High Performance Cross Platform Architecture for Desktop Applications

Thesis Report
Submitted in partial fulfillment of the requirements
for the award of degree of

Master of Engineering
in
Computer Science and Applications

Submitted By
ANURAG MAJI
(Roll No. 601634004)

Under the supervision of
Dr.Vinod Kumar Bhalla, Assistant Professor Thapar CSED

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
PATIALA – 147004
July 2018

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, *"High Performance Cross Platform Architecture for Desktop Applications"*, in partial fulfillment of the requirements for the award of degree of Master of Technology in *Computer Science & Application* submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Vinod Kumar Bhalla* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

Anurag Maji

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Dr. Vinod Kumar Bhalla

Assistant Professor,
Computer Science &
Engineering Department

ACKNOWLEDGEMENT

First of all, I would like to express my sincere gratitude towards my supervisor ***Dr.Vinod Kumar Bhalla*** who has supported and guided me throughout my thesis work. He has been helping me from the very starting of my M.E. work in a proper organized manner. The brainstorming sessions in his cabin are among the most worthwhile experiences I have had during my Masters studies.

I appreciate Computer Science and Engineering Department of Thapar Institute of Engineering and Technology for providing the necessary research facilities. I am also thankful to Dr. Maninder Singh (HOD) and Dr. Sanmeet Bhatia (PG Coordinator) and all the respected faculty members of the department for their teaching and guidance. I would also want to extend my obligation towards Nava Nalanda Central Library for providing access to the prominent research journals.

Last but not the least I would like to thank my parents and friends for their support and encouragement. They have been always wanted the best for me and I admire there presence for me.

ABSTRACT

Amid most recent the quantity of desktop applications has been always developing. Windows MAC OS X and Linux are three vital stages that cover almost all desktops on the planet. Building up an application includes first to pick the stages on which application will run and after that to create particular arrangements i.e. stage particular application for each picked stage utilizing stage related toolboxes.

A cross-platform application is an application that keeps running on various stages. A few structures have been proposed to improve the advancement of cross-stage applications and to lessen improvement and upkeep costs. They are called cross-stage application improvement structures. Anyway as far as anyone is concerned the life-cycle and the nature of cross-stages applications fabricated utilizing those systems have a few confinements.

In our thesis we introduces a new architecture for desktop applications which will easily allow integrating common components with the platform specific components and will be implemented in an existing application which is implemented for MAC and windows and in which we recreate the architecture for windows version of the existing application of the previous release and created high performance architecture for windows.

The application which we implemented is a backend application of the WD Discovery which is a front end portal application provides the user interface of WD Discovery application which is in Node JS and our application is WD Drive Agent which is in windows C++ for Windows and in objective C for MAC application. WD Drive Agent is faceless application.

We also explain the integration of direct-attached storage (DAS) applications i.e. WD Drive Agent with Portal Application i.e. WD Discovery. Our main goal is to first study the processes of development and maintenance of applications built using cross-platform application development frameworks, and then to create common code for existing (DAS) applications both for MAC and Windows and improve the performance of the (DAS) applications and integrate it with portal application.

TABLE OF CONTENTS

CHAPTER- 2

CHAPTER- 3

CHAPTER- 4

CHAPTER- 5

CHAPTER- 6

CHAPTER- 7

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

APIs	Application Program Interface
App	Application
DA	Drive Agent
DAS	Direct Attached Storage
DSL	Domain Specific Language
IDE	Integrated Development Environment
JSRs	Java Specific Request
MSVC	Microsoft Visual C++
NAS	Network Attached Storage
OS	Operating System
PC	Personal Computer
SMB	Simple Message Broker
TTM	Time To Showcase
UI	User Interface
UWP	Universal Windows Platform
VM	Virtual Machine
WD	Western Digital

CHAPTER 1

INTRODUCTION

Right when PCs at first ended up surely understood and started working easily at workplaces and homes, it was desktop applications that got the consideration. In any case, with the happening to the web and the online exchange impact, things changed profoundly and web applications turned into a power to be figured with. A troubling future for software applications was foreseen, and remembering that the notoriety of work zone applications was pushed aside, they never completely vanished. There were some vital complexities between the two that made it obvious that we can't overlook desktop programming application advancement.

1.1 Need of Desktop Applications

It isn't possible to supplant desktop applications since they are greatly required when you require your applications to fulfill certain criteria. We should look at some of important points for it:

1.1.1 Data Security:

Not at all like web applications, is every one of the information put away inside the client's PC framework, so there is no stress over it being hacked. The client has added up to control over independent applications and in this manner it permits security from different vulnerabilities. Web applications are available to a huge community group of clients associated through the web, and this augments the danger.

1.1.2. Available Controls: When comparison with browser based projects, desktop applications accompany various interesting intelligent controls. These intelligent controls incorporate Visual Studio for Windows and outsider controls as well as by other application. The controls additionally let you get to the basic equipment and OS segments. Furthermore, there are likewise console controls that accompany an extra layer of intelligent abilities, including the utilization of arrow keys in the keyboard.

1.1.3. Flexibility: To write desktop applications, developers can use the user's computer hardware like serial ports, camera, network ports, scanners, and Wi-Fi.

1.1.4. Performance: Desktop applications are extensively quicker and more responsive when contrasted with web applications. This is on account of web applications naturally convey overhead that you see with a universally useful web server. Then again, a desktop application, if planned effectively will stack just what's required. So they take up less memory and less assets, in this manner enhancing the execution and expanding the application's proficiency.

Building up a cross-platform application helps to save time. Developers should simply code once, and the application will keep running on all the target platforms.

This considers expanded profitability, lesser TTM (time to showcase) and a top notch application that capacities strikingly well on MAC, Linux, Windows and Android devices.

Cross-stage structures have gotten real promotion throughout the years since they hold some extremely striking points of interest over platform dependent frameworks. Some of them are:

1.1.5. Reusable code - Keeping up and deploying codes makes cross-stage application advancement a simple assignment since it absolutely stays away from repetitive undertakings. There is no compelling reason to compose a new bit of code for each activity while creating applications for all the target platforms.

1.1.6. Faster time to market - Utilizing the brought together code base makes it less demanding for organizations to make applications quicker, and send it on time.

1.1.7. Easy deployment – Cross-platform frameworks accompany a large group of modules and augmentations that make it less demanding to convey and keep up codes to make the applications keep running on every platforms. Each time we make a update to the application, it would be effectively refreshed in all the applications running on different gadgets and platforms.

1.1.8. Uniform user experience - Developers avoid potential risk to provide faultless User Experience. This is done through a solitary code base, and even the general look, feel and consistency of the application on different stages.

1.1.9. Cost-effectiveness - By utilizing a solitary code base, developers can manufacture cross-platform desktop applications, in this way making it simpler for organizations to deal with ventures inside a good budget financial plan.

1.2 Context

Currently there are more than 1 billion desktops on the planet and this number will increase in coming years. A stage subordinate desktop application is an application worked to keep running in a particular stage. As of now there are three noteworthy stages which manage the desktop showcase: windows from Microsoft, MAC OS X from apple and Linux. An extensive number of utilizations that keep running on those stages are accessible in application stores. For instance Microsoft application store is the authority application store of windows applications has in excess of 1 million applications accessible to download. A present test for business endeavors programming organizations and free work area designers is to pick the objective stages for their applications. To cover countless organizations and engineers go for discharging their work area applications on the three said stages.

More main platforms focused on includes to conceivably achieve more clients and, therefore, to expand the effect available. Nonetheless, these days, making a cross-platform application as we specified, infers the improvement of one local application for every platform to target. In this way, organizations need to manage the cost of more than one development processes of their cross-platform applications.

It has a money related effect and furthermore influences the venture association: e.g., an organization must discover particular engineers for focusing on every stage, the improvement length for every stage could be extraordinary and it could exist specialized confinements and contrasts between platforms, for example, authorization strategies. There are parts of work done in cross stage applications so our investigations additionally incorporate a short examination of in cross platform system.

1.3 Cross-platform frameworks

Last few years researches and industry sectors both concentrated for building applications in such a way to resolve the problems they are facing and the basic goals is to decrease the cost and time of creating and keeping up desktop applications.

Perchat et al. [1 2] proposed a method that permits the combination of cross-stage segments in any applications with the objective of soften the distinction between every stage. They provide various cross platform components which can easily integrated in few target platforms successfully.

1.4 Developing Cross Platform Applications

Cross-platform structures are focused for sparing advancement exertion and money and for providing advance procedure by having a solitary group constructing the entire application. Researchers had led distinctive examinations about cross-stage structures example comparison between them [3 4 5 6 7].

Our main goal in this thesis is to study various cross platform framework and implement an cross platform architecture and create an effective cross platform application direct-attached storage (DAS) applications with maximum common code in core C++ and increase the performance of the application using the cross platform architecture from the previous release of the application which was totally platform dependent.

CHAPTER 2

BAGKGROUND

2.1 Microsoft Visual Studio

For windows version of application we are using Microsoft Visual Studio for visual or windows C++ programming.

Microsoft Visual Studio is an integrated development environment (IDE) provided by Microsoft and is utilized to create PC programs, and in addition sites, web applications, desktop application and mobile applications etc.

Visual Studio enables you to write code accurately and efficiently without losing the current file context. We can easily zoom into details such as call structure, related functions, check-ins, and test status. You can also leverage our functionality to refractor, identify, and fix code issues.

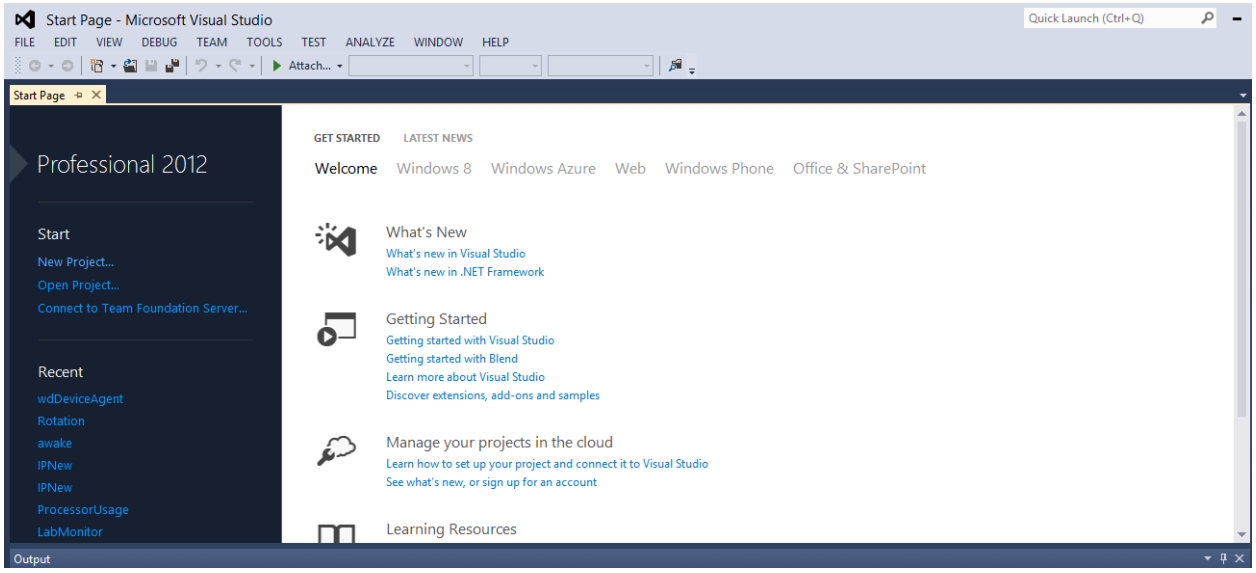

Fig.2.1. Start Page of Visual Studio 2012

2.2 Microsoft Visual C++ (MSVC)

Microsoft Visual C++ (MSVC) is a IDE from Microsoft for the C, C++ programming. MSVC is exclusive programming. It was initially an independent item however later turned into a part of Visual Studio and made accessible in both trial ware and freeware frames.

A Desktop application in C++ is a local application that can get to the full arrangement of Windows APIs and either keeps running in a window or in the system console. A Desktop application is particular from a Universal Windows Platform (UWP) application, which can keep running on PCs running Windows 10, and furthermore on Xbox, Windows Phone, Surface Hub, and different gadgets.

We are building the windows version of our application using visual C++ or windows C++ which is a faceless application and make front end application intelligent by providing the . required data by performing various operations in backend.

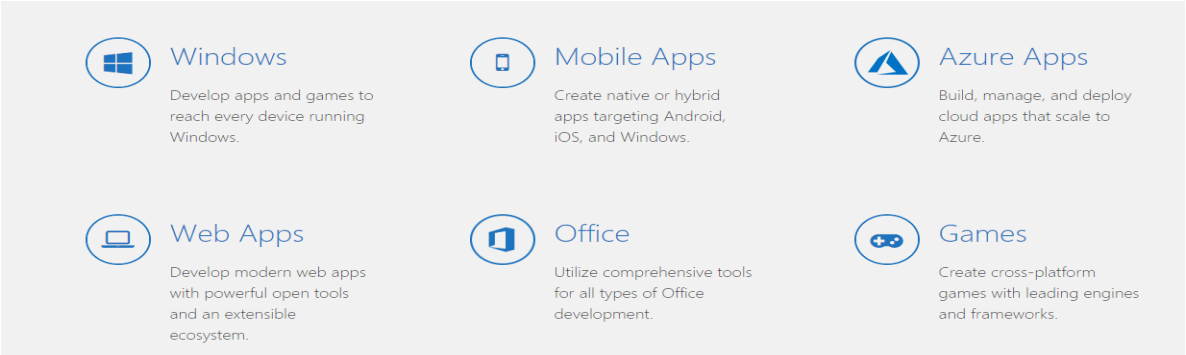

Fig.2.2. Some Features of Microsoft Visual Studio

2.3 Xcode

For mac OS X version of application we are using Xcode for Objective C programming. Code is a IDE for MAC Operating system containing a suite of programming advancement instruments created by Apple for creating programming for MAC OS X, iOS, watchOS, and tvOS. Its first version released in 2003, the most recent stable release is 10 and is accessible by means of the Mac App Store for nothing out of pocket for MAC OS X High Sierra users.

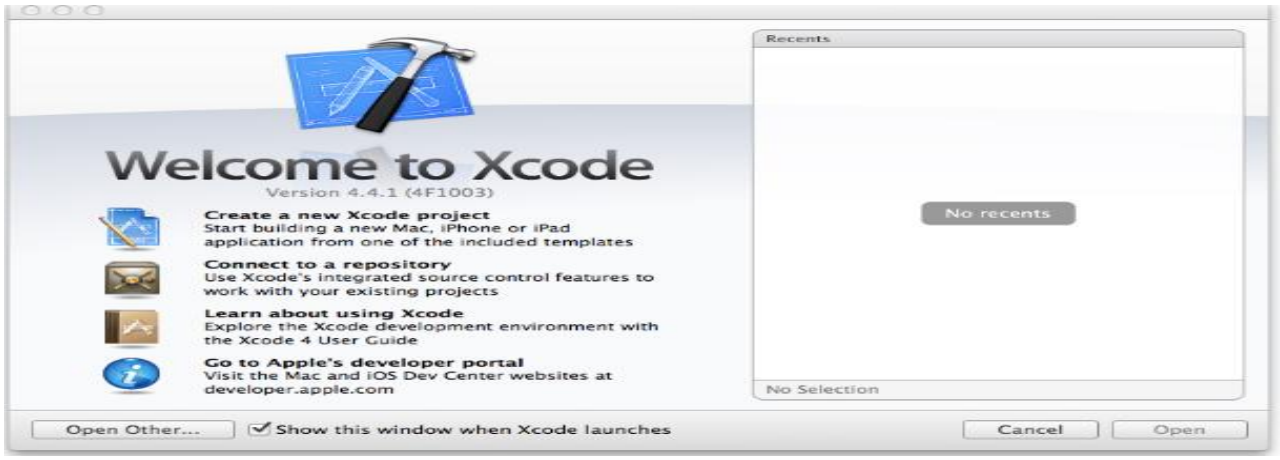

Fig.2.3. Start Page of Xcode

2.4 Objective C

Objective-c is all around helpful challenge arranged programming dialect that adds casual banter style advising to the c programming languages. It is the essential programming dialect used by the MAC OS X and iOS working systems and their APIs .It was picked as the essential instrument used by next for its working structure from which OS X and iOS are later determined.

2.5 Benefits of choosing C++ as common language between the windows and Mac OS X

We are choosing C++ for common language between windows and MAC OS X.

Even with notoriety of java c# and objective-C,C++ being the broadly utilized an ongoing report from Evans data corporation that nearly analyze overall language use found that more than 11 million expert developers know and utilize C++ routinely. JAVA and C# acquired their syntax vigorously from C++ there are a big numbers of developers who are comfortable with it, which implies finding or preparing assets for C++ wont an issue. It additionally implies there is a lot of C/C++ bolster for any usefulness we may required coordinating into a current application.

C++ was fabricated particularly for stage autonomy and in that capacity is found on each and every working operating system in presence. C/C++ code in memory of all gadgets are more than anything else .C/C++ drives a great innovation of little gadgets like the kernel which

cooperates with hardware and also with regular run time libraries. More essential for an development group is that utilization of C/C++ interfaces and libraries for anything they have to implement on any gadget and stage.

2.5.1 C++ is best compatible

A reasonable amount of the libraries accessible for gadgets are uncovered by means of c functional interfaces. One case is posix which is a standout amongst the most broadly utilized and upheld libraries for each stage since it give a typical C API to standard stage administrations. these libraries are inconsequentially simple to utilize specifically in C++ application as there is no necessary reason to make language ties or to generally do any runtime changes just #include the required header and connection the partner library and we are finished.

C++ was initially centered on vast backend frameworks in broadcast communications finance engineering and so on any industry that required ideal execution. Since C++ is a compiled language it runs straightforwardly on the CPU and is all around viewed as the best execution language. Which are the reason diversion designers adore it. The customer isn't the main level that can profit by incredible execution but middleware servers backend frameworks are all also get benefited. C++ conveys similarly too for every one of these levels.

2.6 Boost Library

We are using various APIs of Boost Library to implement our cross platform part of application. Boost allow free companion surveyed convenient C++ libraries .Boost libraries are usually important and usable over a huge number of applications .Its 10 libraries are incorporated into the C++ standards committees library technical report and these libraries are also proposed for institutionalization in C++17.

2.6.1 Benefits of using Boost

For Productivity utilization of top notch libraries like Boost speeds beginning development, brings about fewer bugs, lessens redesign of functions, and cuts long haul upkeep costs. What's more, since numerous developers are as of now acquainted with them.

CHAPTER 3

LITERATURE SURVEY

There are various ways to achieve the creation of common platform applications. Researchers presented various ways to implement it. In this section we will discuss some of the major ways the researchers are implementing the common platform architecture application for various platforms.

There are many papers on creation of common framework which provides various cross platform components to the developer so that developer does not have to go into the depth of various components and just use the components and get the desired values obtained using the cross platform components.

Fernandez Et al. [8] presented a tool features a simple user-interface that features configurable components to easily create applications, they developed this tool for the non-technical people to create their own domain-specific applications but their main concern for components for developing health monitoring system. The main goal in this paper to provide the power of app development to the non technical people without going in the depth of the code by providing pool of tools that allows users without programming experience to build customized applications easily.

Friedrich Et al.[9] presented the report which represents technical survey of some of the approaches associated to operating systems that have been used in order to fulfill needs to be focused on software infrastructure, such as operating systems, with respect to how to provide functionality in order to fulfill the requirements.

Perchat et al.[2] also created common framework for cross platform applications that, they have defined a framework called COMMON(Component Oriented programming for Mobile Multi Os integration).This framework allows the integration of cross-platform components in any

application currently they implemented it for iOS and android. They provided a common public interface of each component to make integration easier, which is platform-independent. Using, this framework they saved 30% of line of code.

One of the ways to achieve is cross-compilers which empower the developers to compose their applications by a common programming language between each objective stage. in this situation they create the related local code for every one of them for this situation the reused code is finished however the mapping the common languages APIs and all the local target language APIs is extremely hard to accomplish. that is the reason much of the time cross-compilers as it were oversee couple of stages and are restricted to normal components from every stage.
This is the case of MoSync, (http://www.mosync.com/) Corona (http://www.anscamobile.com/) and Neomades(http://neomades.com/).

To resolve this mapping problem we created a simplified architecture, where we put the platform specific code to specific locations. We have a abstract super class for platforms which contains the all the member functions and variables, these member functions parameters and member variables values are filled by the platform specific API's which are located to specific location but returns the similar values for each platform. Then these values returned to common code and they use these values according to need.

One other piece of existing arrangements depends on model driven engineering. With these sorts of arrangements developers can characterize theirs applications from models for a few target platforms, then these models can be converted into native code for each target stage. in any case similar to cross-compilers the interpretation among models and local source code is hard to accomplish particularly if the arrangement suppliers need to deal with any local segment. On one hand usixml by vellis et al.[10] and jelly by meskens et al.[11] enable developers to create UI for numerous portable stages.

Interpreters translate, continuously with a devoted engine, a source code to executable directions. Developers execute their cross-platform application, the Interpreters deals with their execution on numerous platforms. For this situation, the interpreter developers must actualize a module ready to decipher the code for each target stage. We can recognize two classes in the versatile

mediator space: Virtual Machines (VMs) and arrangements in view of web languages. The most renowned innovation in light of VM is Java ME. In any case, this innovation is disliked and isn't utilized by developers because the discontinuity of gadgets furthermore, working frameworks is constantly present and even stressed with the large number of existing JSRs (JSRs are Java Specification Requests, basically change requests for the Java language, libraries and other components) in which the application advancement is based.

Kramer *et al*. (2011) [12], created a common Domain Specific Language (DSL). After, the applications written with DSL can run on a VM.

We can implement web based application to provide common application for every platform.
Today, web languages are open to everybody that is the reason a few arrangements in light of it have developed. Numerous methodologies were characterized for web applications. One of them permits transferring, on the gadget, of web applications, which can be contrasted and portable sites having the capacity to get to the gadget hardware. Few of web application solutions are presented by Allen *et al*[13].

But available features with web application are often limited. For instance, it is frequently conceivable to utilize the camera with a specific end goal to take pictures or on the other hand recordings yet it is difficult to exploit its stream.

A few arrangements propose to utilize the cloud as an application stage (Mikkonen and Taivalsaari,[14]).
The principle objective is to designate certain parts of an application to the cloud. For instance, an application that permits confront acknowledgments will be isolated in two sections. The initial one, executed in the gadget, permits catching pictures from the camera stream and the second one will handle the face acknowledgment in the cloud.
Objective of this solution is to spare device energy in disseminating the overwhelming procedures on servers and furthermore to give new highlights to portable applications (Zhang et al.[15]).
With μCloud by March et al.[16], the developers must isolate their applications into numerous components. Every component is arranged by its area: Cloud, mobile or hybrid. At that point, at

run time, a conductor arranges the application execution. Once component which runs on cloud are developed, those component can be reused by any applications. However this solution won't work if the application is not connected to the cloud.

Matias et al.[17] gives the brief investigation of the procedures of improvement and upkeep of utilizations assembled utilizing cross-platform application advancement systems, concentrating especially on the bug-fixing action and after that, they go for characterizing tools for automated repairing bugs from cross-platform applications.

Matias additionally examines the procedures of creating cross platform utilizing cross compiled development frameworks with regard to the platform dependent applications utilizing tools for that propose (Xcode, Android Studio, Visual Studio) and examine faster time-to-market, increase component reuse, decrease cost, maintenance etc.

CHAPTER 4

PROBLEM STATEMENT

4.1 PROBLEM STATEMENT

As we see previously in literature survey section many of the researchers had implemented the cross platform framework and use their components in various platforms, but there will be some major disadvantages of using frameworks.

The programmer is in danger by depending on the framework i.e. losing understanding of the functionality internally. When he implement it the arrangement of the structure and the profitability drops ideal off and can be hard to add anything outside of the framework. The developer who is utilizing it needs to use things in the way that the developer of the framework implemented it. For example if developer have to implement the same operation using different parameter types or requires implementing the task differently it will difficult for him as he is bounded with the framework. In this way the power of the developer who is using the architecture can be reduce as he will bound with the components of the framework.

In this thesis we are defining a common cross platform architecture for desktop application between various platform so that we can separate the platform dependent code and cross platform code optimistically and use some standard libraries like Boost whose major parts is also cross platform and provides better results.

4.2 Proposed system

Aim of research, in this thesis is to study various cross platform frameworks and to design efficient cross platform architecture for desktop applications resulting in the higher reusability in case of cross platform development. Researchers developed a Direct-Attached Storage (DAS) application as model case study. Same methodology can be applied to other desktop applications to reap the benefits of proposed architecture. Redesign of proposed DAS application's component layer will have maximum common code in core C++ to get increased performance of

the application in comparison with the existing versions (Windows OS, MAC OS X) of the previous release of the DAS application which was totally platform dependent.

4.3 Objectives

- To study the architecture of windows, Mac and Linux.
- To study the cross-platform architecture of the existing DAS application.
- To develop the high performance cross-platform architecture for new version of DAS application and compare the performance with existing DAS application.

4.4 Motivation

Previous release of DAS application had two versions, one for windows and other for MAC OS X. Windows code is written in windows C++ or Microsoft visual C++ (MSVC) and MAC OS X code is written in objective C. In this case most of the code was machine dependent. To enhance and maintain both versions development team has to put huge efforts to get the same performance from both versions. Component layer in the existing software had fewer reusability of code because of platform dependent issues. Researchers thought of improving the situation by redeveloping the Component layer for cross platform development to increase the reusability of code, to get higher performance, easy maintenance and enhancement of application. This could overall result in saving time, effort, resources and money. All these factors motivated the researchers to come up with a better cross platform architecture design for component layer.

4.5 Methodology

We studied various C++ libraries and experimented with them to create the common code.

We designed the architecture of the project in such a way it separates the common code and platform dependent code effectively and architecture is very scalable and we can easily extend our project with other operating systems easily according to our need.

Our main goal is to minimize the platform dependent code as much as possible with common code and improve the performance of the existing windows architecture.

CHAPTER 5

IMPLEMENTATION

5.1 COMMON ARCHITECTURE

We presented the component oriented programming for the desktop applications. Our architecture permits the developers of cross-platform applications to integrate cross-platform segments in any platform desktop application. With this architecture, developers must implement the minimal structure of their application, with the platform dependent API's of each objective platform. Along these, developers must give the execution of their applications structure with the windows, then with the MAC OS X and the various target platforms. By executing the application structure with native platforms APIs, we enable developers to give the most ideal user experience for their applications for each host platform. For sure, on every platform, they will have the capacity to utilize any component particular to every local APIs.

Our architecture allow cross platform components to integrate easily with the application.

A substantial part should be reusable in a other platform versions of the application. In addition, the huge part of the application will stage autonomous. To do that, we have characterized the structure of our segments as appeared in **Fig**. 5.1. A part has one usage for each objective stage: Implemented with the windows, at that point with the MAC OS X et cetera. By giving local executions, our parts can utilize any equipment or programming component of any stage. In this way, our parts, running in foundation, will be splendidly incorporated in a local application without modifying the platform dependent part of the application.

We need to have stage autonomous segments. These segments are composed in C++ in light of the fact that it is a sufficiently adaptable language that does not rely upon any stage. Accordingly we permit the combination of segments specifically in the local source codes i.e. visual C++ for windows and Objective C for MAC OS X.

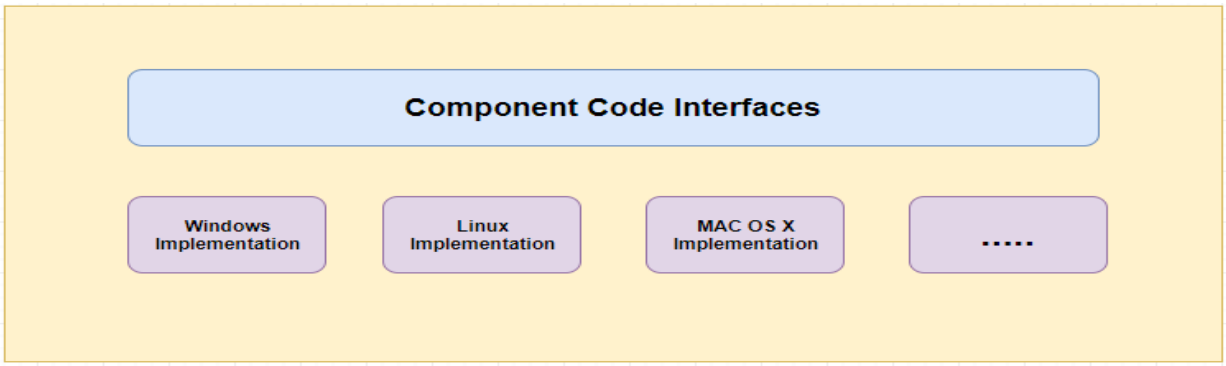

Fig. 5.1. Component structure

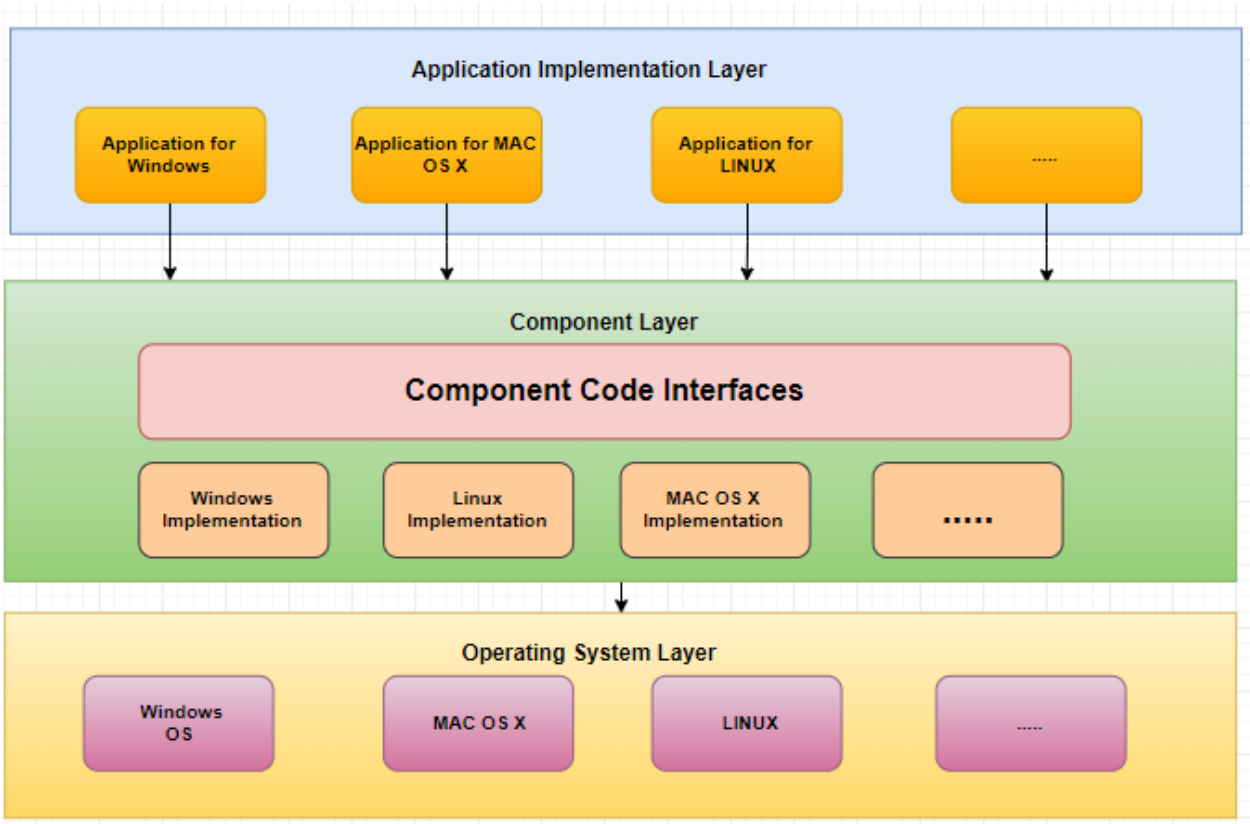

Fig. 5.2. Component-oriented Architecture to create cross-platform applications

As displayed in **Fig. 5.2** Developers define only once the use of a method and reuse it in any native application.

We have use a cross-compiler language i.e. C++ can easily integrate with native languages i.e. windows MAC OS X and Linux to sum up a developer using our architecture developers can implements the minimal structure of its applications with the platform dependent APIs of each required stages. In the next section we are going to present our solution to implement application.

5.2 COMMON ARCHITECTURE IN SITUATION

To validate our solution, we have chosen to develop effective cross platform application direct-attached storage (DAS) applications with maximum common code in core C++ and increase the performance of the application from the previous release of the application which was totally platform dependent. This application is a concrete application with a professional style and an advanced user experience. The goal is to provide a deployable application for desktop users.

Our project describes the integration of direct-attached storage (DAS) applications with Portal App. The existing DAS applications will need to be altered to communicate with Portal App, but for the near term the DAS applications will keep most of their existing user interfaces elements. In the longer term more parts of the DAS applications user interface or possibly the entire user interface will be moved to Portal App.

Portal App is a browser-like application that serves as the user's starting point for all WD DAS and NAS related software and promotions. This document will not describe the design of Portal App. Instead this document will focus on the communication channel between DAS applications and Portal App and a few other internal design issues relating to integration of the DAS applications with Portal App.

Most of the DAS applications exist in a form where they have their own user interface or they cooperate with a component called "App Manager" to present their user interface. Portal App is new and requires some changes to how the DAS applications handle their user interface. To reduce the development schedule the basic design of the existing DAS applications will be retained and only modified to incorporate the new interaction with Portal App for communication and certain portions of the DAS application's user interface.

The **Fig.5.3.**Represents the screen shots of our front end application Discovery which is in build in node js and it is cross platform application.

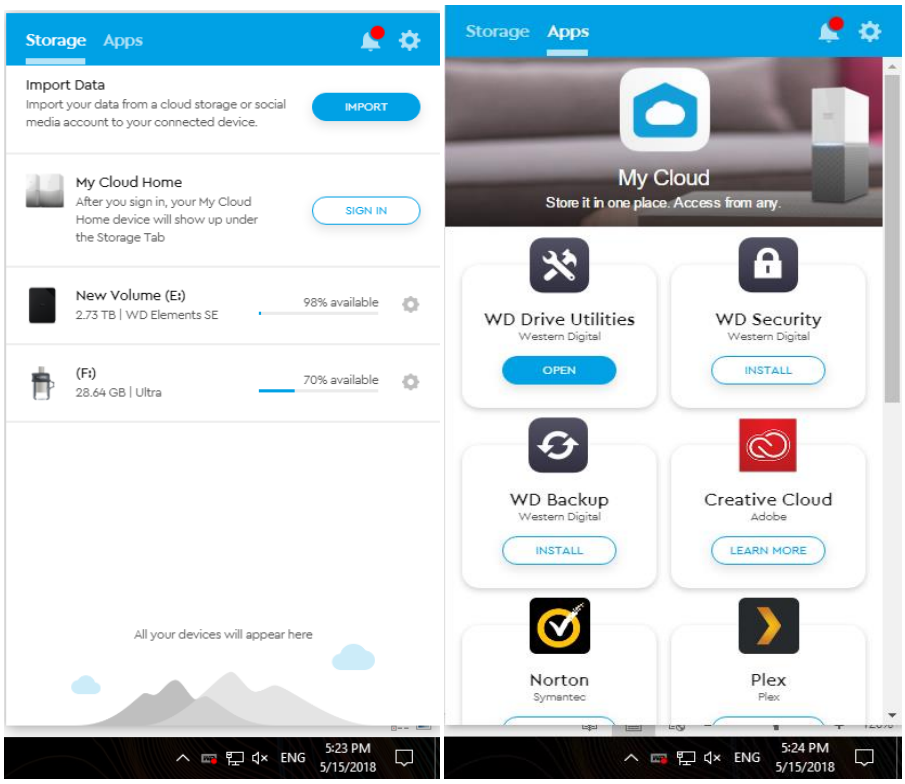

Fig. 5.3. WD Discovery

WD discovery is a front end application for which WD drive agent is providing the intelligence by performing various operations and updating the WD discovery by passing the information by IPC communication (presently PIPE for windows and SOCKET for MAC).

Front end application also allows users to download the WD and third party software as shown in **Fig. 5.3.**

5.3 Block Diagrams

The diagram below shows the major components of the DAS applications and how they interface with Portal App.

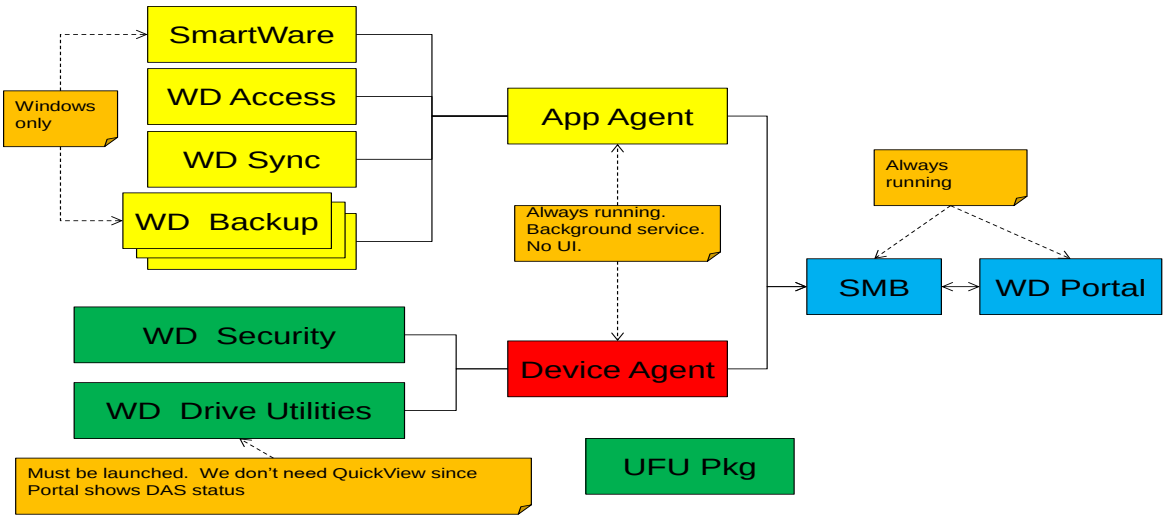

FIG. 5.4. DAS APPLICATION COMPONENTS

Below is a short description of each box in the diagram above:

- The blue boxes (i.e. "SMB" and "WD Portal") are the Portal App main components.
    - The SMB is the "Simple Message Broker" and is responsible for message sending, receiving, queuing, and communication channel management.
    - The "WD Portal" portion is responsible for the user interface of the Portal App
- Device Agent is a background service or faceless app (depending on operating system requirements) that listens for device arrivals, removals, and status changes. The Device Agent has no UI. The Device Agent performs all communication with Portal App for device related notifications. We expect to keep Device Agent and App Agent as separate items for the foreseeable future. We intend to use library code for Portal App communication to avoid duplicating that code in separate programs.
- WD Drive Utilities is primarily just UI that allows the user to interact with devices found by Device Agent

- WD Security is primarily just UI that allows the user to unlock a device found by Device Agent
- UFU Pkg is a firmware update installer package. It is standalone in its design, but for integration with Portal it is capable of being launched from a Portal App button action. The UFU firmware update does not directly communicate with Device Agent or Portal App. Instead a successful firmware update process triggers a re-enumeration of the device which causes Device Agent to send a new device post message to Portal App for the newly updated device.
- App Agent is a background service or faceless app (depending on operating system requirements) that serves as a manager of various Client App plug-ins. The App Agent is an upgraded version of the old App Manager using the same plug-in technology as before, but now capable to delegating the UI to Portal App. The App Agent performs all communication with Portal App for each of the Client App plug-ins.
- SmartWare is an existing Client App that should soon be depreciated and replaced with new functionality in WD Backup. Until that depreciation is complete there might be a need to integrate SmartWare with Portal App at some rudimentary level of functionality.
- WD Access is a Client App plug-in that allows the user to easily find their NAS box and also allows user to copy files to that NAS.
- WD Sync is a Client App plug-in that allows the user to specify folders to be synchronized between DAS devices, NAS devices, and the user's host computer
- WD Backup is a Client App plug-in that allows the user to define and execute backup plans to create archives of the user's data from a disk volume to an archive. The disk volumes and archives are allowed to be on DAS devices, NAS devices, or the user's host computer. The user is also allowed to restore an archive back to a disk volume.
    1. This Client App is only available to Windows
    2. The WD documentation directs Mac OS X users to use the Apple supplied Time Machine backup utility

5.4 Proposed architecture

Previous release was same project with two applications, one for windows and other for MAC.

Windows code is written in windows C++ and MAC code is written in objective C++. We go through various C++ libraries and experimented with them to create the common code.

We designed the architecture of the project in such a way it separates the common code and platform dependent code effectively and architecture is very scalable and we can easily extend our project with other operating systems and can add more components easily according to our need.

We changed the previous release architecture for windows application and make it similar to Mac architecture which makes its performance very fast which was previously slow as we rectify one layer.

Our main goal is to minimize the platform dependent code as much as possible with common code.

The user interfaces also developed cross complier language which is in Node.js. Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

This application is developed by other team which is the portal application, our architecture will be implemented on the backend DAS application, which is a background service or faceless application (depending on operating system requirements) that listens for device arrivals, removals, and status changes.

Discovery is just the front end application to make it intelligent the backend DA Application plays a major role and performs all the operations and passes the required values to the Discovery using IPC communication methods and Discovery displays the information according to the needs.

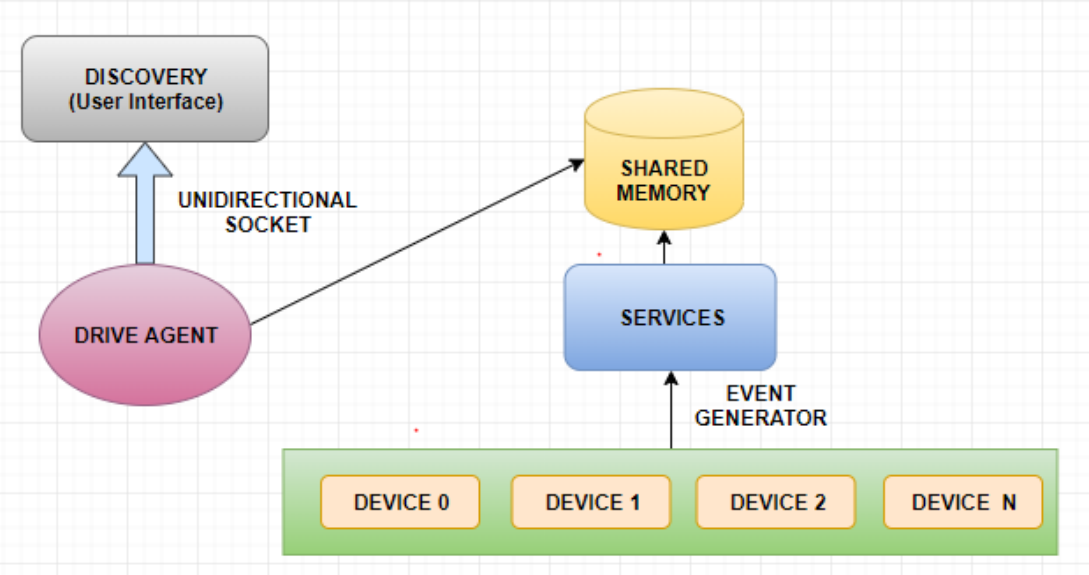

Fig. 5.5. Old Architecture of the backend application in windows

Fig. 5.5 Represents our old application architecture in windows. Where Discovery is the portal front end application and drive agent is the backend faceless application which fetches the information from shared memory, where data is in the form of JSON messages stores by the services which monitors the all the actions of the devices and listens for device arrivals, removals, and status changes.

Shared memory used by the windows was COM. Use of shared memory in our windows application was one of the factors that our application for windows was slow.

We have to create a separate program for service in windows. The service continuously pools the device layer for new event and stores the information to the shared memory. Later Drive Agent will fetch the data from the shared memory and perform the required operations using the data and send the data to the Discovery using unidirectional Socket and Discovery will display the data to the appropriate places according to requirement.

So one of the major requirements of the new architecture is to improve the performance of our windows application as its performance is not that much good as compared to our application works under MAC operating systems.

In our new architecture for windows we rectify the use of the service by adding folder watcher API which will only active when any change occurs to our desired location which rectifies the requirement of continuously polling for event generation. Because of removal of service and

shared memory there is a huge impact of performance of our windows application. **Fig**.5.6 shows the new architecture of our windows application in which we remove shared memory.

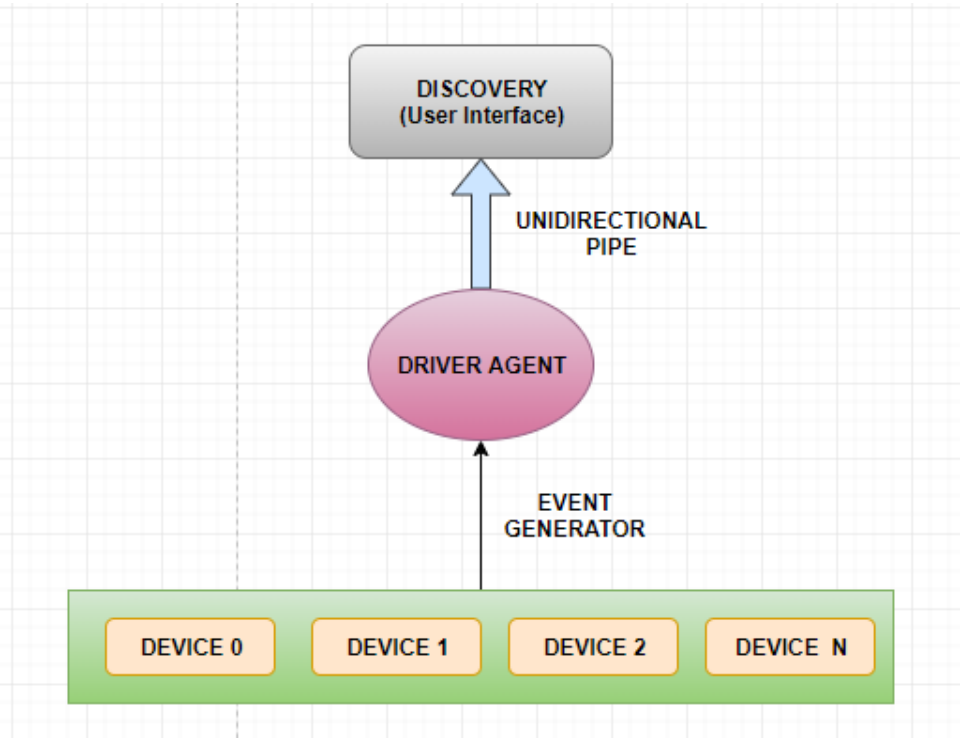

Fig.5.6. New Architecture of the backend application in windows

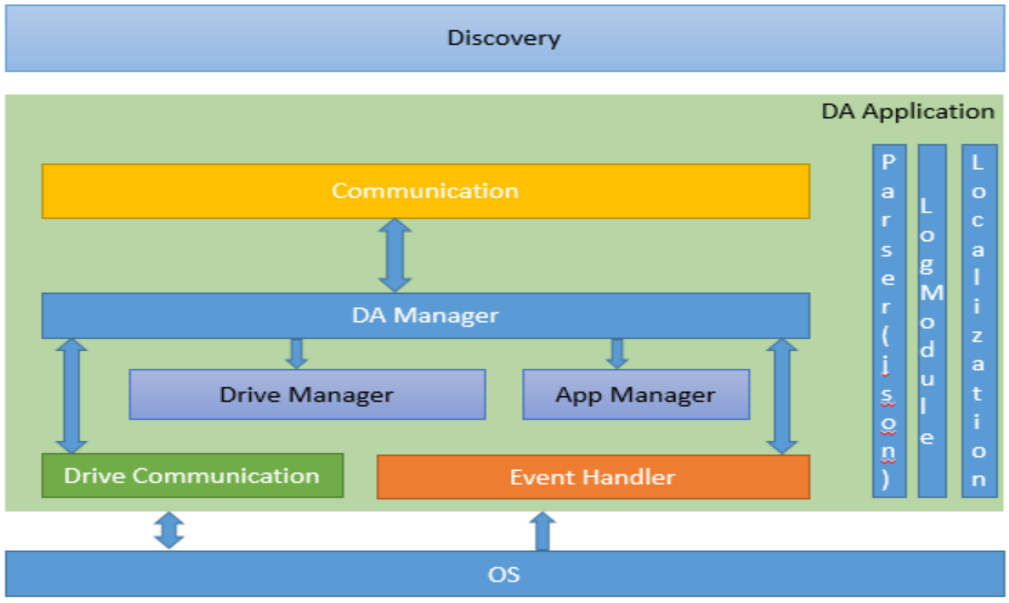

Fig.5.7. Component diagram of the backend DA application

The first layer Discovery is the front end application second layer represents the DA application layer.

1. Communication layer: It is the layer which acts as a bridge between Discovery and the drive agent manager. The drive agent manager which get the details from the drive manager and the app manager .The app manager contains all the details of the application which the user wants to download .Right now we have few applications(i.e. WD security ,WD Backup, WD Drive utilities.. etc) .
2. DAManager is one of the main and central component of our application which in C++ i.e. cross platform language for MAC and windows operating system. DAManager call the functions from the different components pass the obtained values to other component to get the desired values and later pass them to Discovery using JSON messages.
3. Drive Communication: This layer contains the APIs through which hard drives communicate with the applications in our application we are using SCSI (Small Computer System Interface) protocol. It is an arrangement of models for physically associating and exchanging information among PCs and fringe gadgets.
4. Event Handler: In this layer our application polls the operating system for discovery of devices and any operations related to devices and applications to occur according to the poll time continuously and generate an event and report it to Drive Agent Manager.
5. DA Manager filters the type of event either for application or for devices and passes the information to the device manager and application manager and get back the desired values for operation.
6. Localization layer: In this layer we localize the application according to the user's native language and pass the display messages of the discovery to local texts, so that user can easily use the instructions and features of the application.
7. Parser Layer: This layer deals with the parsing of JSON messages, JSON (JavaScript Object Notation) is a lightweight information exchange design, all the layers of our application communicate with each other by generating the JSON files and passes them to the other layers which are expecting the data from that layer.

8. Log Module Layer : This layer contains the Log implementations in our case we create the singleton implementation of Log so it create only one instance of our log class in all over the application and implementation of rotation of the log based on various factors like size and time.

Most of the implementations of the components are in C++ now in this component diagram. Only the implementation of Drive Manager and Application Manager Layers contains the platform dependent APIs. Other components are cross platform components written in C++ so, their implementations and source code will be same so, there is a huge reduction of numbers of lines in source code.

5.6 Integration:

This part will describe the integration of the components we previously we discussed for desktop application. To understand the integration we are using the class diagram of the few main classes which will connect all the components including platform dependent and cross platform functions. In windows we are using windows C++ which is platform dependent and C++ which is cross platform and for MAC we are using objective C++ i.e. platform dependent and C++.
So this class diagram helps to understand the main skeleton or the architecture of our application. We will explain the some of the main classes and show some of their declarations part to understand the integration.

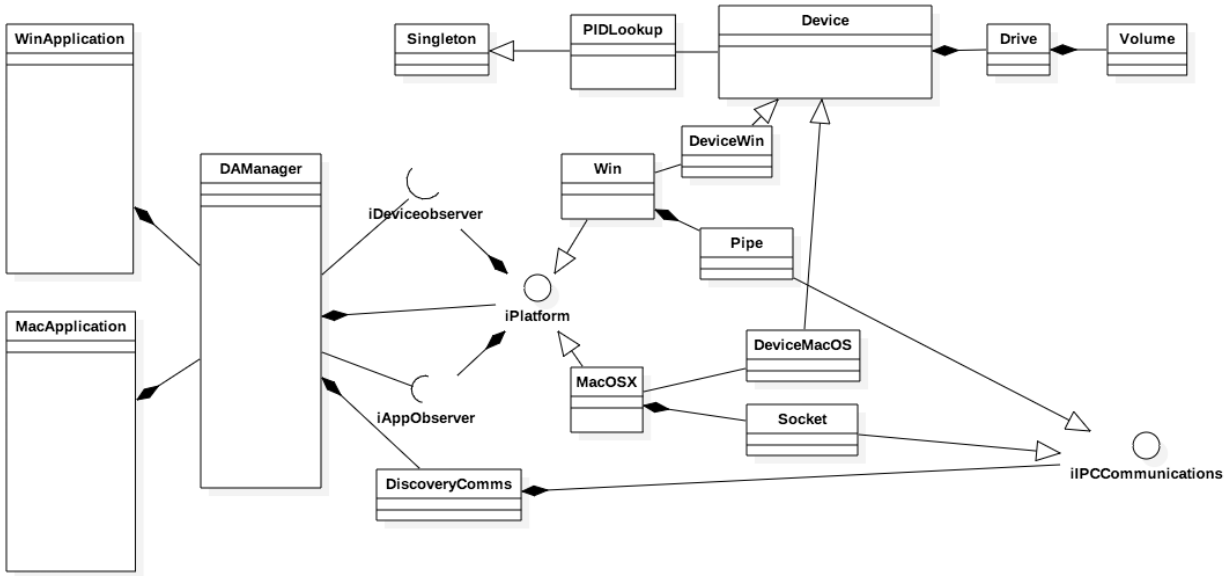

Fig. 5.8. Class diagram of the backend DA application

The **Fig**.5.8. Diagram represents the class diagram of our DAS board backend application. Here WinApplication and MacApplication are the main entry point for the application which is platform dependent code.

The Win, Mac OS X, DeviceWin,DeviceMacOS are the main class containing the platform dependent APIs, rest of the classes are the cross platform components. So for every platform our architecture containing only few files one for starting or entry of application (i.e. winApplication for windows and Mac Application for MAC)and few files containing all the platform dependent API's (i.e. win for windows and Mac OS X for MAC).For windows we are using pipe as it was previously socket to increase the performance of application in windows and for MAC we are using UNIX socket same previously used for windows .We implemented the code for socket cross platform which is in C++.

Fig.5.8 represents the class diagram of few main components of our application and gives the basic idea of integration of all the components in our application. The class diagram contains both platform dependent and cross platform components.

The main entry of the application is starts with WinApplication for windows and MacApplication for Mac.

DAManger is the main class of the program which is common to all the platforms and calls the functions from other files step by step according to the requirement.

For windows we use WindowProc callback function which is an application-characterized function that procedures messages sent to a window.

Fig.5.9. shows the initialization of g_platform which is an reference of WinOS and initialization of g_appManager which is a reference to DAManager.

In DAManager constructor we pass the platform which is currently windows so that using g_platform we can call the platform dependent functions in DAManager and use the values we obtained by the windows APIs.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HDEVNOTIFY hDeviceNotifyDISKGUID;
    static HDEVNOTIFY hDeviceNotifySAMGUID;
    static HDEVNOTIFY hDeviceNotifySESGUID;
    switch(message)
    {
    case WM_DEVICECHANGE:
        if (g_platform)
            g_platform->handleEvent(hWnd,wParam,lParam);
        break;
    case WM_CREATE:
        g_platform = new WinOS();
        g_appManager = new DAManager(g_platform);
```

Fig. 5.9.A screenshot of WinApplication class

Similarly for MAC our application starts from MacApplication. In MAC we use AppDelegate to launch an application. AppDelecates object gets notified when the connected objects reaches certain stages. In our case when application did finish launching we initialize the platform which is reference to MacOSX class and appManager which is reference to DAManager and we pass platform to its constructor to get the required values generated by Mac APIs. **Fig**.5.10. is the screen shot of the declarations of the platform and DAManager in MAC using objective C.

```
- (void)applicationDidFinishLaunching:(NSNotification *)notification
{
    platform = new MacOSX();
    appManager = new DAManager(platform);
}
```

Fig. 5.10. A screenshot of MacApplication class

DAManger inheriting two abstract classes iDeviceObserver and iAppObserver both contains pure virtual functions related to device and application respectively.

Fig.5.11. shows the snapshot of the iAppobserver and iDeviceObserver class and few pure virtual functions in them. As DAManger is their derived class the definition of those pure virtual functions will be define by the DAManger.

```
class iAppObserver
{
public:
    virtual void Update(void) = 0;
};

class iDeviceObserver{
public:
    virtual void Update(Device* device, DeviceEvent deviceEvent) = 0;
};
```

Fig. 5.11. iAppObserver and DeviceObserver class

iPlatform is an abstract class which contains all the pure virtual functions then this functions defined by the derived classes. The main purpose of the pure virtual functions is always to mandatory for the derived classes to have definitions of those functions. In our case **win** and **Mac OS X** are the derived classes of **iPlatform**. **Fig**.5.12. shows a screen shot of few pure virtual functions in **iPlatform.**

The main motive of the **iPlatform** is to implement run time polymorphism as we have two derived classes of it i.e. **win** and **MAC OS X** so pure virtual functions of the **iPlatform** have to be defined in those derived classes. As win and MAC OS X contains the platform dependent APIs but we required the same signature as this functions returned the same parameters values using platform dependent API's so this architecture helps and mandatorily helps to define those functions.

```
class iPlatform
{
public:
    virtual void SubscribeDeviceObserver(iDeviceObserver* observer) = 0;

    virtual std::list<iAppDetails*> SubscribeAppObserver(iAppObserver* observer) = 0;

    virtual void SubscribeTimer(const int& timerId, int secs,void (*callback) (void*), void*) = 0;
    virtual void UnsubscribeTimer(const int& timerId) = 0;

    virtual iIPCCommunications* GetCommunicationChannel(void) = 0;

    virtual void GetLogFilePath(std::string& logFile) = 0;
```

Fig. 5.12. iPlatform showing few pure virtual functions in it

The abstract class has property that we can't create the object of that class but we can create reference of that class. In our case as **win** and **Mac OS X are** the derived classes of **iPlatform** which inherited the same functions as virtual functions.

```
class WinOS : public iPlatform
{

public:
    WinOS();
    virtual ~WinOS();

    void handleEvent(HWND hWnd, WPARAM wParam, LPARAM lParam);

    virtual void SubscribeDeviceObserver(iDeviceObserver* observer);

    virtual std::list<iAppDetails*> SubscribeAppObserver(iAppObserver* observer);

    virtual void SubscribeTimer(const int& timeId, int secs, void (*callback) (void*), void*);
    virtual void UnsubscribeTimer(const int& timeId);

    virtual iIPCCommunications* GetCommunicationChannel(void);
```

Fig. 5.13. WinOS inheriting the base class pure virtual functions as virtual function

As **win** and **Mac OS X are** the derived classes of **iPlatform** we have mandatorily define the pure virtual functions **Fig**.5.14 represents the definition syntax of a function.

```
void WinOS::SubscribeDeviceObserver(iDeviceObserver* observer)
{
    m_deviceObserver = observer;
    enumerateDevices();
}
```

Fig. 5.14. Definition of an inherited pure virtual function

Similarly we will define all the inherited pure virtual functions.

```
LOG(LEVEL_INFO, "Initializing Device Agent");

m_discoveryComms = new DiscoveryComms(m_platform->GetCommunicationChannel(), m_platform->GetLocaleFolderPath(),
                                      m_platform->GetDiscoverySettingsFilePath(), m_platform->GetSystemLocale());

m_platform->SubscribeDeviceObserver(this);
m_appsDetails = m_platform->SubscribeAppObserver(this);
```

Fig. 5.15. Screen shot of accessing the virtual functions definitions

In **Fig**.5.15. m_platform is the pointer to the abstract class WinOS and through this pointer we will access the definitions of the WinOS functions as WinOS is an abstract class and we can't create the objects of the abstract class only references are allowed for abstract classes.

iIPCCommunications is an abstract class which is used to send and receive the messages in JSON format in our application between various components and Discovery. It also contains the pure virtual classes which is inherited by the derived classes pipe and socket. We are using pipe for windows and socket for MAC.

```
class iIPCCommunications
{
public:
    virtual bool Init(void) = 0;
    virtual bool Send(std::string &str) = 0;
```

Fig. 5.16. iIPCCommunications class

Fig.5.16 shows the declaration of iIPCCommunications and few of its pure virtual functions.

Fig.5.17. represents the Pipe and Socket classes respectively. Both the classes are the derived classes of iIPCCommunications so they are defining all the pure virtual functions of their bases classes separately.

```
class Pipe : public iIPCCommunications
{
private:
    std::string m_pipeName;
    HANDLE  m_pipeHandle;
public:
    Pipe(std::string &connectionName);
    ~Pipe();

    bool Init(void);
    bool Send(std::string &strJSONContent);
    bool Receive(std::string &strMsgOut);
```

```
class Socket : public iIPCCommunications
{
public:
    Socket(std::string &homeDirPath);
    ~Socket();

    bool Init(void);
    bool Send(std::string &strJSONContent);

private:
    int         m_sockFD;
    std::string      m_socketFullName;
```

Fig. 5.17. Pipe and Socket classes

In similar way Device is the base class of DeviceWin and DeviceMacOS and also contains some pure virtual functions which will define by the derived classes separately. In this way we are defining those pure virtual functions of base classes to platform dependent code using platform dependent APIs in derived classes function. Pure virtual function ensures the mandatory definition of those functions in each platform so that our application can get the required values and pass them to DAManager which is cross platform code and provides the respective functionalities according to requirement of our application.

At the end when our application is completed we get executable files for windows and mac os x.

To compile and build those files we are giving the paths of common code locations and native code locations.

After the executable created we are placing it to common repository from where WD Discovery will fetch the executable file and use it without installation.

CHAPTER 6

RESULTS & ANANLYSIS & DISCUSSION

6.1 RESULTS

Before starting our design execution for a few platforms we needed to compute the required extra cost on applications utilizing using our architecture like performance,cost,memory utilization,week points. To implement that, we have chosen windows and MAC which are the most demanding operating systems in market. We have developed the WD Drive Agent backend application currently for windows and MAC desktops. Previously it was build in Windows C++ and Objective C++ for Windows and MAC separately, now by implementing our architecture we generated more common code in C++ which helps to reduce line of codes and make our application more scalable and easy maintenance. We presented our new version in previous sections.

In the next subsections, we have first calculated the lines of code saved using our approach, compared to the platform dependent application. Secondly, we have compared the performances regarding the previous versions of WD Drive Agent.

6.1.1 Lines of Code Saved

We have compared the number of lines of code written in the native version and in the version with our architecture, see **Table 1**.

Application version Lines of code Saved	**Platform Dependent Code**	**Platform Independent Code**
100% Platform dependent for Windows	**6500**	**0**
Application with our architecture	**4100**	**2800 (40% approx.)**

Table.6.1. Lines of code written for each application

Using our approach, the developer writes two thousand eight hundred lines of code less than by using the platform dependent architecture. Besides, the parts, which are not implemented, are often the most complex ones. With our approach, the developer only needs to implement the platform dependent part of application in all the targeted platforms.

6.1.2. Performances

Here we can look at the execution of a few modules between the present and past adaptations of our application. The thought about errands can be executed for numerous platform version of the application and not simply in one platform of our application. Subsequently this correlation can be considered as platform-autonomous. Anyway it is compulsory to assess our approach from a genuine application and not simply from a test or fundamental application on the grounds that a genuine application utilizes a greater number of assets than a test application.

As appeared in **Fig**. 6.1 we have estimated the execution time of couple of modules actualized in the WD Drive Agent application. For every module we have estimated the execution time in milliseconds. at that point for every module we have figured the normal between comes about. We have obtained these outcomes on the windows operating system on dell desktop and with mac we are using macbook laptop.

The installed windows version on the device is Windows 10 64 bit and for MAC Sierra.

The Table.3. Shows the configurations of desktops on which we are generating and analyzing our application.

	Windows	**MAC**
RAM	16GB	8GB
HDD	465GB	256 GB SSD
Processor	7th Gen Intel Core i5	7th Gen Intel Core i5
Operating System	Windows 10 64 bit	Mac OS Sierra

Table.6.2. Configurations of the systems

Amid our assessment we have looked at the heaviness of every application form on the gadget. As appeared in table 3 the rendition with our approach has an indistinguishable weight from the local version.

	Old Version	New Version
Windows	3.78 MB	2.38MB
MAC OS X	3.28 MB	3.35MB

Table.6.3. Weight of each application version

At last, in **Fig.** 6.2, we have thought about the memory utilization (RAM) of a portion of the principle segments of our application. For every module, we have ascertained the utilized RAM when their execution of the modules one by one. The used RAM is the expansion of the used RAM for the OS, by then for different applications in execution ultimately for our application. To get these measures from the stage subordinate form and the one actualized with our engineering, we have utilized the standard APIs gave in windows C++ .similarly, for the MAC adaptation, we have utilized the APIs gave by objective C++.

6.2. DISCUSSION

In the first place we will examine the feasibility of WD Drive Agent applications, particularly as far as reconciliation with UI and client encounter. At that point we will remark the lines of code spared and investigations the exhibitions.

6.2.1. Feasibility

We succeeded in implementing the complete application with the platform dependent APIs. It also was a success with our architecture to integrate in our application.

Thus, the developer can use any platform dependent API's with our architecture. Currently our application i.e. WD Drive Agent version 3.0 is totally functional and feasible with MAC and windows. As we disused earlier our architecture is very scalable and feasible if we want to extend our application for other operating systems like any Linux operating system or Android operating system etc we have to implement only the APIs of that platform according to the syntax of the functions required and return the values to the common code functions and we don't have to implement the common code for every platform. Using the same common code and implement it to all platforms makes easy maintenance and bug fixing. More the common code implementation more lines of code reduces as we reuse the same common code to each platform.

Our application is the backend faceless application which integrate with the front end application i.e. WD Discovery which we earlier discussed in Node JS and also cross platform.
Our application provides the front end application proper data which makes the front end application intelligent, high performable and extends its features.
By implementing our architecture to other backend desktop application developers can get the benefits which we discussed previously.
As we elaborately explain in class diagram in previous sections the main classes like DAManger,iPlatform,iPCCommunication and use of pure virtual functions helps to make the application more scalable and same can be implement in any backed desktop application.
With our approach, developers don't confront any confinement while actualizing a back end desktop application and the produced applications will be dependable and proficient quality.

Previously we required to install our backend application again and again in case of any update now we have place the executable file on repository so whenever any updates occurs WD Discovery automatically download the executable file and replace it with the previous one and WD Discovery will run the new executable in backend .

6.2.2. Lines of Code Saved

In **Table 2**, with our architecture, WD Drive Agent developers saved 40% of lines of code compared to platform dependent version. In table **Table.4** we showed the size of few of the components which are totally platform independent. The table contains the line of code numbers of few components which are totally cross platform however platform dependent codes also have some major parts in C++ which is cross platform and rest in platform dependent languages.
For example for SCSI protocol there are two files SCSI.cpp and another winSCSI.cpp. So for SCSI 25% to 35 % code is in C++ and rest in windows C++.

Cross platform components	**Line of Codes**
DA Manager	251
Localization	200

DiscoveryComms	650
Logger	280
pidLookUp	200

Table.6.4. Weight of few of the cross platform components

This versatility has a cost particularly in number of lines of code. When we stretch out our application to another stage we can accept that the developers will, once again save more than two thousand lines of code.

Around then, the quantity of lines composed for the normal code will be the same as the quantity of lines spared by developers. Our answer will be gainful for the back end application which needs to actualize for in excess of one target stage.

6.3. Performances Analysis

As represented in **Fig. 6.1**, for few of the main modules we mentioned, the application implemented with our architecture provides the same or better execution time as the previous platform dependent version as for MAC OS X application, but for windows application there is a huge improvement. We implement new architecture and better cross platform features.

For example, for Localization, with our solution, the application takes 2.8 ms while with the platform dependent code, the application takes 3ms. The execution time is almost same or better. In this manner, with our solution, we can assume that the created application will have a same performance but with widows as we change the architecture of our application its overall performance is huge but for MAC almost same performance.

The execution of functions through our cross-platform files do not take more time than platform dependent files. Also, for all the assessed undertakings, our approach gives preferred outcomes over the application than previous versions.

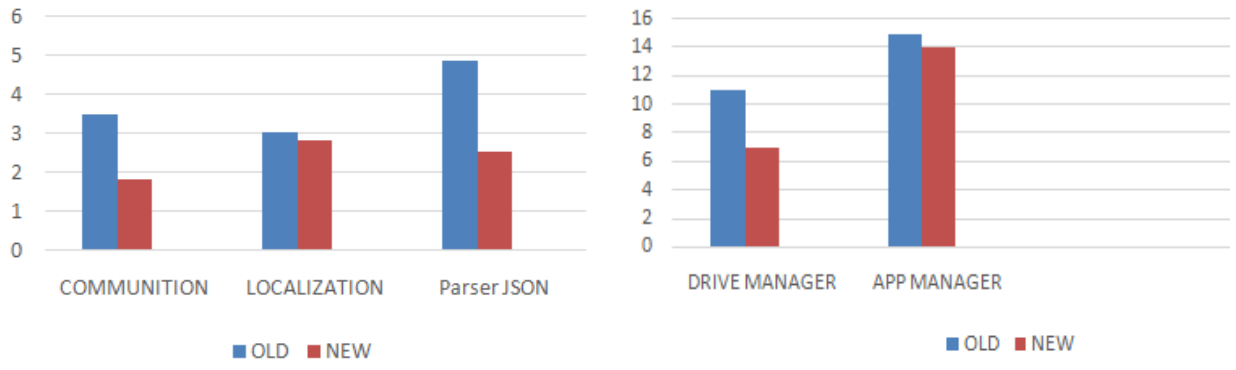

Fig. 6.1. Average execution times of some WD Drive Agent application's tasks

Fig.6.2. Used RAM for each WD Drive Agent application's version after various tasks execution

Fig.6.3 Represents the response time of the few major devices of Western Digital for windo**ws** version of the application. By implementing our new architecture for windows application now response time of the devices is much fast from the previous. **Fig 6.4.** Represents the average CPU usage of our application for MAC and windows. As previously the CPU usage is changes very frequently as our application used to poll for changes continuously. Now after implementation of folder watcher it do not poll continuously, it only invoke when any change ours.

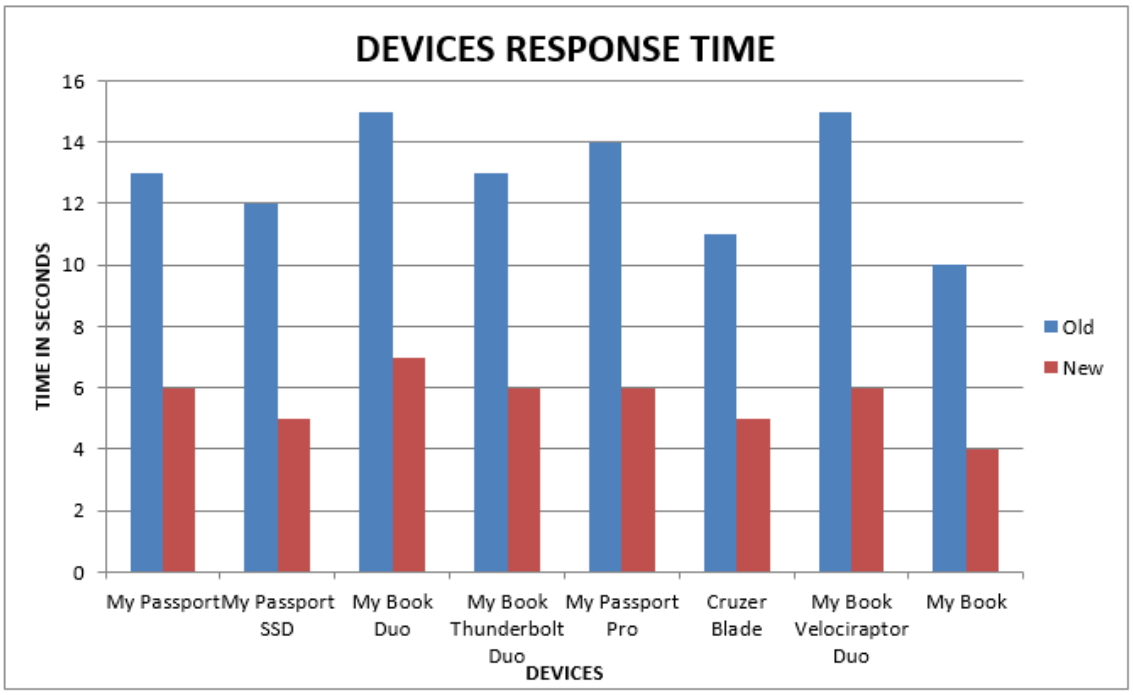

Fig.6.3. Used RAM for each WD Drive Agent application's version after various tasks execution

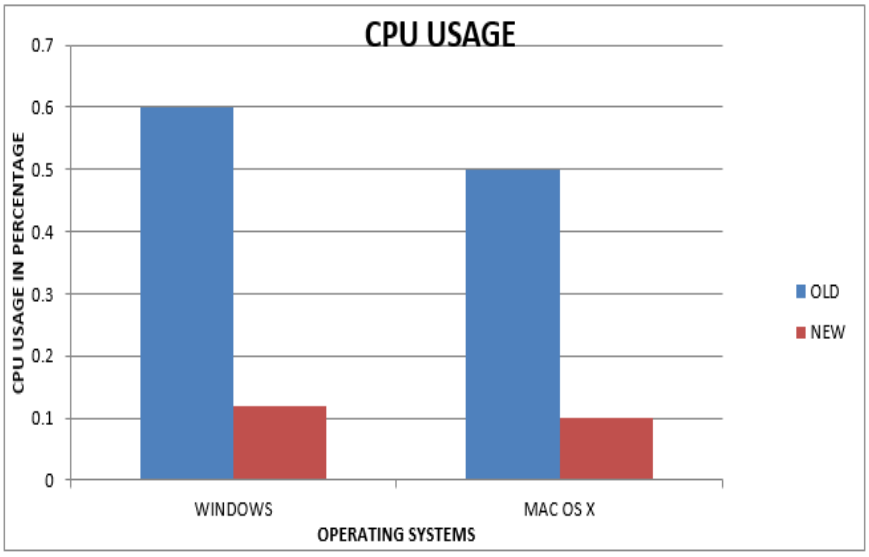

Fig. 6.4. Average CPU utilization

CHAPTER 7

7.1 CONCLUSION

Common framework components are very useful as we can implement its component to all the possible applications where we need to implement it but there will few difficulties which we already discussed in previous sections. If the developer implements our architecture as we implement in our application he have to write only one time the common code but he will not faces any of the difficulties faced by the common framework. Once developer writes the common code part of the application he can extend the application easily for any platform as C++ is common to all the platforms.

DA on Mac (OSX) has better acceptance, performance and almost zero install/update errors.

DA on Windows has a huge improvement scope (in terms of):

Installation, Update, Performance, Re-design (removing coupling, dependency of WD Drive Agent and WD Drive Service)

Benefits

- Common Device Agent
- Better engineering process, less maintenance overhead.
- Install time : 0 seconds (no .pkg or .MSI)
- Discovery package size reduced by ~10M on Mac, ~5M on Windows (no DA bundled)
- Same localization across Mac and Windows (.tmx on Win./ .lproj on Mac now it's JSON)
- Performance improvement:

Device Addition/Deletion would be faster than old Mac and Windows device agent.

1. CPU utilization improvement: Device agent won't poll every second of the newly installed applications.

2. Memory usage: Issues related to memory won't be anymore.

7.2 FUTURE SCOPE

We can easily extend this application to other platforms like Android or any of the Linux platforms like Ubuntu, Fedora etc and discover for more common cross platform libraries of C++ to extend the common code part of our application.

We can implement this architecture for Mobile application and build similar project for mobiles applications for various mobile application platforms like android, iOS and windows etc.

REFERENCES

[1] Martinez, M., & Lecomte, S. (2017, May). Towards the quality improvement of cross-platform mobile applications. In *Mobile Software Engineering and Systems (MOBILESoft), 2017 IEEE/ACM 4th International Conference on* (pp. 184-188). IEEE. [2] Joachim Perchat, Mikael Desertot, and Sylvain Lecomte.Common framework: A hybrid approach to integrate cross-platform components in mobile application. *Journal of Computer Science*, 10(11):2165, 2014.

[3] Dalmasso, I., Datta, S. K., Bonnet, C., & Nikaein, N. (2013, July). Survey, comparison and evaluation of cross platform mobile application development tools. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International* (pp. 323-328). IEEE. **[4]** Rita Francese, Michele Risi, Genoveffa Tortora, and Giuseppe Scanniello. Supporting the development of multi-platform mobile applications. In *2013 15th IEEE International Symposium on Web Systems Evolution(WSE)*, pages 87–90. IEEE, 2013.

[5] Heitkötter, H., Hanschke, S., & Majchrzak, T. A. (2012, April). Evaluating cross-platform development approaches for mobile applications. In *International Conference on Web Information Systems and Technologies* (pp. 120-138). Springer, Berlin, Heidelberg. **[6]** Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti.Comparison of cross-platform mobile development tools.In *Intelligence in Next Generation Networks (ICIN), 201216th International Conference on*, pages 179–186. IEEE,2012.

[7] Desruelle, H., Lyle, J., Isenberg, S., & Gielen, F. (2012, September). On the challenges of building a web-based ubiquitous application platform. In *Proceedings of the 2012 ACM conference on ubiquitous computing* (pp. 733-736). ACM.

[8] Balagtas-Fernandez, F., Tafelmayer, M., & Hussmann, H. (2010, February). Mobia Modeler: easing the creation process of mobile applications for non-technical users. In *Proceedings of the 15th international conference on Intelligent user interfaces* (pp. 269-272). ACM.

[9] L. F. Friedrich, J. Stankovic, M. Humphrey, M. Marley,and J. Haskins. A survey of configurable component-based operating systems for embedded applications. IEEE Micro, 21(3):54–68, 2001.

[10] Vellis, G., Kotsalis, D., Akoumianakis, D., & Vanderdonckt, J. (2012, October). Model-based engineering of multi-platform, synchronous and collaborative UIs-extending UsiXML for polymorphic user interface specification. In *Informatics (PCI), 2012 16th Panhellenic Conference on* (pp. 339-344). IEEE.

[11] Meskens, J., Luyten, K., & Coninx, K. (2010, May). Jelly: a multi-device design environment for managing consistency across devices. In *Proceedings of the International Conference on Advanced Visual Interfaces* (pp. 289-296). ACM. **[12]** Kramer, D., T. Clark and S. Oussena, 2011. Platform independent, higher-order, statically checkedmobile applications. Int. J. Design, Analysis Tools Circuits Syst.

[13] Allen, S., Graupera, V., & Lundrigan, L. (2010). *Pro smartphone cross-platform development: iPhone, blackberry, windows mobile and android development and distribution*. Apress.

[14] Mikkonen, T., & Taivalsaari, A. (2013). Cloud computing and its impact on mobile software development: Two roads diverged. *Journal of Systems and Software*, *86*(9), 2318-2320.

[15]Zhang, X., Kunjithapatham, A., Jeong, S., & Gibbs, S. (2011). Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, *16*(3), 270-284.

[16]March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., & Lee, B. S. (2011). µcloud: towards a new paradigm of rich mobile applications. *Procedia Computer Science*, *5*, 618-624.

[17] Martinez, Matias, and Sylvain Lecomte. "Towards the quality improvement of cross-platform mobile applications." *Mobile Software Engineering and Systems (MOBILESoft), 2017 IEEE/ACM 4th International Conference on*. IEEE, 2017.

[18]Boost C++ Libraries https://www.boost.org

[19] Windows Application Development Documentation
https://msdn.microsoft.com/en-us/dn308572.aspx

[20] MAC OS X Application Development Documentation
https://developer.apple.com/documentation/

APPENDIX A

VIDEO PRESENTATION LINK

YouTube Link:

https://www.youtube.com/channel/UCT0fWjRyZfY9Qebb_aBAh9w?view_as=subscriber

APPENDIX B

PLAGIARISM REPORT

Thesisfile_anurag

ORIGINALITY REPORT

13%	11%	11%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	thescipub.com Internet Source	7%
2	arxiv.org Internet Source	1%
3	Submitted to Mount Aloysius College Student Paper	1%
4	Joachim Perchat, Mikael Desertot, Sylvain Lecomte. "COMMON FRAMEWORK: A HYBRID APPROACH TO INTEGRATE CROSS-PLATFORM COMPONENTS IN MOBILE APPLICATION", Journal of Computer Science, 2014 Publication	<1%
5	www.visualstudio.com Internet Source	<1%
6	webcheerz.com Internet Source	<1%
7	docs.di.fc.ul.pt Internet Source	<1%

8	141.84.8.93 Internet Source	<1%
9	Submitted to Laureate Education Inc. Student Paper	<1%
10	Perchat, Joachim, Mikael Desertot, and Sylvain Lecomte. "Component based Framework to Create Mobile Cross-platform Applications", Procedia Computer Science, 2013. Publication	<1%
11	Submitted to Universiti Teknologi Petronas Student Paper	<1%
12	Ivor Horton. "Ivor Horton's Beginning ANSI C++: The Complete Language", Springer Nature, 2004 Publication	<1%
13	Matias Martinez, Sylvain Lecomte. "Towards the Quality Improvement of Cross-Platform Mobile Applications", 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2017 Publication	<1%
14	Hussin, Mohamad Fahmi, Ahmad Asari Sulaiman, Mohamad Huzaimy Jusoh, Mohd Zafran Abdul Aziz, and Affira Mastika Abdul Azid. "Safety and health inspection checklist for iOS application", 2014 IEEE Conference on	<1%

	Systems Process and Control (ICSPC 2014), 2014. Publication	
15	mangabb.co Internet Source	<1%
16	www.dell.com Internet Source	<1%
17	"Ubiquitous Computing and Ambient Intelligence. Sensing, Processing, and Using Environmental Information", Springer Nature, 2015 Publication	<1%
18	Beginning C++, 2014. Publication	<1%
19	www2.snowman.ne.jp Internet Source	<1%
20	"Internet of Things. IoT Infrastructures", Springer Nature, 2016 Publication	<1%
21	www.diva-portal.org Internet Source	<1%
22	Arunesh Goyal. "Moving from C to C++", Springer Nature, 2013 Publication	<1%
23	Michiel Willocx, Jan Vossaert, Vincent	

Naessens. "Comparing performance parameters of mobile app development strategies", Proceedings of the International Workshop on Mobile Software Engineering and Systems - MOBILESoft '16, 2016

Publication

<1%

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		