

- Time when the bug was logged
- Severity – Related to complexity of fixing the bug
- How to reproduce the bug
- File name in which bug occurs
- Error behavior or Bug Description
- Person who logged the defect
- Person working on the solution
- Status

This tool provides a centralized view of bugs and their status. It provides valuable inputs after testing phase and during the production phase. This may also be used to generate reports on the productivity of programmers at fixing bugs. There is a possibility to identify a pattern in the bugs that suggests a broader fix than just patching the individual bugs. Refactoring might be indicated with the help of this tool.

Change logger:

The purpose in implementing a change logger tool is to ensure that the project is meeting critical success factors, help avoid potential delays, prevent higher costs and achieve its results sooner. Logging changes enhances project planning and serves great when working within a team of large members who are working on the same source code.

A change logger might contain the following information:

- File name
- Version
- Change description

This helps in building up a maintenance history which is very essential during maintenance of the project or when a new developer wants to make changes on an existing project.

Maintaining the order of changes in a source code file induced by various team members is an integral part of project management and software development.

Requirements database:

Requirements are desired characteristics of a product being developed. They are usually in the form of a formal statement of a function to be performed. This tool is used to gather, store, track, prioritize and implement requirements. As requirements are prone to be frequently changed by certain customer and there is a very high possibility of introduction of new requirements in any project, this tool saves valuable time by storing and tracking each requirement. A project manager can use this tool to ensure that before the final delivery of the project, all the requirements or specifications are met.

The following are some of the key uses of this tool.

- This tool enables you to gather "structured" requirements. This basically means you can define attributes you'd like to track for each requirement (such as Requestor, Needed Date, Owner, etc) - and then make sure each requirement has those attributes.
- This tool can save a ton of time when it comes to managing requirements, as they automate a lot of requirements management tasks like creating requirements documents.
- Most Project Managers who're responsible for gathering and/or tracking requirements will gain a lot of advantage as this tool can eliminate a lot of the unnecessary stress associated with this process.
- This tool will help in making the products and projects more successful.
- This tool will enable in collaborating with internal and external stakeholders efficiently & effectively.

Essentially a code repository can act as a version management system managing various versions of files but this tool is an additional resource for the various actors involved.

6.3 Activities

As all the three are tracking tools, they perform very similar activities except that the data they deal with differs with each one of them. Requirements database deals with requirements and few other details associated with it, Bug tracking deals with files and bugs related to it, Change logger deals with files and changes related to it. The activities for all the three tools are laid out in a general way in this section and wherever there is a difference, it is explained.

There are two possibilities. The user can give an input to be stored in the Database or can retrieve data from database as an output.

6.3.1 Get the input from user

The user enters his input in the user interface based on the tool.

Requirements database – Enters details like Requirement, component it belongs to, developer working on that and its status. There could be some additional details that can be logged.

Bug tracker – Enters details like Time when the bug was logged, Severity, How to reproduce the bug, File name in which bug occurs, Error behavior, Person who logged the defect, Person working on the solution, Status

Change logger – Enters details like Filename, version and change description.

6.3.2 Create an object based on the tool

All the input details are collected into an object and then sent to another module to be stored in the database. The input details are stored as properties of that object. When the tool used is requirements database, an object called RequirementInfo is created with the input details. When the tool used is Bug tracker, a BugInfo object is created and when the tool used is change logger a ChangeInfo object is used.

6.3.3 Send the object to Data Storer

The created object with all the input details is sent to the DataStorer module.

6.3.4 Store the details in the Database

The Data storer module takes care of extracting the details from the object and sends it to the database. An external database is used to store these details.

6.3.5 Show confirmation message for user

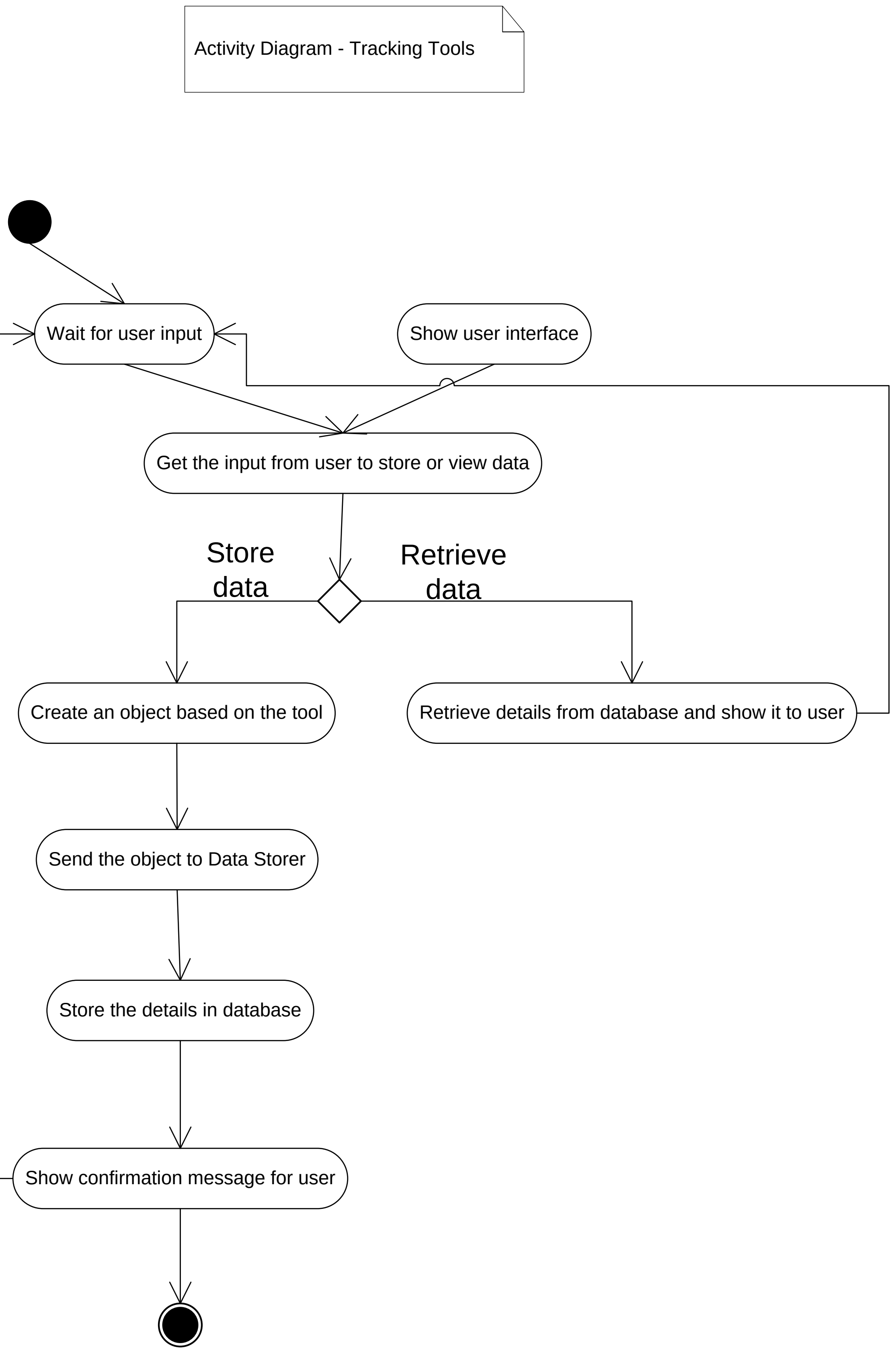
When the data is stored successfully in the database, the input module displays a message to the user denoting the same.

6.3.6 User gives an input in order to view details from database

The user can also view the details from the database apart from storing details. In this case he/she searches the database based on some of the input details they had used while storing them.

6.3.7 Retrieve details from database and show it to user

The request for data is passed on to a Data Retriever module which extracts data from the database and sends it to the input module to be shown on screen.


Activity Diagram - Tracking Tools
Wait for user input
Show user interface
Get the input from user to store or view data
Store data
Retrieve data
Create an object based on the tool
Retrieve details from database and show it to user
Send the object to Data Storer
Store the details in database
Show confirmation message for user

6.4 Partition

Bases on the tasks and the activities within each task, there are 7 modules considering the tracking tools as a single unit.

1) Input module
2) Requirement Info module
3) Bug Info module
4) ChangeInfo module
5) Data Storer module
6) Data Retriever module

The responsibilities of each module, dependencies between them and the contained classes are described in detail below:

6.4.1 Input module:

This module is responsible for accepting user input. The user can either provide input details to store in database or search existing data in the database to view the same. The interface for each tool depends on the details appropriate to that tool.

6.4.2 Requirement Info module

This module defines an object that stores all attributes related to requirements to be stored in the requirements database. It contains details like the requirement description, status, component that it belongs to etc., There might be other interesting attributes in this module. This module is invoked by the input module after receiving input from Requirements database tool.

6.4.3 Bug Info module

This module defines an object that stores all attributes related to bugs that need to be stored in the database. The details that would be stored as part of a bug tracker tool has already been mentioned earlier. This module is invoked by the input module after receiving input from Bug tracker tool.

6.4.4 ChangeInfo module

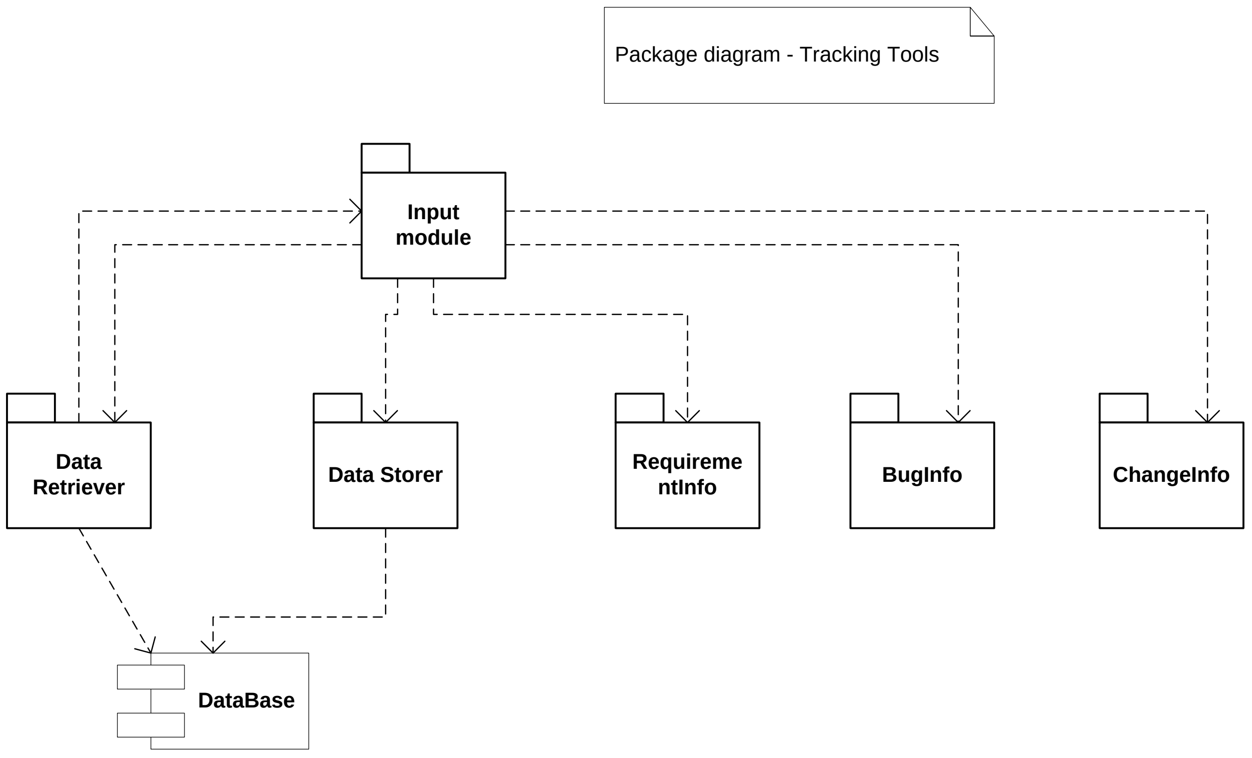
This module defines an object that stores all attributes related to a change logger tool. The details that would be stored as part of a change logger tool has already been mentioned earlier. This module is invoked by the input module after receiving input from Change logger tool.

6.4.5 Data Storer module

This module is an important module that takes an appropriate object based on the tool and stores all the details in an external database after extracting from that object. It establishes database connection and performs transactions with the database. When the transaction is successful, it sends a confirmation message to the input module.

6.4.6 Data Retriever module

This module is another important module that takes a request containing some attributes belonging to a particular tool based on which a search should be made in the database.. It establishes database connection and performs transactions with the database. It retrieves data and send it back to the input module to be displayed on screen.



6.5 Critical issues

6.5.1 Maintenance / Management

Maintenance and Management of these tracking tools is an important issue. In case the database is down, it must be notified, and some person must take action accordingly.

Solution:

A single person like the Manager of the project or a single team can be made responsible for managing these tools and database. For example, the bug tracking tool can be managed by the testing team, since they would be using the tool most often. Similarly managers would be using the requirements database frequently and hence can be made responsible. For database, a person who has enough knowledge on database administration should be responsible for managing that. This will ensure proper management and defect free environment.

6.5.2 Efficient Retrieval of data

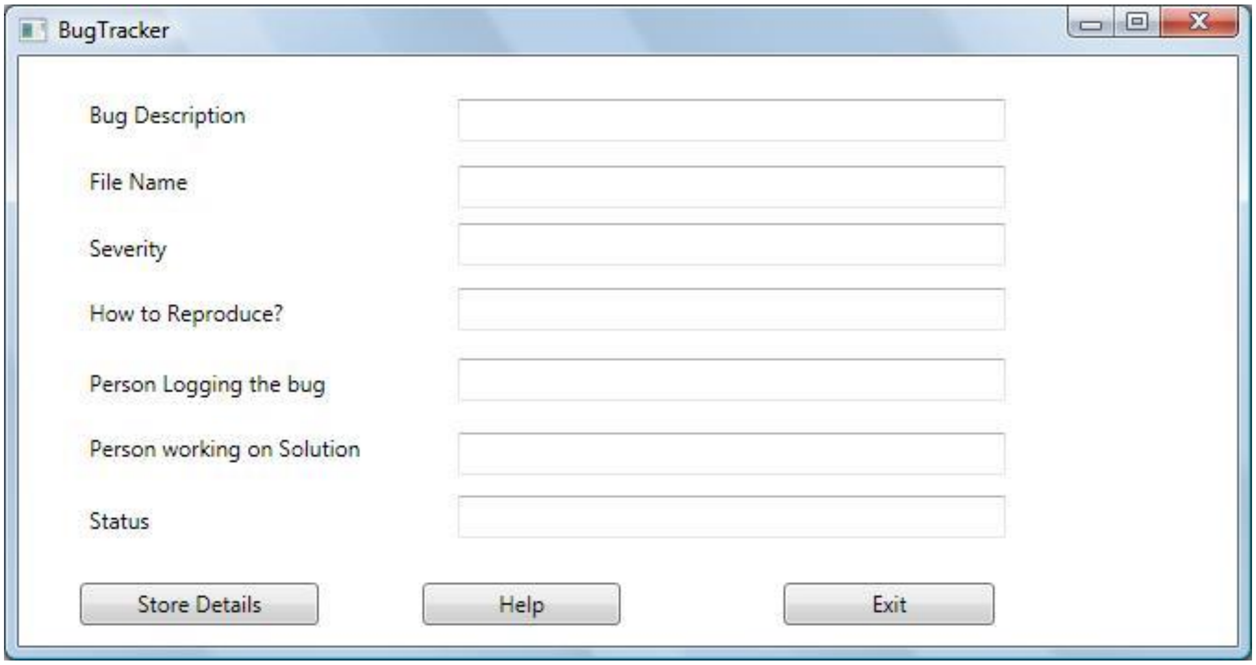
As data is stored and retrieved from database, efficient retrieval of data is an important issue to be addressed since it affects the performance of the tool

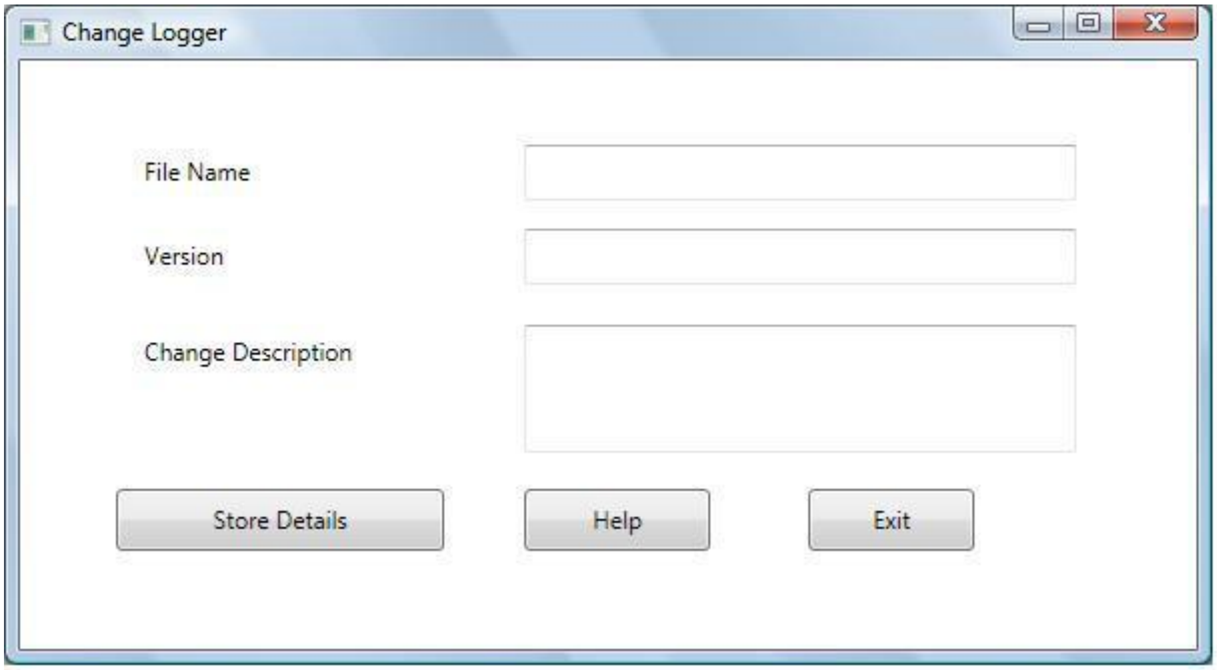
Solution:

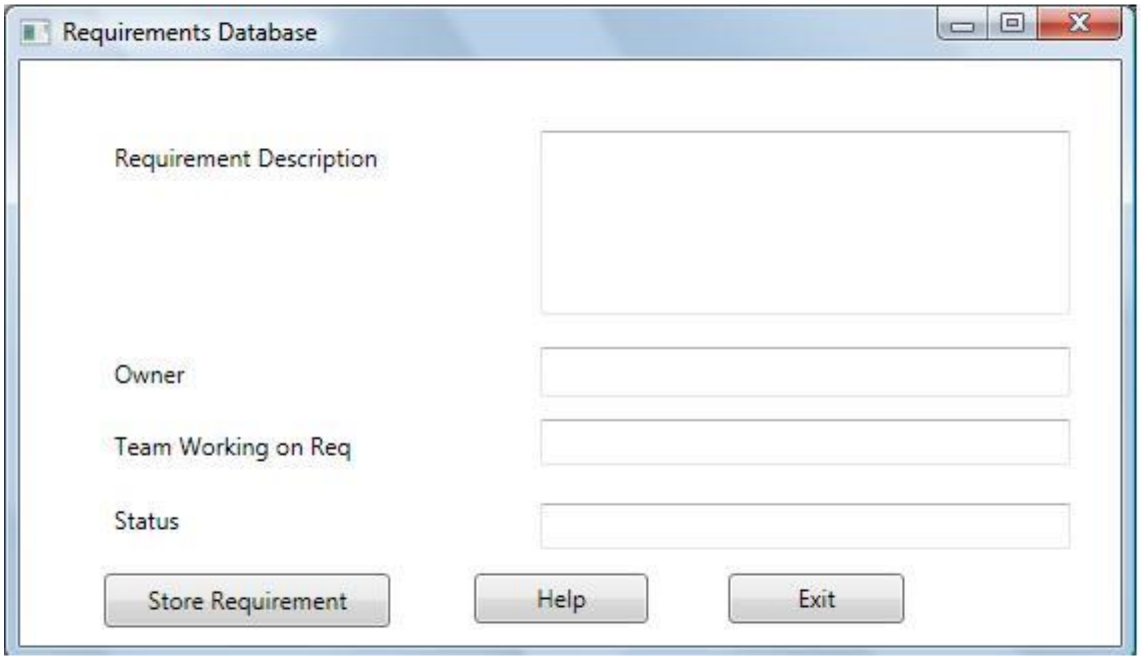
For a relational database, the query language used could be optimized to improve performance. Sufficient effort must be spent for optimization. This will improve performance to a greater extent.

6.6 Views

This section describes the appearance of the tracking tools to its users and designers. The proposed WPF based graphical user interface is depicted in the following screenshots below:




Change Logger
File Name
Version
Change Description
Store Details
Help
Exit


Requirements Database
Requirement Description
Owner
Team Working on Req
Status
Store Requirement
Help
Exit

All the views shown above for each of the tool are similar except that the data that is fed by the user is different.

1) The user gives the required details.

2) The user clicks on the Store button corresponding to each tool so that all these details are fed into the database.

The User can also view the data that would be retrieved from database. There will be a separate user interface for that functionality which accepts search criteria and displays the results.

7. Pretty Printer

7.1 Introduction

Pretty Printer is a simple tool that provides output that is a transformation of a specified source file that obeys a rule set concerning the appearance of source code. For a code to be maintainable and understandable, its code must be formatted properly. This process may consume a lot of time when done manually. Also there are many layout issues that need to be addressed which can be detected using some pre-defined rule set and can be automated thus saving a lot of time.

For example, a manual page and maintenance page is required for each source file to describe about it. For a project involving a large set of code baseline, it consumes a lot of time and effort in adding them to each file. This tool may detect these types of layout issues and put manual and maintenance page in each input file thus saving tons of time.

This is the only tool which should have write access to the files in the repository as it is performing a transformation of the source file adjusting its layout and appearance and also the implementation would not be affected because of this.

7.1.1 Objective and key idea

Its main objective is to transform a specified source file so that it obeys a rule set concerning the appearance of source code. The key idea is to use a Parser by defining some rules concerned with appearance, run it against each file and adding the missing things and transform it into a formatted code.

7.1.2 Obligations

The tool will be developed as an efficient and user-friendly one meeting all the stated requirements. The main obligation of the tool would be

- Transform an input source code into a formatted code based on some rule-sets.

7.2 Uses

The following are considered as the uses of this simple tool:

Uses:

- A formatted code is easier to maintain
- Easily readable and understandable for any developer especially for new developers
- Detect and solve common layout issues thus saving a lot of time and effort

7.3 Activities

We shall assume that rules concerning appearance would include addition of a manual and maintenance page template and addition of test stub template if it doesn't exist for each file. There could be many other rules concerning the appearance of source code. There could be another rule to merge multiple empty lines consecutively present in the code into a single empty line so that the code looks neat. The major tasks would be as follows:

7.3.1 Get the input from user

Get the input filesets which need to be transformed from the user through a user interface specific to this tool.

7.3.2 Retrieve the files from repository

As the filesets contain only the file paths, retrieve the files from the code repository.

7.3.3 Store the templates to be added

Initially store the templates for manual page and test stub in some file or data structure so that it could be used while processing each file.

7.3.4 Perform addition of needed templates if not available

Initially check the source code if it contains manual and maintenance page and test stub sections as these are essential for each file. This could be done using the Parser using some pre-defined rules. This is explained while discussing about the Rule-based Defect analyzer tool which reports rule violations like absence of manual page and test stubs. In case any of the templates is missing, add the needed templates from some store.

7.3.5 Perform merging of consecutive multiple empty lines

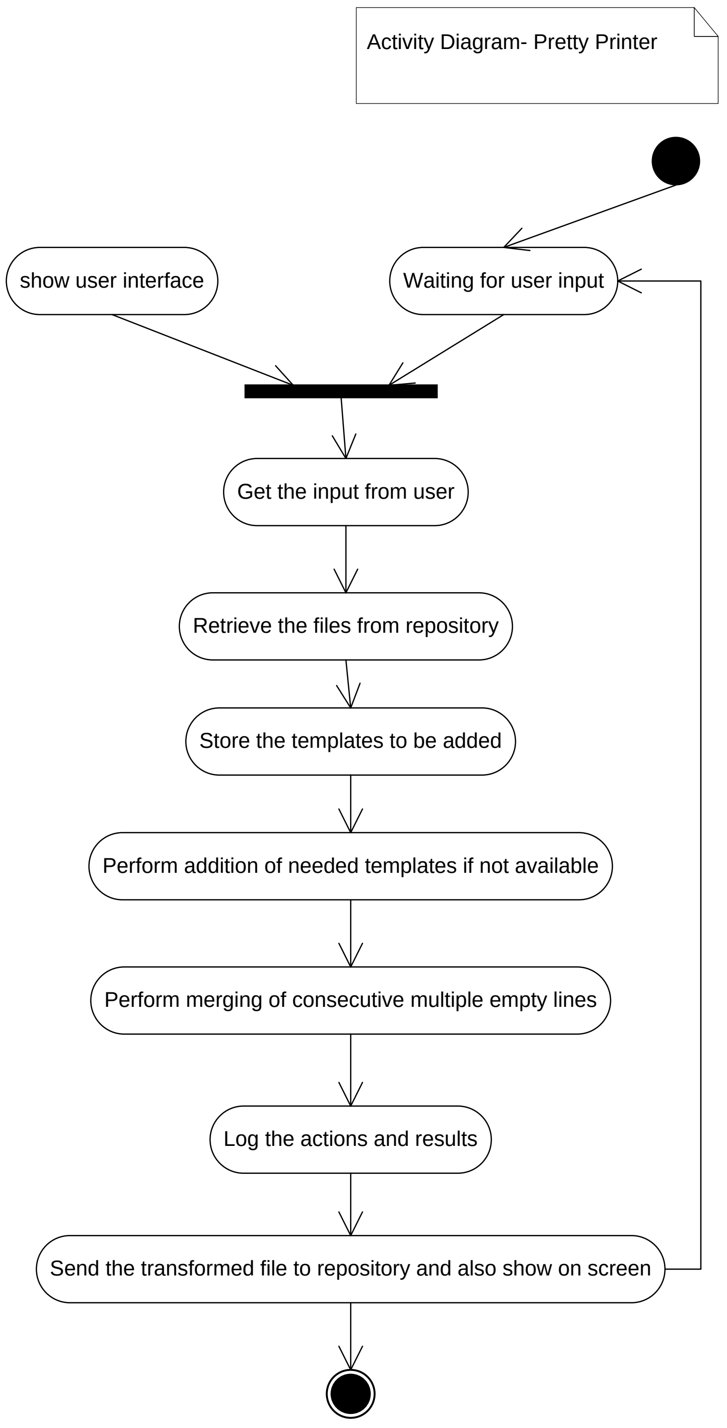
In case the source code contains multiple empty lines consecutively, which could be identified by using a rule and building a parser out of it, these will be merged into single empty line by removing all empty lines and adding a single empty line.

7.3.6 Log the actions and Results

All the changes that are made to the file as part of the transformation should be logged. As only templates are added, it is essential to report to the user about the incomplete manual page that has been added so that he/she could fill and complete those sections. This is true for a test stub section also.

7.3.7 Send the transformed file to repository and also show on screen

The transformed output file is written to the code repository reflecting the changes and also the output is shown on screen.


Activity Diagram- Pretty Printer
show user interface
Waiting for user input
Get the input from user
Retrieve the files from repository
Store the templates to be added
Perform addition of needed templates if not available
Perform merging of consecutive multiple empty lines
Log the actions and results
Send the transformed file to repository and also show on screen

7.4 Partition

Bases on the tasks and the activities within each task, the tool is subdivided into 7 modules as described below:

7.4.1 Input module

This module is responsible for accepting user input. The input is a fileset which needs to be transformed with changes in layout. The fileset contains paths of files pointing to the code repository.

7.4.2 Transform File module

This module is invoked by the input module after accepting the user input. This module configures a parser with some rules concerning the appearance of the source code and then sends the files to the Parser module for analysis.

7.4.3 Parser module

The Parser module is a container of rules each of which is a detector for some grammatical construction like class, struct, interface, enum, functions etc., This is written by Dr.Fawcett. The rules contain a set of actions which is executed when a particular rule is detected. For example when a class or interface definition rule is found out, appropriate action is taken. The parser uses the services of Tokenizer module and Semi-Expression evaluator module for its operation. Besides the existing classes and interfaces, some additional classes are needed to achieve the required functionality.

7.4.4 Semi-Expression evaluator module

This module contains the C# semiexpression detection class written by Dr. Fawcett. Semiexpression is a partial C# expression, and is a sequence of tokens that have meaning for code analysis. It is a minimal sequence of tokens that ends with the characters '{', '}', ';', or '\n' if the line in which the newline appears starts with the char '#'.This module uses Tokenizer module for its operation.

7.4.5 Tokenizer module

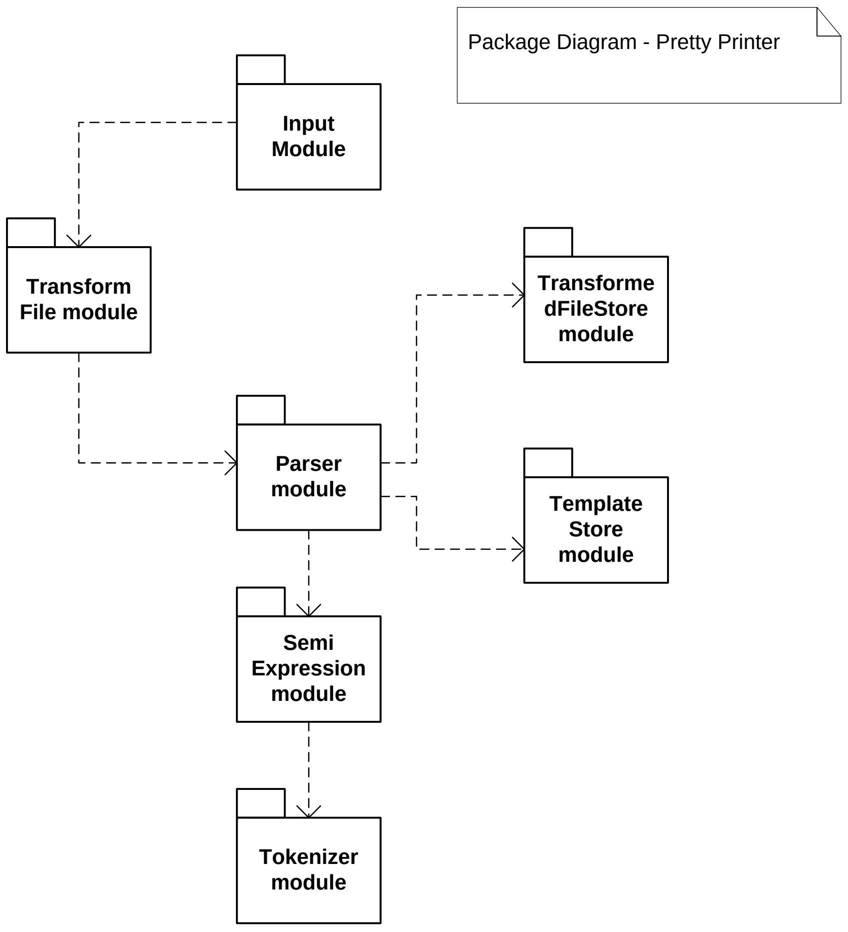
This module contains the C# tokenizer class written by Dr. Fawcett. This is a simple tokenizer that does not combine multi-character operators (such as "<<", "+=", "++") into single tokens. It supports reading words, called tokens, from a string or a file stream. Transitions from alphanumeric to whitespace or punctuator characters and back again, are treated as token boundaries. Quoted strings and comments are returned as single tokens.

7.4.6 TemplateStore module

This module stores definitions for templates that need to be added to source files that are transformed. These are associated with layout of the source code and hence don't change the implementation. Some of the sample templates would be that of manual and maintenance pages, test stub section etc.

7.4.7 TransformedFileStore module

This module manages files once they are transformed an need to be sent to the repository. It temporarily creates a folder to store all the transformed files so that they could be updated in the repository by navigating that folder.



7.5 Critical issues

7.5.1 Controlling length and readability of code

As this tool detects layout issues and also changes the source code by adding certain templates and does alignment, we should ensure that the code doesn't get lengthy and the code remains readable after solving the layout issues.

Solution:

Only important templates are added and the templates should be short enough so that it indicates the user that the section added should be filled as needed by the developer.

7.5.2 Impact on testing

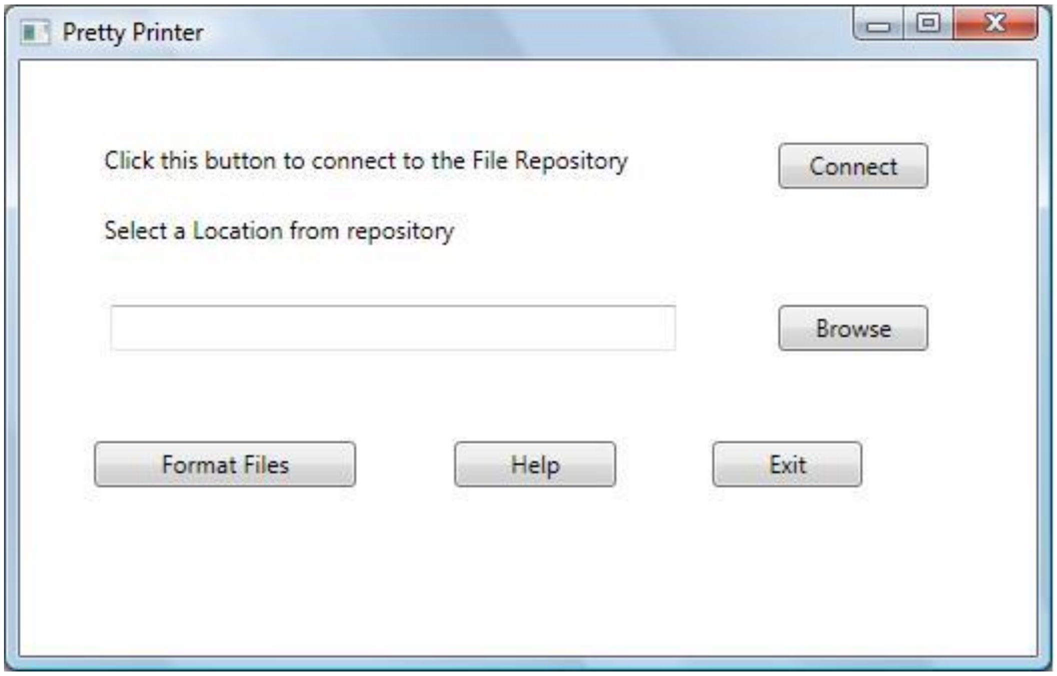
As this tool changes the source code, we should ensure that there is not a lot of time spent on testing the source files after running this tool since it only changes layout and not implementation.

Solution:

It is useful to spend a lot of effort in testing the proper functionality of this tool since it changes source code so that we can ensure that the implementation part doesn't get modified and less effort could be made while testing.

7.6 Views

This section describes the appearance of the Pretty Printer to its users and designers. The proposed WPF based graphical user interface is depicted in the following screenshots below:


Pretty Printer
Click this button to connect to the File Repository
Connect
Select a Location from repository
Browse
Format Files
Help
Exit

1) The User selects a remote location after connecting to the repository server.

2) After this, the user selects Format Files option to execute the Pretty Printer tool on the input files in that location.

8. Server-Based Tool Holster

8.1 Introduction

The server based tool holster is a manager tool for all the QAT tools and is responsible for deploying the appropriate tool based on request from the client. When dealing with a large set of tools as part of a tool set it is very helpful to have a single source of management and contact for all these tools which takes care of managing addition of new tools, removal of existing tools and deployment of tools. A tool could be a library or an application and hence the holster can accordingly deploy each tool. All tools must be registered with the tool holster.

The holster spawns a new process to run each library or application.

8.1.1 Objective and key idea

The objective of the tool holster is to act as a single point of contact to use all the tools by the clients. It has knowledge of all the tools present across the network and to achieve this, all tools must be registered with the tool holster. The key idea is to use a configuration file that is XML-Based and stores information about all tools. When a tool gets registered, a new entry is placed in this file.

8.1.2 Obligations

The tool will be developed as an efficient and user-friendly one meeting all the stated requirements. The main obligation of the tool would be

1) Act as a single source of management for all the tools
2) Register new tools
3) keep the configuration file up to date and consistent.
4) Send the results of each tool to Logger module after deploying them

8.1.3 Organizing Principle

Having a Single Tool holster to deploy all the tools could be considered as an organizing principle relevant to the context of this system.

8.2 Uses

The Tool holster is the heart of this system. A Tool holster is mainly in the server but there could also be some at the client machines, the reason for which has already been discussed earlier while discussing the architecture of the system. The following are considered to be the uses of this tool.

1) This acts as a single source of contact with the tools

2) Registration of tools and managing them are abstracted from developers

3) The complexity of the Architecture of the system is reduced by using this tool holster

4) Having a tool holster in client machines avoids network access for tool execution.

5) The holster provides a pluggable interface so that any arbitrary new tool could be plugged-in.

8.3 Activities

8.3.1 Get Tool information

A new tool may be added to the system or an existing tool could be removed from the system. Both has to updated with the toll holster and hence the holster has to get the tool information initially

8.3.2 Register a tool

In case of registration, an entry is added to the XML-Based configuration file with the tool information. The structure of the configuration file is discussed in the Appendix section.

8.3.3 Remove a Tool

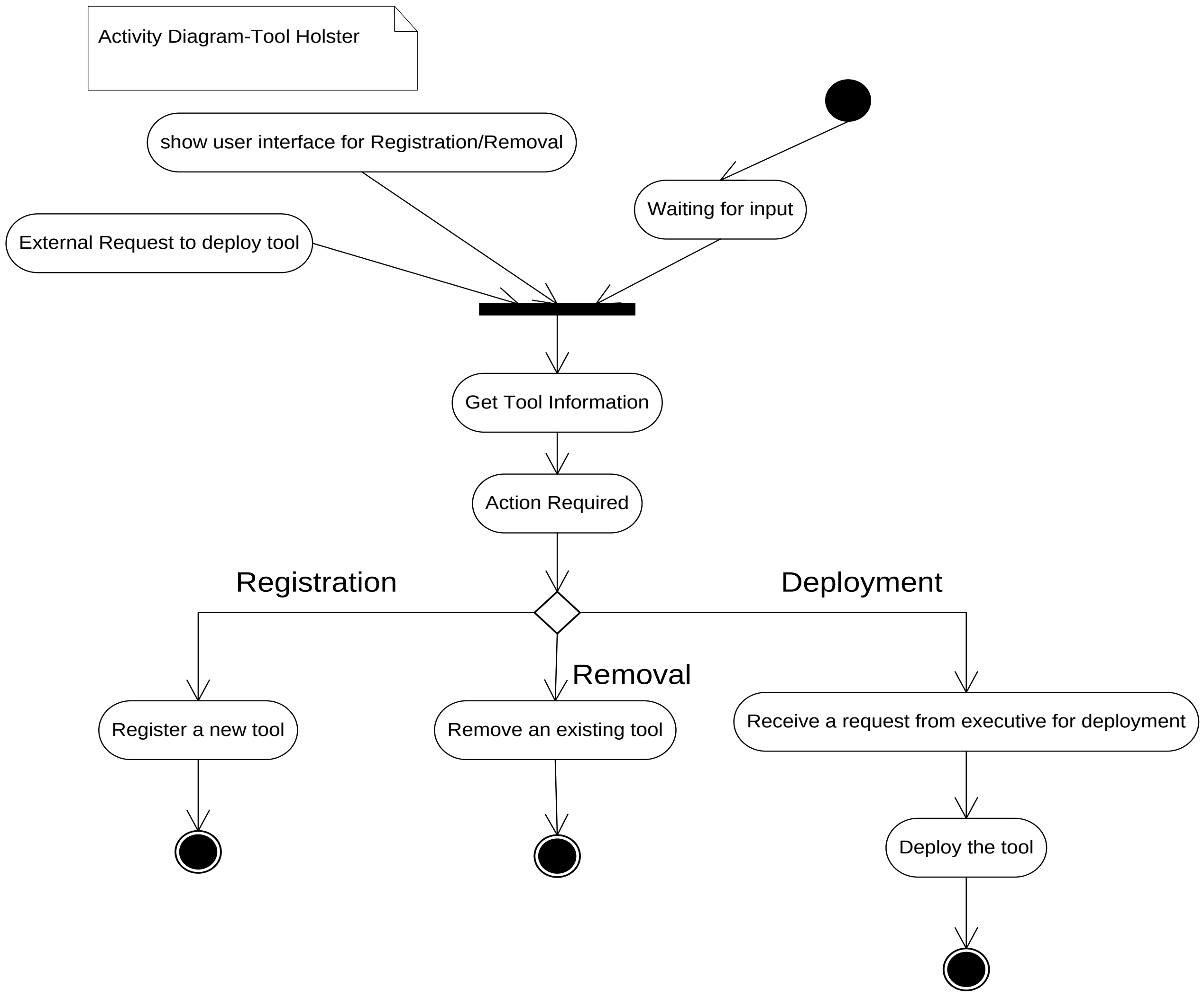
In case of removal tool, an existing entry is removed from the configuration file based on the tool information.

8.3.4 Receive a request from Local or Remote Executive for tool deployment

The tool holster receives a message from the local or remote executive based on its location with the tool name and a fileset in order to execute the tool with the fileset as input.

8.3.5 Deploy a tool

For deploying a tool, the holster spawns a new process and runs the tool by passing its input as command line arguments. It creates instances of the tool using s static creational function provided by the plugin tool itself. The details of creation of this instance are discussed in the Partitions section. Deployment of a tool occur when the tool holster receives a request from a Remote or a Local Executive depending on the holster being in the Server or client side respectively.


Activity Diagram-Tool Holster
show user interface for Registration/Removal
Waiting for input
External Request to deploy tool
Get Tool Information
Action Required
Registration
Deployment
Removal
Register a new tool
Remove an existing tool
Receive a request from executive for deployment
Deploy the tool

8.4 Partition

Bases on the tasks and the activities within each task, the tool is subdivided into 7 modules as described below:

1) Input Module
2) Tool Holster Interface Module
3) Tool Holster Factory Module
4) Tool Holster Manager Module
5) ConfigFileWriter Module
6) Tool Communicator module
7) Log Manager Module

8.4.1 Input Module:

This module implements a small user interface for the tool holster to support registering and removal of tools from its configuration file. Although other ways of achieving it are discussed in the Critical Issues section, this is one of the architectural possibilities. In this interface a tool path could be selected from any remote location and then any of the actions like Registration or Removal could be performed. Deployment of the tool is possible only when a request comes from a client machine and will not be supported by this interface.

8.4.2 Tool Holster Interface and Factory Module:

It should be possible to add new tools without changing the tool holster code. The structure should also support addition of any arbitrary new tool. To achieve this, the holster provides a plug-in interface and an object factory that will provide means to instantiate plug-in instances, either through reflection or by providing means to subscribe for invocation of a static creational function provided by the plug-in. A delegate could be provided to implement this. Registration would have to be part of the new tool installation, which is essentially an entry in the XML-based configuration file. This could be done using the input module's simple user interface.

8.4.3 Tool Holster Manager module

This module is responsible for major operations carried out by the Tool Holster. Whenever a new tool is registered or an existing tool is removed, the details of the tool are passed on to the ConfigFileWriter module to update the changes in the configuration file. Besides this, whenever a request comes from a client for execution of a tool, it passes on the request to the ToolCommunicator Module to send messages to the tool and deploy it.

8.4.4 ConfigFileWriter Module

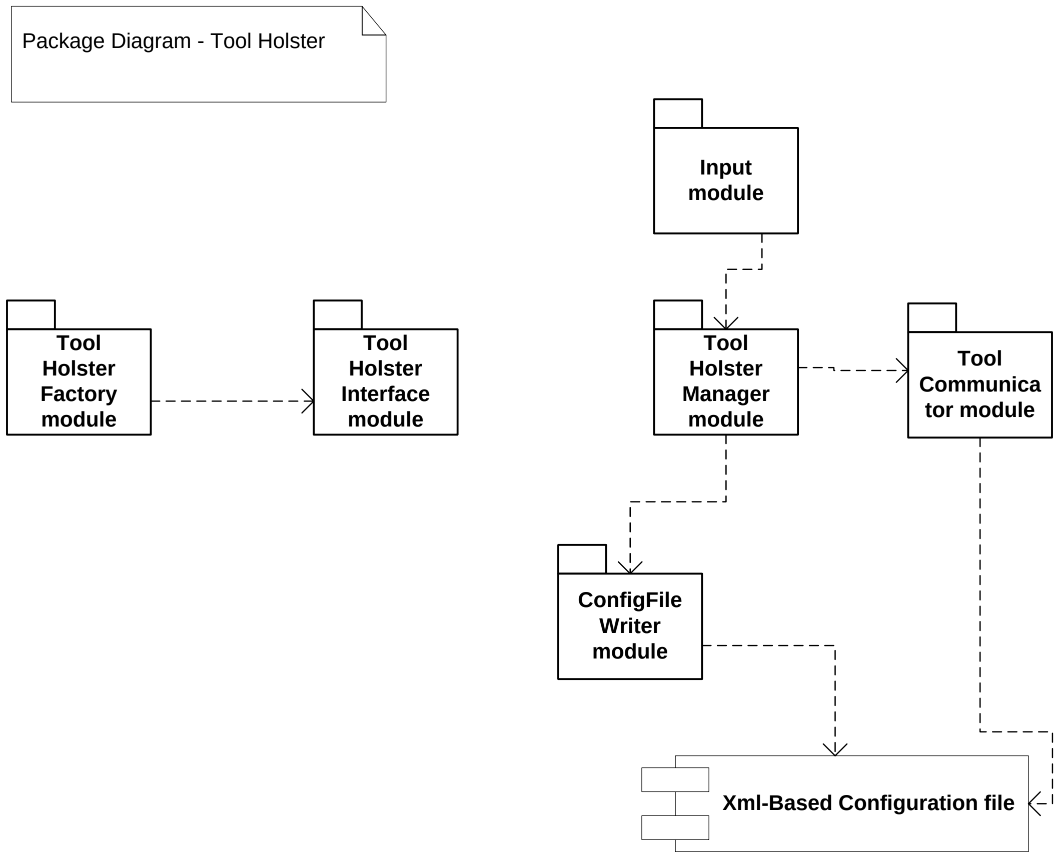
This module is responsible for updating the Xml-based configuration file with tool information as part of registration process and also removal of tools. All changes to the config file happen through this module.

8.4.5 Tool Communicator Module

This module is responsible for accessing the config file to know the location of the tool whenever a request is passed from the Tool Holster Manager module. Once it identifies the location of the tool, it communicates with the tool by sending messages and is responsible for deploying the tools.

8.4.6 Log Manager Module:

This module is responsible for passing on the logs received as a result of execution of the tool to the Logger component so that they are queued up for log storage.


Package Diagram - Tool Holster
Input module
Tool Holster Factory module
Tool Holster Interface module
Tool Holster Manager module
Tool Communicator module
ConfigFile Writer module
Xml-Based Configuration file

8.5 Critical issues

8.5.1 The configuration file for the holster should know about all the tool information. It should know information about newly added tools which are not registered and tools that have been removed without its knowledge.

Solution:

A small configuration management tool could be built that can query all the machines in the network to check if there are any new tools installed or removed from that machine. Based on reply from those machines, it can be updated accordingly.

8.5.2 Synchronization

Every client UI has a configuration file which contains all the available tools and this tool should be in sync with the server-based tool holster which contains a list of all the tools.

Solution:

Whenever the configuration file associated with the server-based tool holster is updated, it can send a broadcast message to all machines in the network about its change and based on that, all the configuration files of UI can update themselves so that they are in sync.

Another possible solution is that each client machine can query the server if there are any changes in the configuration file and if so can get a reply back from server about the change.

8.5.3 Interface language of the holster

The hard part is to provide an interface language that is simple but powerful enough for all communication between the holster and plug-in tool.

Solution:

A message-passing interface makes that easier, as we can then move all interpretation of message to the client that knows about the plug-in and the plug-in itself.

So, the new plug-in tool must implement the plug-in interface, provide a creational facility for its internal objects, and document its message interpretation protocol so that clients can construct appropriate messages.

8.5.4 Tool Holster Security

An incoming tool for registration with the tool holster may not be a secure tool. Since it accesses source files, it must be made sure that tool is secure. Similarly, the accesses to the file system should be controlled for the tools.

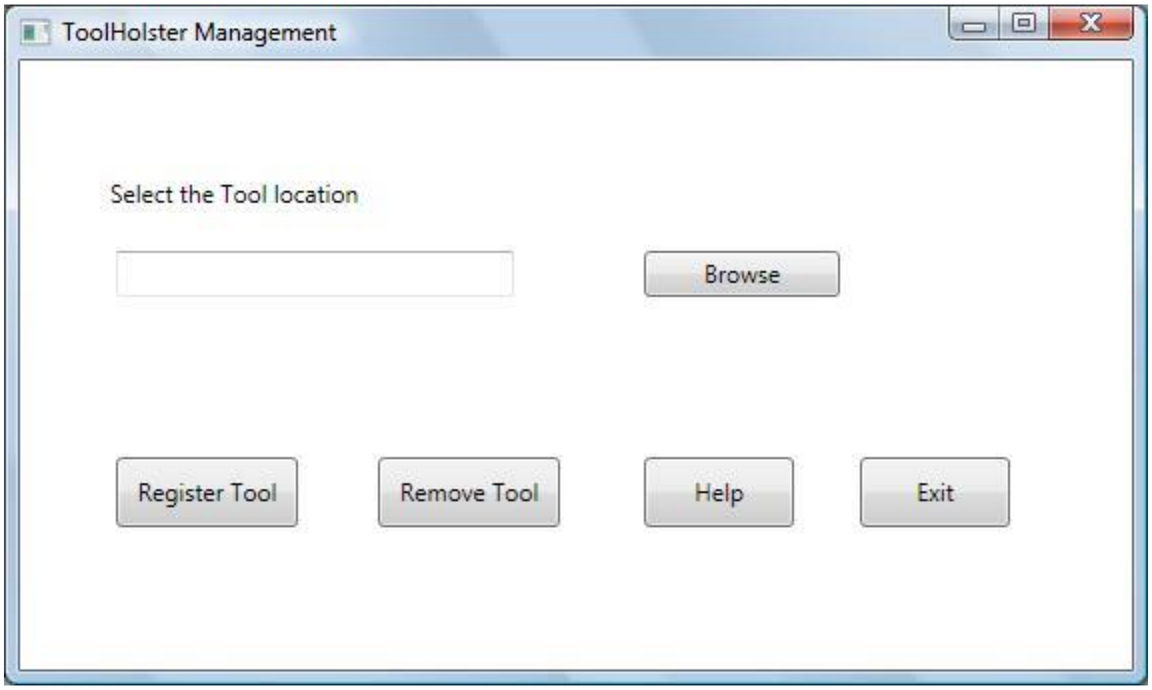
Solution:

Security is an important issue and there could be a wide variety of discussions about that. For our system, to ensure that the incoming tool is secure, the tool holster can test the tool by giving a test file as input to that tool and after its execution, can check the test file for changes if any. If there are no changes, the tool could be considered secure presumably. But in case of any changes in test file, the holster should not allow registration.

Also access control is an important part of tool holster's responsibilities. It provides only read access to the tools to access the code repository. The only tool which may be granted write access is the Pretty Printer since it changes the layout of source files. Hence it is really essential to check if the Pretty printer tool is secured enough.

8.6 Views

This section describes the appearance of the Tool Holster to its users and designers. The proposed WPF based graphical user interface is depicted in the following screenshots below:



1) The user selects the location of the tool by browsing on the remote machines.

2) After selecting the tool location, the user can Register the tool or Remove a tool from the holster.

3) A confirmation message is presented to the user after the request is processed.

4) In case the holster finds that the tool is not secure, the user will be notified of the same.

9. Agents

9.1 Introduction

Agents are programs that have, in some sense, mobility, are focused on a single set of tasks, and can be scripted. An example is an agent that collects logs/results of a Risk analyzer tool that is scheduled to run on a particular date and time. In this case, agent mobility means the ability to invoke the logger component to retrieve the results. Scripting could be as simple as providing an XML file that defines what results are expected. This implies that a small language for specifying search data has been defined that the agent recognizes. It would probably be helpful to build a small tool to help the manager create the XML and launch the agent with that script. Here is an example of a script.

```
<Script>
  <ToolResults>
    <Start>01/01/10</Start>
    <End>15/02/10</End>
    <ToolName>RiskAnalyzer</ToolName>

    <Where>

          <Team>*<Team>
    </Where>
  </ToolResults>
</Script>
```

The agents that are part of this system can reside in code server as well as clients and can be scripted to run scheduled QA analyses and generate reports of the same. For example, an agent could be scheduled to run a Risk analyzer tool on a particular date and time without human intervention. An input fileset is provided to the agent in order to carry out its task.
The agent would have a small, simple XML parser, probably built with Linq.

9.1.1 Objective and key idea

The objective is to develop small programs that can be scripted to conduct and report on scheduled QA analyses without human intervention. The key idea is to use an XML script to run the agent defining the actions that it need to carry out and specify the date and time in a simple interface along with the input script.

9.1.2 Obligations

The tool will be developed as an efficient and user-friendly one meeting all the stated requirements. The main obligation of the tool would be

- Conduct scheduled QA analysis
- Generate reports of QA analysis

9.1.3 Organizing Principles:

Providing a script for the agent in any specific language that the agent can parse and understand could be considered as an organizing principle used in the context of running the agents.

9.2 Uses

The following would the uses of an agent program:

1) The most important advantage in using agents is that there is no manual intervention in running the tool as the tool can be scheduled.

2) A manager/lead will generally use the agents to conduct routine QA analysis to know the state of the project.

3) The reports automatically generated by the agent helps the manager in viewing the history of the tool and analyzing the project's code baseline.

4) As it can be scheduled, it prevents a manager to remember to run specific tool on a certain day in his busy schedule.

5) The reports generated can be used by the manager to present it to the customer and explain to them the status of the project.

6) Scheduling saves a lot of time and effort even though considerable time need to be spent on defining script and input for the agents.

7) The logs are retrieved for any duration of time and for any tool and a report is generated by the agents. This prevents a user especially the manager to manually check a particular log or collect logs within a period of time from a large log repository.

9.2.1 Use cases:

The following would be the use cases:

1) Define a script for the agent

2) Specify date and time to schedule agent

3) Define an input for that agent

9.3 Activities

The activities of an agent are divided into those that are involved in providing inputs for agents and those that are part of agent execution. Both are described and depicted as below.

Providing input for agents :

1) Get the script for agent

As an Xml script is required to conduct the agents, get that script from the user which is used to define the action that the agent need to carry out

2) Get date and time to schedule

As agents could be scheduled, define the date and time when it needs to run.

3) Get the input fileset/ensemble

For the agents to run a tool, an input fileset need to be provided. A configuration tool could be used for this purpose by the managers.

4) Create a job or batch file that launches the agent when scheduled

Create a job or a batch file in order to launch the agent with the script and specified input at the specified date and time. This is to ensure scheduling of the QA analyses.

Activities within an agent:

1) Parse the script

The input script is parsed using an Xml parser, probably built with Linq and the information is extracted from the script.

2) To conduct analysis, create an XML message

In case of QA analysis, an Xml message is created with the tool name and the input file set.

3) Send the message to Tool Holster

The constructed Xml message is passed on to the Tool holster (client or server as discussed earlier).

4) Run the tool

Once the message arrives at the tool holster, the tool is deployed and run with the input fileset

5) To generate reports, Define rules/actions to search data

In case of generating reports, a Parser structure could be used. Define rules and actions in order to search data required.

6) Define rules/actions to collect data

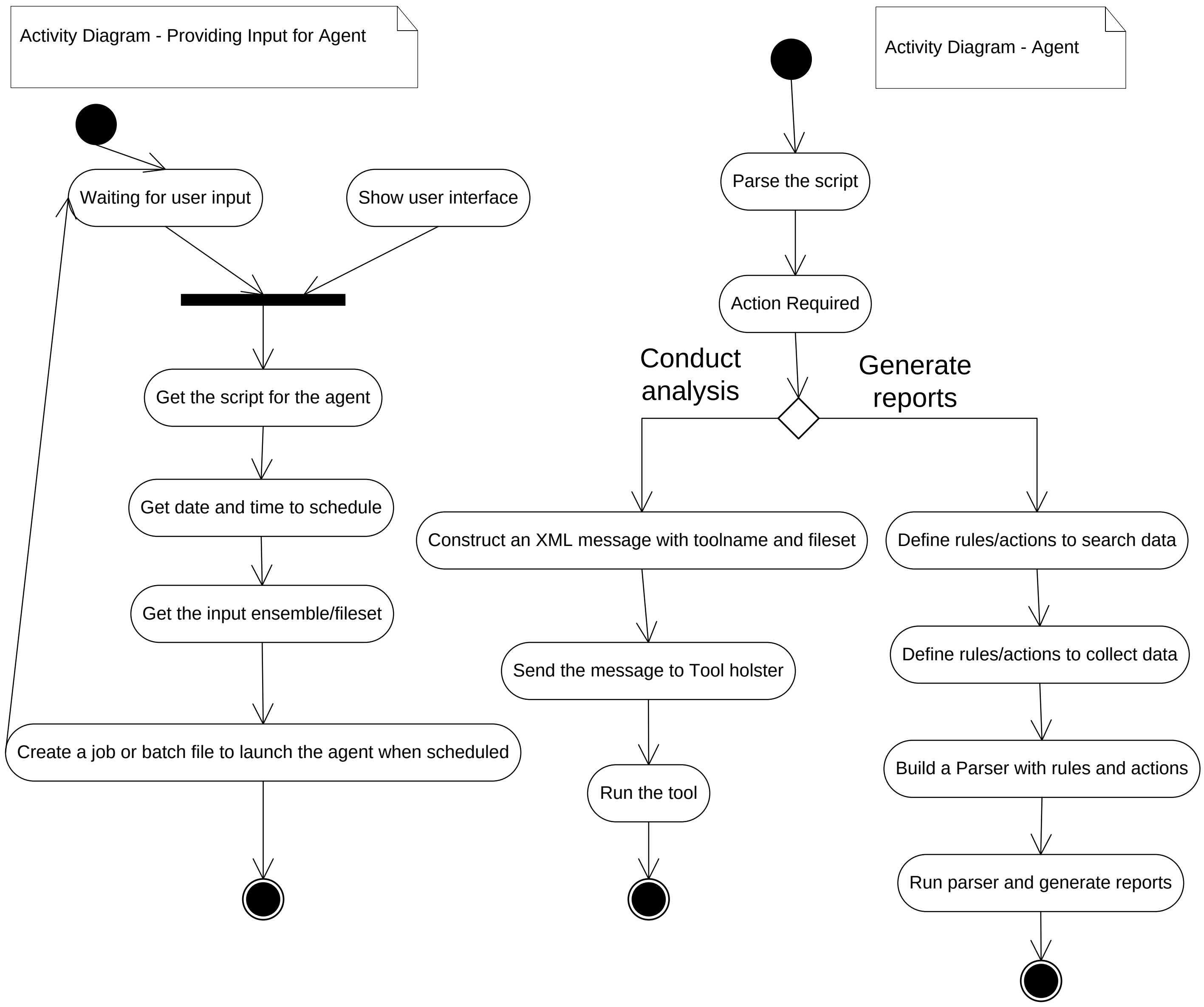
Define rules and actions in order to collect the data required to generate reports

7) Build a Parser with rules and actions.

Configure the parser using the rules and actions defined above so that when the Parser runs, the reports are generated appropriately.

8) Run the Parser and generate reports.

Finally run the parser and generate all the required reports according to the script specification.


Activity Diagram - Providing Input for Agent
Waiting for user input
Show user interface
Get the script for the agent
Get date and time to schedule
Get the input ensemble/fileset
Create a job or batch file to launch the agent when scheduled
Activity Diagram - Agent
Parse the script
Action Required
Conduct analysis
Generate reports
Construct an XML message with toolname and fileset
Send the message to Tool holster
Run the tool
Define rules/actions to search data
Define rules/actions to collect data
Build a Parser with rules and actions
Run parser and generate reports

9.4 Partition

Bases on the tasks and the activities within each task, the tool is subdivided into 16 modules as described below:

1) Input module
2) Agent Scheduler module
3) Script Parser module
4) Analysis conductor module
5) Report Generator module
6) Report store module
7) Parser module
8) Semi-Expression module
9) Tokenizer module

The responsibilities of each module, dependencies between them and the contained classes are described in detail below:

9.4.1 Input module

This module provides a simple user interface for the users like managers to schedule the agents with required information. It accepts the script used to conduct agent, date and time to schedule and the input fileset for feeding the agent.

9.4.2 Agent Scheduler module

This module is responsible for creating a job or a batch file that launches the agent at the specified date and time automatically. This schedules the agents based on the input information.

9.4.3 Script Parser module

This module is responsible for parsing the input script that contains information to run the agent. It uses a simple Xml parser like one build with Linq to parse the script and extract information out of it. Based on the extracted information it either invokes Analysis Conductor module or Report Generator module.

9.4.4 Analysis conductor module

This module is responsible for conducting the scheduled QA analysis by running the specified tool. It constructs an Xml message with the tool name and input fileset and then sends these messages to the Tool Holster which is responsible for deploying the tool.

9.4.5 Report Generator module

This module is responsible for generating reports for a specified date or a period of time. It uses the Parser module to build a parser using rules and actions that are defined to search and collect data for the agent. The log repository is accessed by the Parser rule and logs are retrieved based on search criteria like date, tool name and team name.

9.4.6 Report Generator module

The generated reports are stored and maintained by this module. The reports generated are stored in a specific location decided by this module. The user later goes to that location and sees the reports.

9.4.7 Parser module

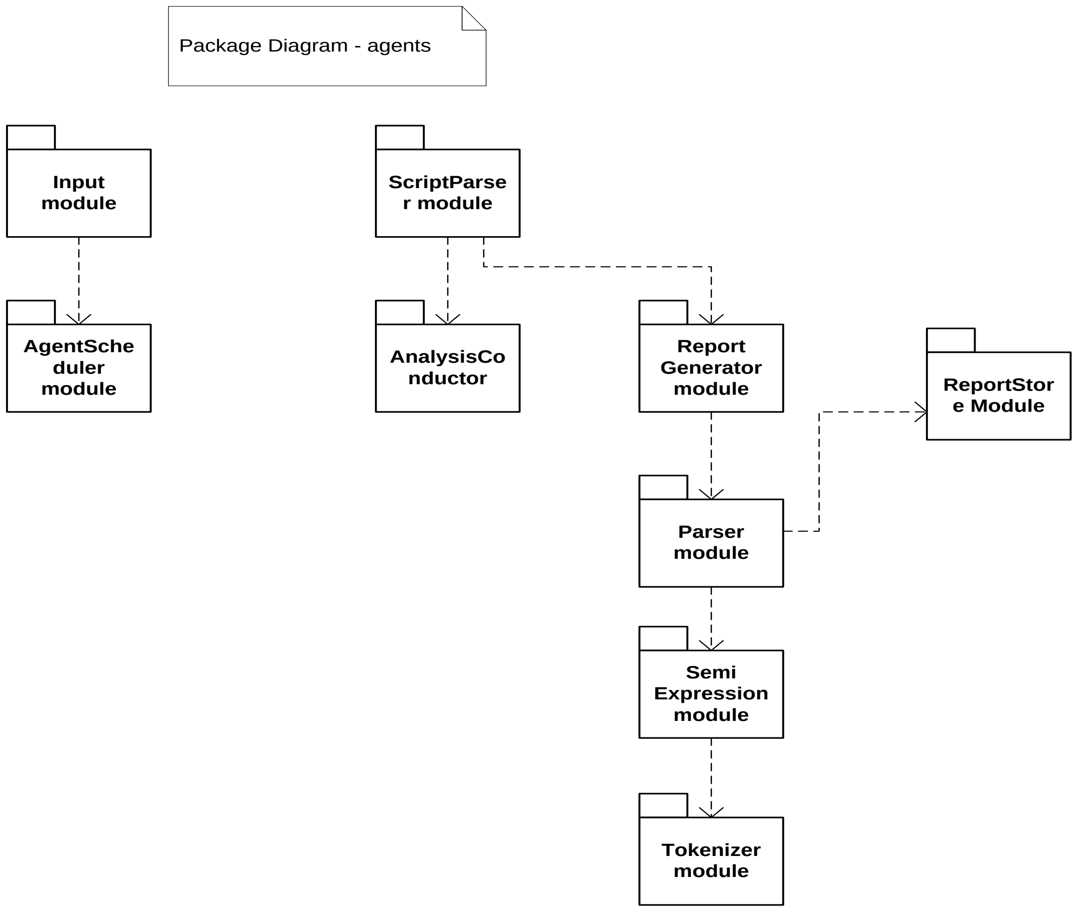
The Parser module is a container of rules each of which is a detector for some grammatical construction like class, struct, interface, enum, functions etc., This is written by Dr.Fawcett. The rules contain a set of actions which is executed when a particular rule is detected. For example when a class or interface definition rule is found out, appropriate action is taken. The parser uses the services of Tokenizer module and Semi-Expression evaluator module for its operation. Besides the existing classes and interfaces, some additional classes are needed to achieve the required functionality.

9.4.8 Semi-Expression evaluator module

This module contains the C# semiexpression detection class written by Dr. Fawcett. Semiexpression is a partial C# expression, and is a sequence of tokens that have meaning for code analysis. It is a minimal sequence of tokens that ends with the characters '{', '}', ';', or '\n' if the line in which the newline appears starts with the char '#'.This module uses Tokenizer module for its operation.

9.4.9 Tokenizer module

This module contains the C# tokenizer class written by Dr. Fawcett. This is a simple tokenizer that does not combine multi-character operators (such as "<<", "+=", "++") into single tokens. It supports reading words, called tokens, from a string or a file stream. Transitions from alphanumeric to whitespace or punctuator characters and back again, are treated as token boundaries. Quoted strings and comments are returned as single tokens.


Package Diagram - agents
Input module
AgentScheduler module
ScriptParser module
AnalysisConductor
Report Generator module
ReportStore Module
Parser module
Semi Expression module
Tokenizer module

9.5 Critical issues

9.5.1 Creating an expressive language:

As the agent runs using an XML-based script, it is important to provide an expressive language that the agent can parse and understand.

Solution:

A simple language as described earlier as a sample script could be used. It should be easier for parsing and understanding by the agent.

9.5.2 Flexibility for the agent to carry out a number of related tasks

The agent should be designed flexible enough to carry out a number of related tasks since it runs without manual intervention. This is an important issue to be addressed.

Solution:

Using the Parser structure for this could make the design flexible. The Parser provides plug-in interface for adding rules and actions for a number of related tasks.

9.5.3 Communicate large set of results to the user

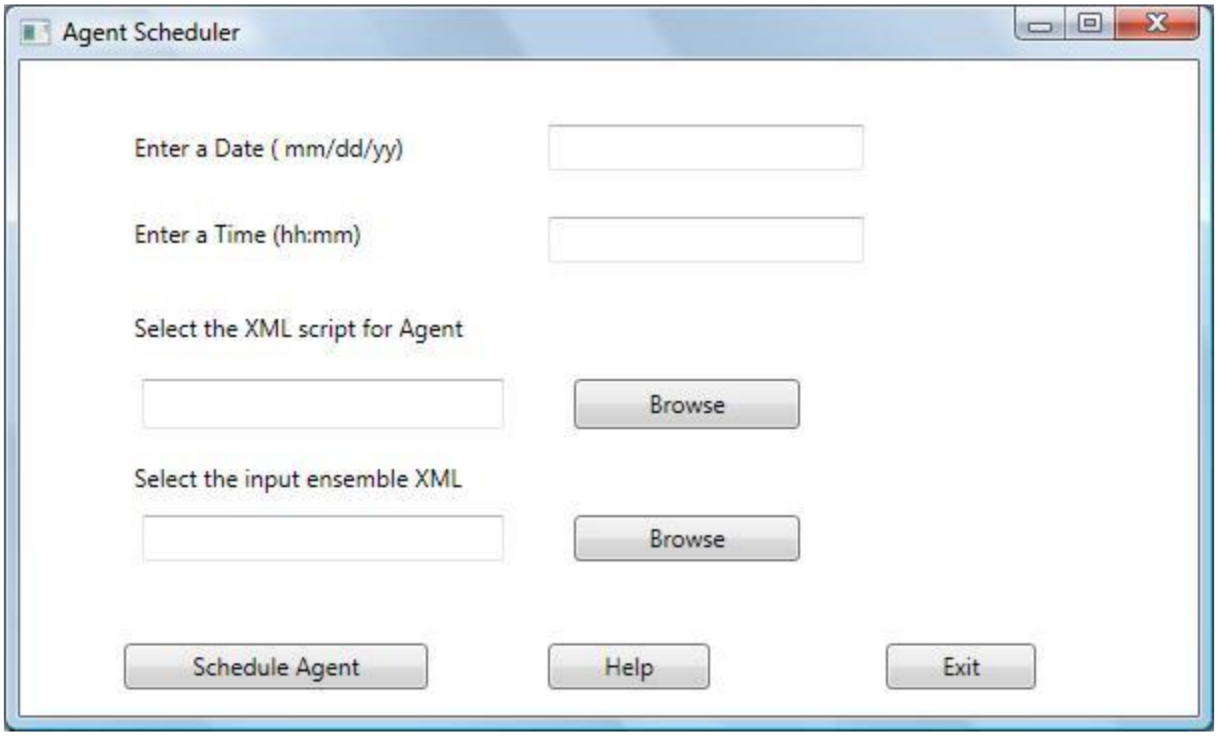
The results generated by the agent could be very huge. For example a report generated for a particular tool for the past one year might be very huge. This should be efficiently communicated to the user.

Solution:

The user can shown output part by part instead of showing everything at once. The user can be provided options to view the recent log files or older log files so that navigation is easier. This feature could be useful for users who mainly want to focus on the recent logs out of the report generated.

9.6 Views

This section describes the appearance of the Agent scheduler to its users and designers. This tool is mostly used by Project managers. The proposed WPF based graphical user interface is depicted in the following screenshots below:



1) The User enters the Date and Time for scheduling the agent

2) The User selects the Xml script for the agent and also the ensemble Xml for providing input for the agent. After selecting the inputs, the user can click on Schedule Agent to schedule the agent with the specified input and script.

10. Configuration Tool to define input

10.1 Introduction

This tool is a very simple tool that assists the users managers like managers in defining input filesets for agents. The input filesets are names and XML-Based so that the agents can parse and understand. The fileset would contains a set of file paths which would be retrieved from the code repository by the analysis agents. The agents basically perform two types of job: conduct QA analysis and report generation. For conducting QA analysis, an input fileset is required on which the analysis takes place. The key idea is to generate an XML based on the files that are selected by the user. The XML might look like the one below:

```
<Input>

<FileSet name=”Parser”>

            <File1>”somePath/Parser.cs”</File1>

            <File2>”somePath/Tokenizer.cs”</File2>

            <File3>”somePath/SemiExpression.cs”</File3>

<FileSet>

</Input>
```

10.2 Uses

1) This tool is used to define input filesets so that every time the user like manager schedules the agents, he need not specify all the input files.

2) It is used to achieve “Define Once, Use Many times” approach.

3) The same fileset could be used for running multiple agents or single agent multiple times.

10.3 Activities

This simple tool performs a small number of activities which are described below:

10.3.1 Get input files

The set of input files for the agent are obtained from the user. The user just selects the path of the files.

10.3.2 Create an XML file

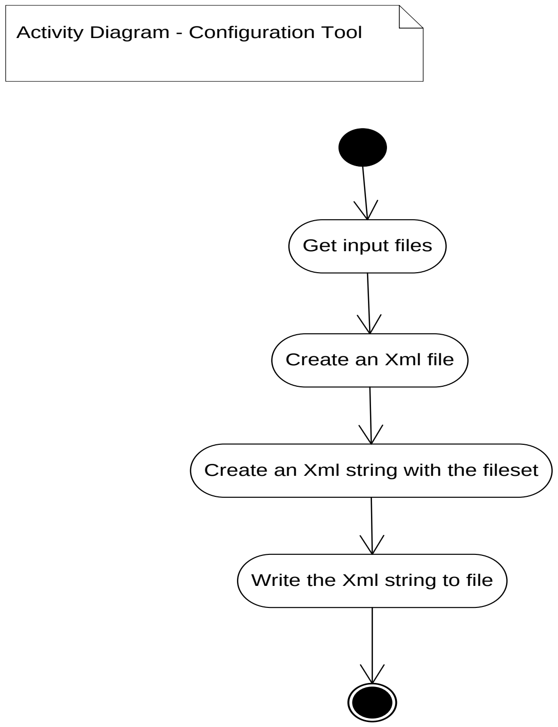
As the second step, an Xml file is created since this tool just creates a named, XML-based representation of filesets for analysis.

10.3.3 Create an Xml string with the fileset

From the raw input, an Xml string needs to be generated. The xml string should represent the set of files in a single tag grouping files together in a component tag. This is just one wa of representing the filesets and there are many other options too.

10.3.4 Write the Xml string to file

After constructing the xml string, the final step is to append the xml string to the files that was created in one of the earlier steps. Thus a named Xml file has been created representing the input fileset.


Activity Diagram - Configuration Tool
Get input files
Create an Xml file
Create an Xml string with the fileset
Write the Xml string to file

10.4 Partitions

Based on the tasks and activities, this simple tool is just divided two modules.

1) Input module
2) Xml File Writer module

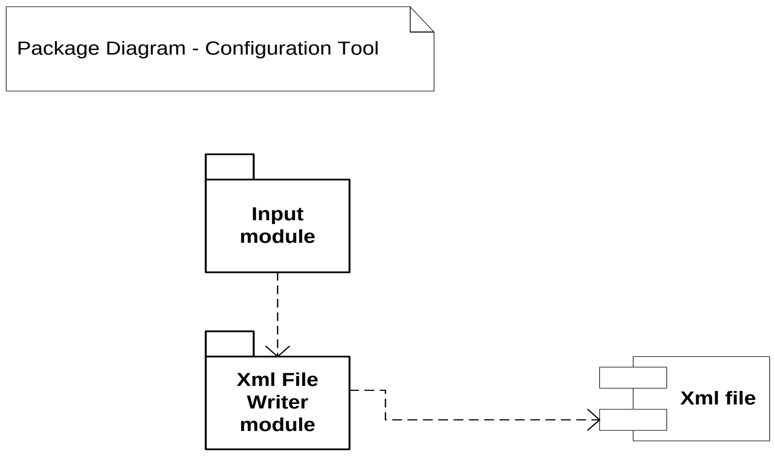
The responsibilities of each module are as follows:

10.4.1 Input module:

This module is responsible for collecting the input filesets selected by the user from a specified path on the repository system. After getting the input, this module passes the files to Xml File Writer module

10.4.2 Xml File Writer module

This module is responsible for creating an xml string representing the input set of files. After this, an xml file is created and the string is written into it. It manages the xml file that is created as an input for the agents.



10.5 Non-Critical Issues

1) Choosing a representation

The only issue with this simple tool is to choose a proper representation of input filesets so that agents can easily parse and understand the input.

Solution:

As the input fileset is Xml-based, there could be tag for each component and child tags that defines file names and paths. This would represent a set of files under a component as one tag in the XML and this protocol could be used by the agent in understanding the input.

11. Logging Facility/Tool

11.1 Introduction

This tool provides an interface for all the plug-in tools to log their actions and results. This is a very important tool since logging the actions and results of tools is very important for the users. The logs could be retrieved at any point of time. The summaries of actions and results of any tool need to be captured since there are some tools which may have performed an incomplete job and in those cases, the logs indicate the same. Without logs, it is not possible to know what the tool has executed and if it has completed its job or not.

For example, a pretty printer should ensure that a source code file has appropriate manual page, maintenance page, and test stub sections. However, it probably will not have enough information to complete those parts, so it should indicate the incompleteness of its output in the tool log.

Objective and Key idea:

The main objective is to capture the output of any running QAT tool. The key idea is to receive the logs generated by the tool as an Xml string and configure the output into a time-stamped XML segment. After configuring the output, it is queued up for log storage since, many QAT tools may be running for more than one client concurrently.

11.2 Uses

The uses of this tool are described as follows:

1) The summaries of actions can be referred by the user to ensure that the processing is completed by the tool. That would indicate any incompletion.

2) Logs within a particular period could be retrieved by users for reference and comparison.

3) History of logs is useful for Managers and Leads to monitor the progress of the project

4) The Logs could be used by the Quality assurance team while assessing the quality of the project.

11.3 Activities

The implementation has many interesting features. The logging facility can use many child threads to assist its processing since it can be contacted from multiple clients concurrently. Hence asynchronous processing is essential for the logging toll to return back to its primary activities once configuring output. The major implementation part involves configuring output into an XML based time-stamped log segment and queue it up for storage. The major tasks or activities are described below:

11.3.1 Receive the log generated by a running QAT tool

The logs generated by the running tools are captured from the tool holster for processing.

11.3.2 Generate an Xml string representing various parts of the log message

An xml string is generated for the log representing the various parts of the log in a proper format. This is because all the log files created by this module are xml-based files.

11.3.3 Create a child thread for processing the Xml string

A separate child thread is started for processing the xml string. The xml string is time-stamped in order to allow easy and efficient searching based on date and time by other tools.

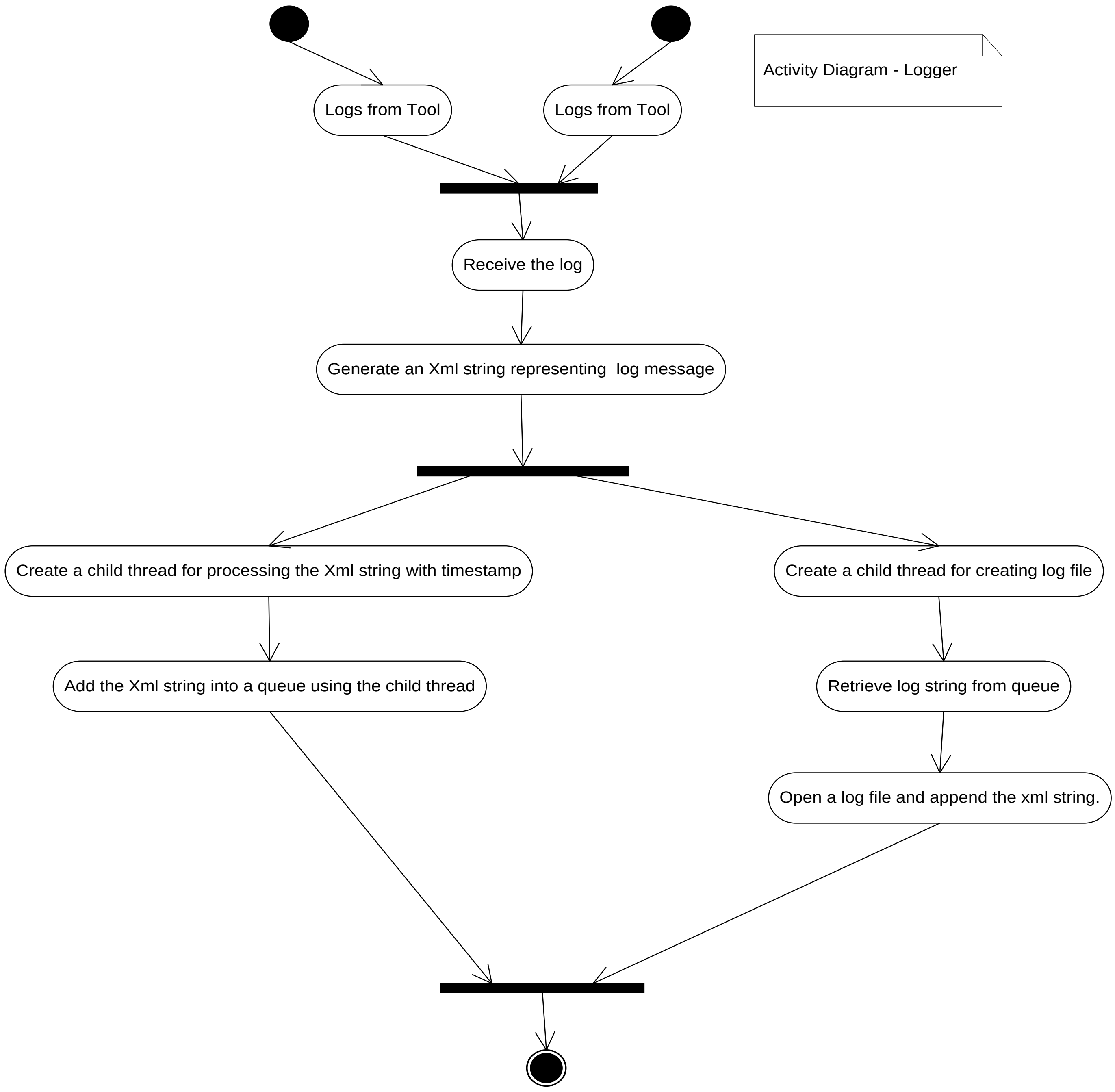
11.3.4 Add the Xml string into a queue using the child thread

After constructing a time-stamped xml log segment, the xml string is added into a queue so that it can continue with processing other xml log strings.

11.3.5 Create another thread to open a log file and append the xml string.

A separate thread is started in order to take the xml string from the queue, create a new xml file and add the contents into that file.

This asynchronous processing helps in handling logs from multiple tools running in multiple clients concurrently.


Activity Diagram - Logger
Logs from Tool
Logs from Tool
Receive the log
Generate an Xml string representing log message
Create a child thread for processing the Xml string with timestamp
Create a child thread for creating log file
Add the Xml string into a queue using the child thread
Retrieve log string from queue
Open a log file and append the xml string.

11.4 Partitions

Bases on the tasks and the activities within each task, the tool is subdivided into 4 modules as described below:

1) Log Manager module
2) Xml Log Generator module
3) Log file Creator module
4) Log Storage Module

The responsibilities of each module, dependencies between them and the contained classes are described in detail below:

11.4.1 Log Manager Module:

This module provides the Logging interface that captures all the logs of a running QAT tool. Once it receives logs of a running tool, it passes on the log to the Xml Log Generator module in order to configure the output log.

11.4.2 Xml Log Generator module:

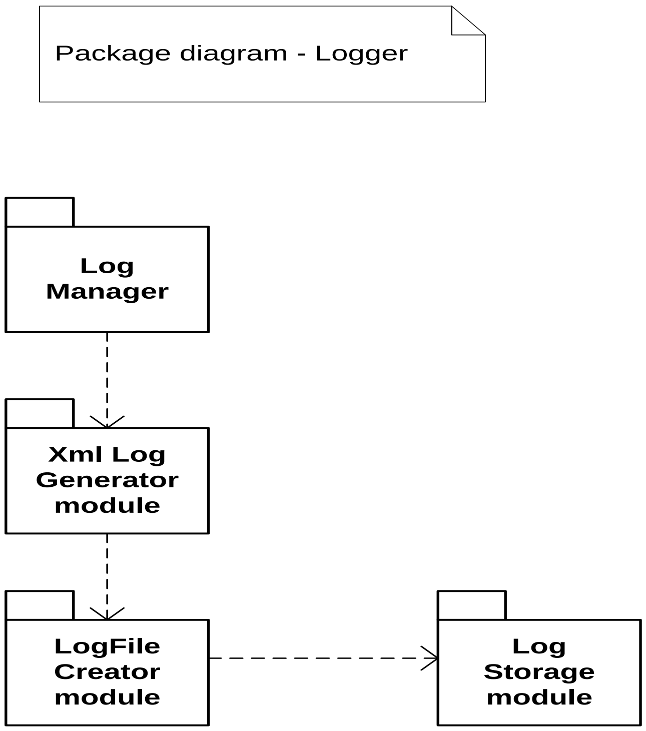
This module is responsible for configuring the output log into a time-stamped Xml log segment containing all the relevant parts of the log. This Xml string is then placed into a queue to be handled by Log File Creator module. All log files created are Xml log files.

11.4.3 Log File Creator module:

This module is responsible for retrieving an xml string from the queue, creating a log file and then appending the xml string into the file. The file name also contains a timestamp – date and time on which the file is created. This file is then stored in the appropriate folder as decided by the Log storage module.

11.4.4 Log storage module:

This module is responsible for storing the files created by the Log file creator module in a folder corresponding to the tool which is executed. It could be assumed that there is folder for each tool to store its logs.


Package diagram - Logger
Log Manager
Xml Log Generator module
LogFile Creator module
Log Storage module

11.5 Critical Issues

11.5.1 Multiple threads accessing queue

In this tool, there are multiple threads accessing a single queue. It is very essential to handle multiple threads accessing single queue and ensure that no deadlock occurs, no race conditions occur and that the thread that dequeues wait till the thread that enqueues items has completed its processing.

Solution:

To solve this issue, a Blocking queue developed by Dr.Fawcett shoule be used. It is designed to perform double checked locking and handle multiple threads efficiently**.**

11.5.2 Xml-Representation of logs

The logs are represented as time-stamped XML log segments and choosing a proper representation that reflects all the parts of the log is essential.

Solution:

The representation should be simple with less number of tags that represent all parts of the log file. This is not so critical to be addressed, but choosing a simple representation makes things simple and saves effort.

12. Communication System

Message passing using WCF system would be appropriate for this architecture for communication across machines and between components as it is very flexible.

WCF is Microsoft's unified programming model for building service-oriented applications. It is a great tool for programmers to build secure and reliable solutions that integrate across multiple platforms. It also has defined standard protocols for application-to-application communication and helps in distributed transaction coordination and reliable communication.

Message Passing (MP) uses a very simple interface as shown below:

```
interface {

Public void postMessage (Message msg);

 Internal Message getMessage ();

}
```

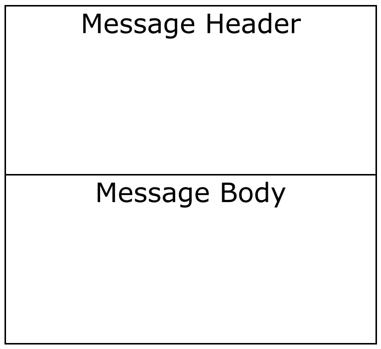
This would ensure that all of the application specific details about what processing occurs and what its results are can be hidden away within the Message structure which is XML , so the communication system can be reused for many types of systems.

Hence, all of the message interpretation moves out into the application code, away from the core services, as it should. Consequently, all of the communication infrastructure can focus on security, activation models, chunking, and the other things that are communication specific.

Another useful property of MP is that messages can easily be queued to help with bursty clients and messages can be saved for later delivery if a channel is currently not operating.

We can use Message contracts for defining XML messages. Message Contract is a class which encodes information about how the message will be transmitted and how it is going to be transformed into a SOAP message. In simple words, it represents a request or response message.

We are going to transmit the information in XML format enveloped using SOAP protocol. The Message object is serialized into XML when defined using a Message contract and is encoded and decoded by WCF at both the end points. A Message contract defines Message Headers and Message Body separately depicted as below.



An XML Message format

A picture of how a message contract and WCF operation contract would look like in presented in the Appendix section. The interface used for Message Passing is implemented using a WCF Operation contract which exposes services to all components. It is also possible to specify protection levels while sending WCF Messages.

With regard to the activation model a Per-Session model could be used for this system. When this model is used, a private session between a client and a particular service instance is established. When the client creates a new proxy to a service configured as session-aware, the client gets a new dedicated service instance that is independent of all other instances of the same service. One possible downside of this model would be that it will not be able to support more number of clients.

A sample XML message that would be sent from the client to server is presented in the Appendix section.

13. Appendix

13.1 Xml Message structure for a Tool

The following is a suggested XML structure for sending messages to the tool when the client requests execution of a tool.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Request>
  <Tool name="RiskAnalyzer">
    <input>
      <returnAddress ip="192.38.26.21" port="8000">
      </returnAddress>
      <fileset path="repository/bin/v_002_00/" ip="192.37.48.1" port="8080">
      </fileset>
    </input>
  </Tool>
</Request>
```

The Xml message contains The tool name, the client return Address to return back the results and finally the input file path in the code repository, it’s ip address and port number for the tool to retrieve the files.

13.2 Xml structure for the config-file used by Tool holster and Client UI

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ToolsInfo>
  <Tool>
    <name>Risk analyzer</name>
    <location>
      <ip>192.36.21.2</ip>
      <path>"bin/tools"</path>
    </location>
    <description>Tool for performing Risk analysis on file sets</description>
  </Tool>
  <Tool>
    <name>Pretty Printer</name>
    <location>
      <ip>192.38.22.8</ip>
      <path>"bin/tools"</path>
    </location>
    <description>Tool for formatting appearance of source code</description>
  </Tool>
</ToolsInfo>
```

The file contains tool information like name, location (ip and path within the machine) and a small description about the tool.

13.3 Xml Structure representing a script for the agent to conduct QA analysis

```
<?xml version="1.0" encoding="utf-8" ?>
<Request>
  <Tool name="RiskAnalyzer">
    <input>
      <returnAddress ip="192.38.26.21" port="8000">
      </returnAddress>
      <fileset source="input_RiskAnal.xml">
      </fileset>
      <date>
        01/02/10
      </date>
      <time> 0900 </time>
    </input>
  </Tool>
</Request>
```

The conduct an analysis, the agents needs the tool name to execute, the return ip address to get back the results, date and time it needs to run and finally, the input files as an Xml-file created by a separate Config-Tool.

13.4 A simple WCF ServiceContract and Message Contract complementing the discussion in Section 8 – Communication System.

```
[ServiceContract]

public interface ICommService

{

  [OperationContract(IsOneWay=true)]

  void PostMessage(Message msg);

  // Not a service operation so only server can call

  Message GetMessage();

}

[DataContract]
```

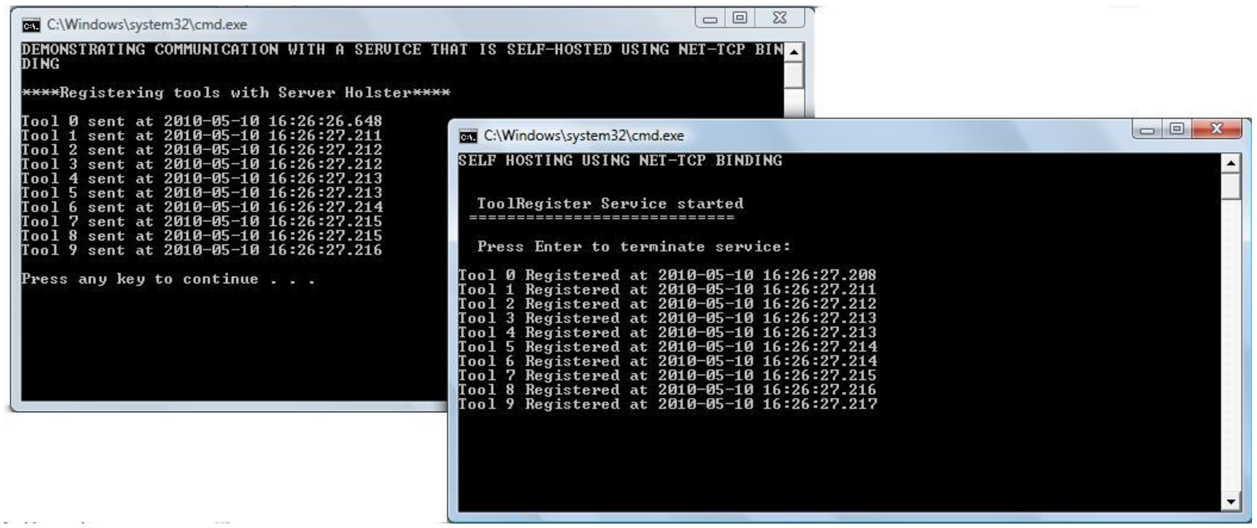
```
public class Message

{

  [DataMember]

  Command cmd = Command.DoThis;

  [DataMember]

  string body = "default message";

  public enum Command

  {

    [EnumMember]

    DoThis,

    [EnumMember]

    DoSomething

  }
```

13.5 Prototype

A prototype has been implemented to support the concept of the architecture of this system. A message passing system has been implemented with self hosted netTcp binding and with IIS hosted WSHttp binding. This system has been analyzed for the message rates for several prototype tool messages for each hosting method and the performance loss for hosting communication services in IIS has been evaluated.

SELF HOSTED NET TCP BINDING:

The below screenshot represents the message transfer rates between the client and a self-hosted service for 10 messages. The service implemented is a prototype service for the Tool Holster component which exposes services for registration of tools. A new tool registers with the holster by calling one of its service methods. In this prototype, tool names are sent from client to the service method and the message transfer rates are recorded. The screenshot shows the time the message is sent and time it is received.



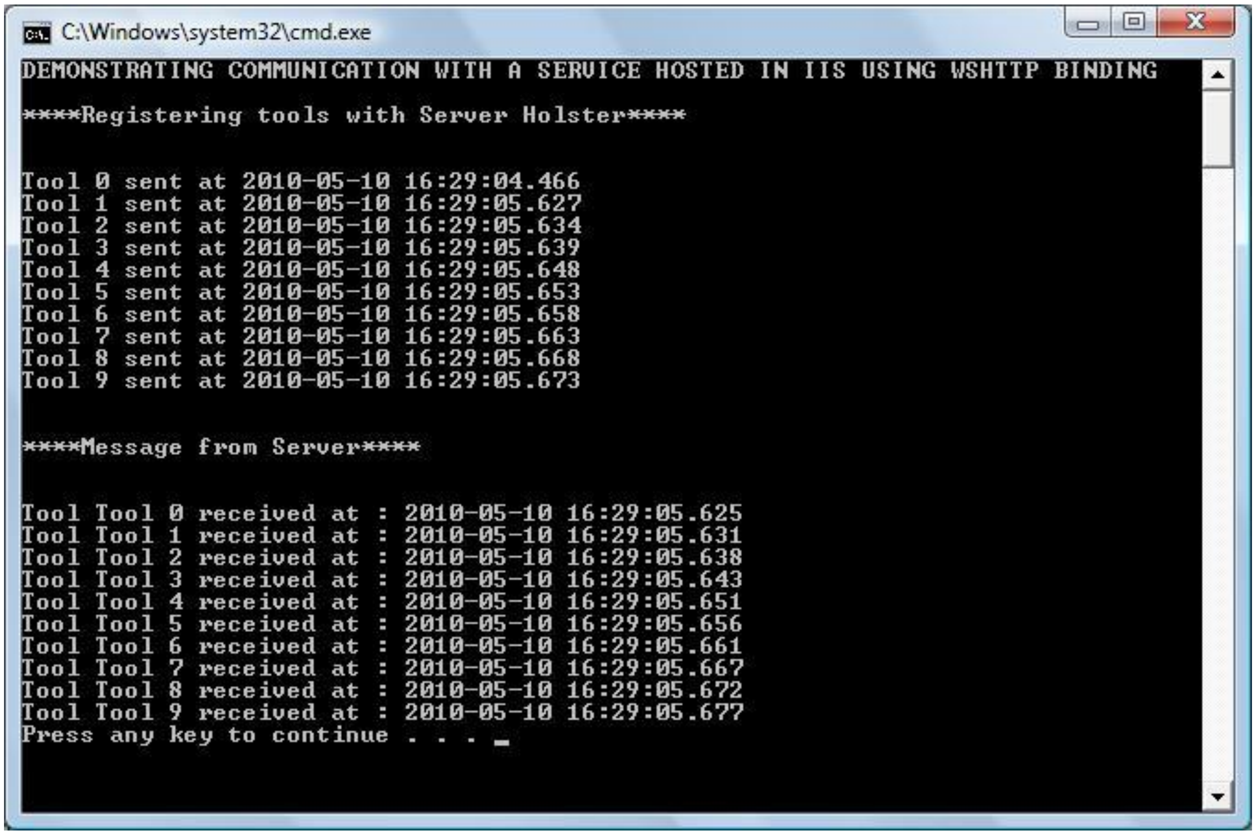
```
DEMONSTRATING COMMUNICATION WITH A SERVICE THAT IS SELF-HOSTED USING NET-TCP BIN
DING

****Registering tools with Server Holster****

Tool 0 sent at 2010-05-10 16:26:26.648
Tool 1 sent at 2010-05-10 16:26:27.211
Tool 2 sent at 2010-05-10 16:26:27.212
Tool 3 sent at 2010-05-10 16:26:27.212
Tool 4 sent at 2010-05-10 16:26:27.213
Tool 5 sent at 2010-05-10 16:26:27.213
Tool 6 sent at 2010-05-10 16:26:27.214
Tool 7 sent at 2010-05-10 16:26:27.215
Tool 8 sent at 2010-05-10 16:26:27.215
Tool 9 sent at 2010-05-10 16:26:27.216

Press any key to continue . . .
```

```
SELF HOSTING USING NET-TCP BINDING


 ToolRegister Service started
============================

 Press Enter to terminate service:

Tool 0 Registered at 2010-05-10 16:26:27.208
Tool 1 Registered at 2010-05-10 16:26:27.211
Tool 2 Registered at 2010-05-10 16:26:27.212
Tool 3 Registered at 2010-05-10 16:26:27.213
Tool 4 Registered at 2010-05-10 16:26:27.213
Tool 5 Registered at 2010-05-10 16:26:27.214
Tool 6 Registered at 2010-05-10 16:26:27.214
Tool 7 Registered at 2010-05-10 16:26:27.215
Tool 8 Registered at 2010-05-10 16:26:27.216
Tool 9 Registered at 2010-05-10 16:26:27.217
```

The performance is better when using a SELF HOSTED service as depicted in the screenshot. There is very minor difference in the time taken to transfer the message (1-2 millisecond). Only the first message consumes some time to be transferred to the service since that includes the time necessary to create channels for communication.

IIS HOSTED WS TCP BINDING:

The below screenshot represents the message transfer rates between the client and a IIs hosted service. The service implemented is a prototype service for the Tool Holster component, same as the one above, which exposes services for registration of tools. A new tool registers with the holster by calling one of its service methods. The screenshot shows the time the message is sent and time it is received.



C:\Windows\system32\cmd.exe

```
DEMONSTRATING COMMUNICATION WITH A SERVICE HOSTED IN IIS USING WSHTTP BINDING

****Registering tools with Server Holster****


Tool 0 sent at 2010-05-10 16:29:04.466
Tool 1 sent at 2010-05-10 16:29:05.627
Tool 2 sent at 2010-05-10 16:29:05.634
Tool 3 sent at 2010-05-10 16:29:05.639
Tool 4 sent at 2010-05-10 16:29:05.648
Tool 5 sent at 2010-05-10 16:29:05.653
Tool 6 sent at 2010-05-10 16:29:05.658
Tool 7 sent at 2010-05-10 16:29:05.663
Tool 8 sent at 2010-05-10 16:29:05.668
Tool 9 sent at 2010-05-10 16:29:05.673


****Message from Server****


Tool Tool 0 received at : 2010-05-10 16:29:05.625
Tool Tool 1 received at : 2010-05-10 16:29:05.631
Tool Tool 2 received at : 2010-05-10 16:29:05.638
Tool Tool 3 received at : 2010-05-10 16:29:05.643
Tool Tool 4 received at : 2010-05-10 16:29:05.651
Tool Tool 5 received at : 2010-05-10 16:29:05.656
Tool Tool 6 received at : 2010-05-10 16:29:05.661
Tool Tool 7 received at : 2010-05-10 16:29:05.667
Tool Tool 8 received at : 2010-05-10 16:29:05.672
Tool Tool 9 received at : 2010-05-10 16:29:05.677
Press any key to continue . . . _
```

When the service is hosted is IIS, performance decreases when compared to a self-hosted service. As seen in the above screenshot, time taken for a message from client to the service is 3- 5 milliseconds on an average except for the first message which consumes more time. The received time is the time the tool is registered.

Hence based on the analysis results, it could be concluded that a self-hosted services is more performant than a IIS hosted service.

Some of the prototype codes are as follows:

13.5.1 SELF HOSTED SERVICE (NET TCP)

IToolRegisterService.cs

```
using System;7
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace SelfHosting_netTcp
{
    [ServiceContract]
    public interface IToolRegisterService
    {
        [OperationContract]
        void RegisterToolWithHolster(ToolInfo toolInfo);

        [OperationContract]
        ArrayList GetToolList();

    }


    // Use a data contract to add composite types to service operations.
    [DataContract]
    public class ToolInfo
    {

        [DataMember]
        public string toolName
        {
            get;
            set;

        }

        [DataMember]
        public string toolPath
        {
            get;
            set;

        }

        [DataMember]
        public string toolDesc
        {
            get;
            set;

        }


    }
}
```

ToolRegisterService.cs - Service Implementation

```
/////////////////////////////////////////////////////////////////////////
// StreamService.cs  -   WCF Service Implementation - Prototype       //
//                                                                     //
// ver 1.0                                                             //
// Language:    C#, Visual Studio 9.0, .Net Framework 3.5              //
// Platform:    Dell Vostro 1520, Windows Vista                        //
// Application: Pr#5 , CSE681, Spring 2010                             //
// Author:      Arunkumar Manikantan, Syracuse University              //
//              (860) 796-8393, amanikan@syr.edu                        //
/////////////////////////////////////////////////////////////////////////

using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace SelfHosting_netTcp
{
    [ServiceBehavior]
    public class ToolRegisterService : IToolRegisterService
    {
        private ArrayList toolList = null;

        //----< constructor   >--------------------------------------------------
--
        public ToolRegisterService()
        {
            if (toolList == null)
                toolList = new ArrayList();

        }

        #region IToolRegister Members

        //----< Register a tool with the holster   >--------------------------
-----------------------
        public void RegisterToolWithHolster(ToolInfo toolInfo)
        {
            Console.WriteLine("{0} Registered at {1}", toolInfo.toolName,
DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
            toolList.Add(toolInfo);
        }

        //----< Return list of registered tools   >---------------------------
----------------------
        public ArrayList GetToolList()
        {
            return toolList;
        }

        #endregion

        // Host the service within this EXE console application.
        public static void Main()
        {
            Console.WriteLine("SELF HOSTING USING NET-TCP BINDING\n");
            // Create a ServiceHost for the CalculatorService type.
            using (ServiceHost serviceHost = new
ServiceHost(typeof(ToolRegist
erService)))
            {
                // Open the ServiceHost to create listeners and start
listening for messages.
                serviceHost.Open();
                // The service is now available
                Console.Write("\n  ToolRegister Service started");
                Console.Write("\n ============================\n");
                Console.Write("\n  Press Enter to terminate service:\n\n");
                Console.Read();
                Console.Write("\n");
            }
        }
    }
}
```

CLIENT INVOKING SELF-HOSTED SERVICE

Program.cs

```
/////////////////////////////////////////////////////////////////////////
// Program.cs  -   Client Console Application invoking a self-hosted   //
//                 Service                                            //
//                                                                    //
// ver 1.0                                                            //
// Language:    C#, Visual Studio 9.0, .Net Framework 3.5             //
// Platform:    Dell Vostro 1520, Windows Vista                       //
// Application: Pr#5 , CSE681, Spring 2010                            //
// Author:      Arunkumar Manikantan, Syracuse University             //
//              (860) 796-8393, amanikan@syr.edu                      //
/////////////////////////////////////////////////////////////////////////
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;
using SelfHosting_netTcp;
using System.Threading;
using System.ServiceModel;

namespace Client_SelftHost
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("DEMONSTRATING COMMUNICATION WITH A SERVICE
THAT IS SELF-HOSTED USING NET-TCP BINDING\n");
            IToolRegisterService service =
CreateSendChannel("net.tcp://localhost:8080/SelfHosting_netTcp/ToolRegisterSe
rvice/");

            Console.WriteLine("****Registering tools with Server
Holster****\n");
            for(int i=0;i<10;i++){
                ToolInfo tool=new ToolInfo();
                string toolName="Tool " + i;
                tool.toolName=toolName;
                Console.WriteLine("{0} sent at {1}",
toolName,DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
                service.RegisterToolWithHolster(tool);

            }
            Console.WriteLine();

        }
        //creates a channel for communication
        public static IToolRegisterService CreateSendChannel(string url)
        {
            NetTcpBinding binding = new NetTcpBinding();
            EndpointAddress address = new EndpointAddress(url);

            ChannelFactory<IToolRegisterService> factory
               = new ChannelFactory<IToolRegisterService>(binding, address);
            return factory.CreateChannel();
        }
    }
}
```

13.5.2 IIS-HOSTED SERVICE

The interface contract and implementation are very similar to the previous one since the same service that was self-hosted earlier is now hosted in IIS

IToolRegisterService.cs – WCF Service Contract

Service1.svc.cs- Service Implementation

CLIENT INVOKING IIS HOSTED SERVICE

Program.cs

```
/////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////
// Program.cs  -   Client Console Application invoking a IIS-hosted   //
//                 Service                                             //
//                                                                     //
// ver 1.0                                                             //
// Language:    C#, Visual Studio 9.0, .Net Framework 3.5              //
// Platform:    Dell Vostro 1520, Windows Vista                        //
// Application: Pr#5 , CSE681, Spring 2010                             //
// Author:      Arunkumar Manikantan, Syracuse University              //
//              (860) 796-8393, amanikan@syr.edu                       //
/////////////////////////////////////////////////////////////////////////
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;
using System.ServiceModel;
using IISHost_WsHttp;
using System.Threading;

namespace Client_IISHost
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("DEMONSTRATING COMMUNICATION WITH A SERVICE 
HOSTED IN IIS USING WSHTTP BINDING\n");
            IToolRegisterService service = 
CreateSendChannel("http://localhost/ToolRegisterService/Service1.svc");
            Console.WriteLine("****Registering tools with Server 
Holster****\n\n");
            for (int i = 0; i < 10; i++)
            {
                ToolInfo tool = new ToolInfo();
                string toolName = "Tool " + i;
                tool.toolName = toolName;
                Console.WriteLine("{0} sent at {1}", toolName, 
DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff"));
                service.RegisterToolWithHolster(tool);

            }
            Console.WriteLine("\n\n****Message from Server****\n\n");
            ArrayList toolList=service.GetRecvTimeList();
            foreach (string s in toolList)
            {
                Console.WriteLine(s);

            }

        }

        //creates a channel for communication
        public static IToolRegisterService CreateSendChannel(string url)
        {
            WSHttpBinding binding = new WSHttpBinding();
            EndpointAddress address = new EndpointAddress(url);

            ChannelFactory<IToolRegisterService> factory
               = new ChannelFactory<IToolRegisterService>(binding, address);
            return factory.CreateChannel();
        }
    }
}
```

14. References

1) Project#5 Requirements doc - http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Projects/ Pr5S10.doc

2) http://en.wikipedia.org/wiki/Wiki

3) Handouts and Presentations from http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/