# Abstract

Java Applets have been around for almost twenty years now. They have been employed in displaying attractive features on the Internet. Java version 7 update 51 has enhanced the security clearance required to run any applet on client-side. This update was relayed to the users in light of many security loopholes being exploited by hackers around the globe.

Do Hoang Giang and Nguyen Vu Tuan's previous project outlined an implementation of a privacy preserving online social network that employed an embedded Java applet to communicate with a client-side standalone application. Given the nature of applet being used, it was decided to find an alternative. This project has created just the one.

The author has utilized a Windows level background process called Windows Communication Foundation service embedded in a Windows service for this purpose. There is no interaction between the users of the social network and the Windows service. This implementation is backwards compatible asserted by the fact that the functionalities of the social network are kept intact. Since the service is loaded only once onto the system, the process of communication is faster than the Java applet, which had to be loaded at every page.

# Acknowledgement

The author would like to thank his Final Year Project supervisor, Associate Professor Dr. Ng Wee Keong for his timely guidance and ever-gleeful approach to this project. His inspiration has been vital to the success of this project. The author would be failing in his duties if he did not express his exclusive gratitude to Do Hoang Giang. Do was the author's go-to person for clearing all kinds of doubts on this project. This project would have been an impossibility had it not been for him. Finally the author would like to mention Sahil Bajaj for being a great friend to bounce off ideas during the crunch times of this project and to provide varying insights into the functioning of the system.

As always, the author's family takes away the most gratitude for being understanding all along.

# Contents

# List of Figures



# List of Tables

# Chapter 1

# Introduction

### 1. Background

The current project undertaking is an extension to one "Privacy preserving Online Social Network" submitted by Do Hoang Giang AY 2012-13 and "Privacy preserving Facebook system" submitted by Nguyen Vu Tuan AY 2012-13. The previous project(s) deal with creating an online social network that gives users the control over their data without risking any third party or provider interference. The data being stored at the provider's server undergoes heavy encryption by employing functions from the Bouncy Castle library. A concise structure of this framework as a flow can be seen in Figure 1. The webserver carries the user interface and the general functionality of a social network. The client side standalone application deals with encrypting and decrypting data being sent or received from the website. These applications communicate via a Java Applet hosted by every page on the website and started as soon as a page loads.

For instance, when a page wants to submit a form, the JavaScript attached to it calls the applet. This applet then writes the DOM (Document Object Model) of the form to a file in a folder monitored by the Standalone Application. The application then reads the file, encrypts certain fields and writes the modified DOM to another file monitored by the applet. The applet reads the file, modifies the form's innerhtml by using this modified DOM. JavaScript submits the form then.

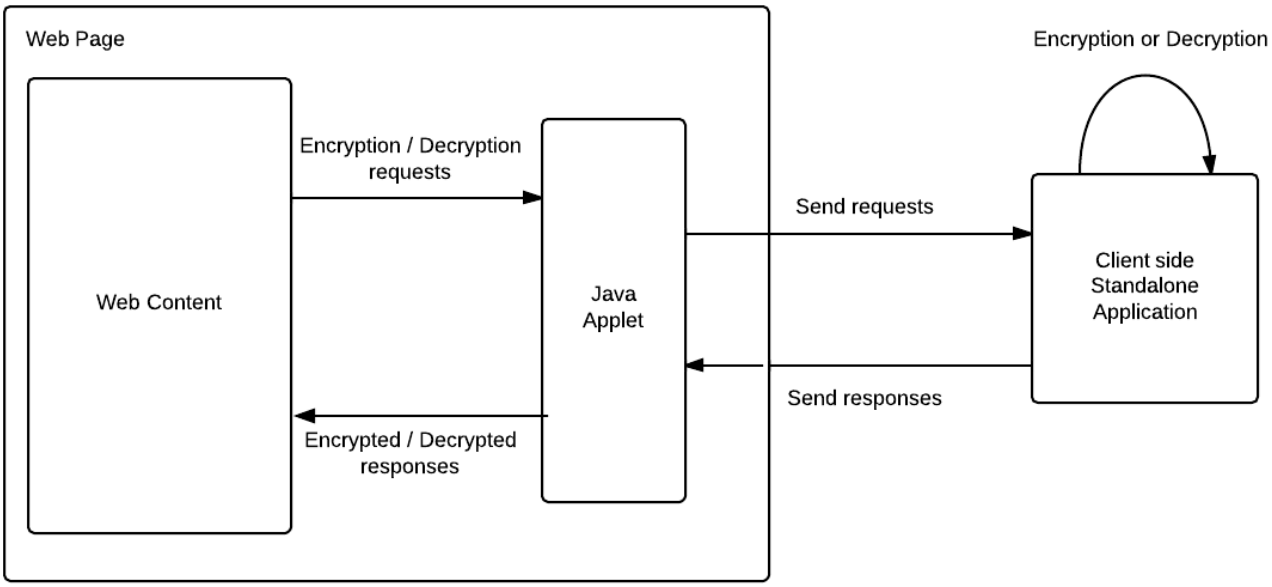


**Figure 1 Previous Implementation's Structure**

This encryption process uses seven different keys. Each user holds a secret master key and two public key pairs. These keys are generated at the local machine when the user registers with the social network. The master key is stored at client side whereas the two public key pairs are stored on the server side.

These key pairs are composed of an ElGamal encryption key pair and a DSA signature scheme key pair. The private keys of these two public cryptosystem are encrypted by a symmetric encryption such as AES scheme. The key for that symmetric encryption is derived from user's secret master password. The private keys are stored at the server in encrypted form, while public keys are stored in the database in plain form. This key management mechanism enables users to correctly exchange their public keys when a friendship connection is established. For example, user Alice wants to share her keys with Bob, she simply encrypts her keys using Bob's public ElGamal key and puts on the server. Beside that, since non-repudiation verification is performed using user's

public DSA key, any user can also verify that property of any signed message. For each session, a user owns a symmetric key for encryption when posting content. This symmetric key is also stored at the server in encrypted form using ElGamal encryption [1] A session is defined by the amount of content posted on the server.

## 2. Motivation

Over the years, the concept of a Java applet was solidified making it market leader in creating interactive web pages stimulating user responses. Java applets were in extensive use in all leading sites. But, this exposed these sites to malicious attacks that try to either siphon user information or alter settings on the web page. In the summer of 2013, Oracle released a long awaited update to Java, which prohibited users from running unsigned or malicious applications. This change came under Java version 7 update 51. Java has enhanced security model to make user system less vulnerable to the external exploits. The new version of Java does not allow users to run the applications that are not signed (Unsigned), Self signed (not signed by trusted authority) and the applications that are missing permission attributes. [2]

This meant that the current project model, involving running of a Java applet rendered the ideology of privacy futile. Apart from running the risk of being deprecated in the future, applets pose serious security loopholes. A skilled attack on the website can disable the applet and siphon the entire one-sided transaction happening. This can result in the social network to fail. Another way of looking at the model would be to analyze the effect of a Trojan horse attack. The malicious code if embedded onto the site, can route sensitive information over to the attacker. Since, the current model allowed the applet to directly read files on client side, there was possibly no stopping to the amount of hazards waiting to happen.

As an alternative, in this project the author proposed amendment to the applet design by using Windows operating system level processes or more commonly, background processes.

## 3. Objectives

The following are the three underlying objectives of this project:

- To create a folder watcher that can read as well write files on the client side.
- To implement a secure communication bridge between the webserver and the client side standalone application.
- To integrate the bridge with the current system of functions.

## 4. Challenges

There were primarily three major challenges to designing the new implementation of the project:

- Choice of background process
    - Every operating system has its pet background process structure implementation. For instance on Unix, they are called Daemons but Windows™ implements them as Services. The author had to take up one as a starting point to extend the project. The criterion used in deciding focused on the ease of implementation and integration with the current set up. It was essential that the previous work on the project must not be hampered in the due process of adapting the system.
- Choice of communication channel
    - System security settings in every major operating system look down upon background processes trying to

communicate directly over TCP/IP. Hence there rose a need to decide upon the choice of communication framework to be employed. The required framework would have a secure channel and must provide with tangible endpoints for the network to access the service functions.

- Choice of integration technique
    - The previous code was written with a mindset of involving a Java applet. The sending and receiving of data was accomplished with having the applet embedded on every page. With a background process, this changes, as the communication channel cannot be embedded on any page but acts as a mediating party. Thus, it was essential for the author to make the integration process smooth and less pertaining to any glitches.

## 5. Scope

Keeping in mind the complexity and delicacy of the change, the author decided to limit the expanse of the project to a single Operating System with backward compatibility. The author chose Windows as the starting point for the change. The Online Social Network chosen was eFacebook [5] Also, this system retains all the functionality of the previous one and enhances a few for the benefit of the process.

## 6. Project Schedule

Table 1 highlights the different phases and the scheduled tasks associated with them for the project.

**Table 1 Project Schedule**

| Phase | Time | Targets |
|---|---|---|
| Initiation And Learning | June 2013 to September 2013 | • Assimilate work done in the field of privacy preserving social networks by analyzing scientific papers<br>• Understand the cryptographic framework used in the previous project<br>• Finalize the new additions or changes needed for the project |
| Implementation | October 2013 to December 2013 | • Create Windows Services in order to monitor the client side<br>• Create a wrapping Windows Communication Foundation (WCF) Service in order to communicate between the client and the website<br>• Create communication links between the Windows Service and Windows Communication Foundation Service<br>• Integrate the setup with different pages on the actual project |
| Testing and Tweaking | January 2014 to February 2014 | • Add correspondence between the response received by the Service and the one sent to the server<br>• Automate Service initialization and installation<br>• Test the code on multiple pages |

## 7. Report Organization

The first chapter, viz. Introduction gives the readers a brief insight into the beginnings of this project. Review of Technologies Used follows next. It illustrates the technologies used by the author in implementing the system. The 3rd chapter, Methodology is the theoretical explanation of the system design employed by the author. Chapter 4 talks about the actual programming implementation of the system. The final chapter discusses the SWOT analysis and provides concluding remarks by the author.

# Chapter 2

# Review of Technologies Used

### 1. Microsoft Windows Services

#### 1.1 Definition

Microsoft Windows services, also known as NT services, help user to create long-running executable applications, running in the background as long as the Windows session is live. Windows services can be started, paused and stopped automatically whenever the PC boots up or while it is in operation. Since these services do not operate with a user interface, it makes them ideal to run on servers or mainframes. These services act as long running scripts executing commands without explicitly showing anything on the display. Windows services can also be run on a specific User account. They can be managed by employing Windows Service Manager. Since they run on the Operating System level, the services must have proper security clearances. A User level service has lesser clearances than a System level service.

#### 1.2 Basic Features

- All services must be installed on the System or User level before being started. Unlike normal computer programs, Windows services cannot be compiled and run.
- Windows Services run in a different security context from any other program.
- .NET framework does not allow the services to interact with the interactive sessions of the User as their station of operations are different.

- A service can exist in one of three basic states: running, paused or stopped. A service can exist in its running state indefinitely until it is either stopped or paused or until the computer shuts down.

### 1.3 Types

There are two broad categories of Windows Services. Services that are the only service in a process are assigned Win32OwnProcess category whereas the ones that share a process with one or more services are assigned Win32ShareProcess category. In terms of the kind of jobs done, Windows services can be of the following types,

- Adapter service
    - For a hardware device that needs its own driver.
- File System Driver service
    - File system driver or a kernel device driver.
- Interactive Process service
    - Rare breed that can only communicate with the desktop.
- Kernel Driver service
    - For low-level hardware devices such as a hard disk.
- Recognizer Driver service
    - Used during start up to determine if certain file systems exist on the system

### 1.4 Architecture

All Windows service applications inherit from System.ServiceProcess.ServiceBase class. They can utilize the following methods in order to fulfill the basic functionality of a background process,

- OnStart

    - Actions to be taken when the service starts. It can either be a call to another method or can contain the business logic of the application.
- OnPause
    - Actions to be taken when the service is paused either manually or programmatically.
- OnStop
    - Actions to be taken when the service is stopped either manually or programmatically.
- OnContinue
    - Actions to be taken when the service restarts from after being paused.
- OnShutdown
    - Actions to be taken when the PC is shutting down, if the service is running then.
- OnCustomCommand
    - Actions to be taken when the service returns a custom command.
- OnPowerEvent
    - Actions to be taken when a power management event is received, such as a low battery or suspended operation.

The ServiceBase class's Run method must also be called in order to initiate the process of the service.

In order to install the service and furthermore run it, the service class must also inherit System.ServiceProcess.ServiceProcessInstaller. By overriding its methods, this program adds executable classes that are used further. Services can be run directly from this class by calling appropriate post installation event handlers.

The service is installed onto the system by a .NET utility called InstallUtil.exe . This executable file uses the classes created above and

loads the service onto the system. If not left to be manual, the service starts after the installation as coded in the post installation event handers.

## 2. Windows Communication Foundation

### 2.1 Definition

Windows Communication Foundation or WCF is a framework for designing services that can communicate with other processes or services by sending asynchronous data among themselves. WCF services can either be hosted in a variety of processes or can sustain as self-hosted services. For most common uses, WCF services are hosted in a Windows service or under Internet Information Services (IIS). The behavioral aspects of the service are written in metadata, like XML.

### 2.2 Terminology

- message
    - self contained unit of data with a header and a body
- endpoint
    - construct that assists in sending or receiving messages. It comprises a location (an address) that defines where messages can be sent, a specification of the communication mechanism (a binding) that described how messages should be sent, and a definition for a set of messages that can be sent or received (or both) at that location (a service contract) that describes what message can be sent. [3] A WCF service is exposed via these endpoints. Each endpoint can have a function associated on its end.

- address
    - the location where messages are received and is specified as Uniform Resource Identifier or URI. This can be integrated with HTTP or TCP/IP protocols. A WCF service must have a unique endpoint address in order for it to function calmly.
- binding
    - definition of how the endpoint communicates with the world. It is constructed by a set of components called binding elements that "stack" on each other in order to produce a communication infrastructure. A binding defines the transport protocol; security protocol and the encoding used being used in the data communication. There are four major System-Provided bindings viz.
        - *BasicHttpBinding*
            - HTTP protocol binding used majorly in letting ASMX-based web services communicate with the WCF service.
        - *WSHttpBinding:*
            - interoperable binding used in letting web services of all types communicate with the WCF service.
        - *NetNamedPipeBinding:*
            - connects with WCF endpoints on the same machine.
        - *NetMsmqBinding:*
            - connects with WCF endpoints using queued message connections.

- binding element
    - component of the communication stack in the binding. Describes the communication protocol, encoding and the security context of the address.
- behaviors
    - component that controls the various run-time aspects of the service. Behaviors are grouped according to the scope. Common behaviours affect all components whereas specific ones do not.
- service operation
    - procedure that defines functionality of the service. It can have multiple arguments and can return data of any type.
- service contract
    - ties together operations into one bed. It defines the namespace and other service level settings. It is a required component of the class.
- operation contract
    - defines the parameters and return type of each service operation in the service using System.ServiceModel.Web namespaces like WebInvoke. Return type, method to do so, body style of the returned value and the URI extension are mentioned.
- hosting
    - WCF service must be hosted in some process or can be self hosted. The host controls the lifetime of the service. Services can be self-hosted or managed by Windows Services, IIS etc.

- client application
    - program that exchanges messages with the WCF service. This can either be a web page or another service for that matter.

- security
    - In WCF, includes confidentiality (encryption of messages to prevent eavesdropping), integrity (the means for detection of tampering with the message), authentication (the means for validation of servers and clients), and authorization (the control of access to resources). These functions are provided by either leveraging existing security mechanisms, such as TLS over HTTP (also known as HTTPS), or by implementing one or more of the various security specifications. [3]
- transport security mode
    - confidentiality, integrity, and authentication are provided by the transport layer protocols. Despite being robust, "man-in-middle" attacks can be a worry.

### 2.3 Architecture

An application interacts with the service by sending or receiving data through endpoints provided by the WCF service. The WCF service then initiates its response by looking up at the Contract Layer. This layer has contracts for the Service, Operations and Messages as detailed in the previous sub-section. It describes the behaviors of the service in terms of protocols to be used and the encoding to be followed. The Service run-time Layer describes the actions that are to be taken when this service is operational. It acts as a watchdog and provides contingency measures in case of any failure. The Message Layer is composed of channel stacks. There exist two types of stacks, viz., transport stack and protocol stack. The former stack is involved in reading and writing messages received/sent and to encode them respectively. The protocol stack on the other hand acts as the implementer of the message processing protocols. It reads or writes additional headers to the messages. The Hosting Layer describes the way WCF service is hosted on the system. It has nothing to

do with the communication and thus acts as a cup to hold the WCF service. Figure 2 illustrates the stack.

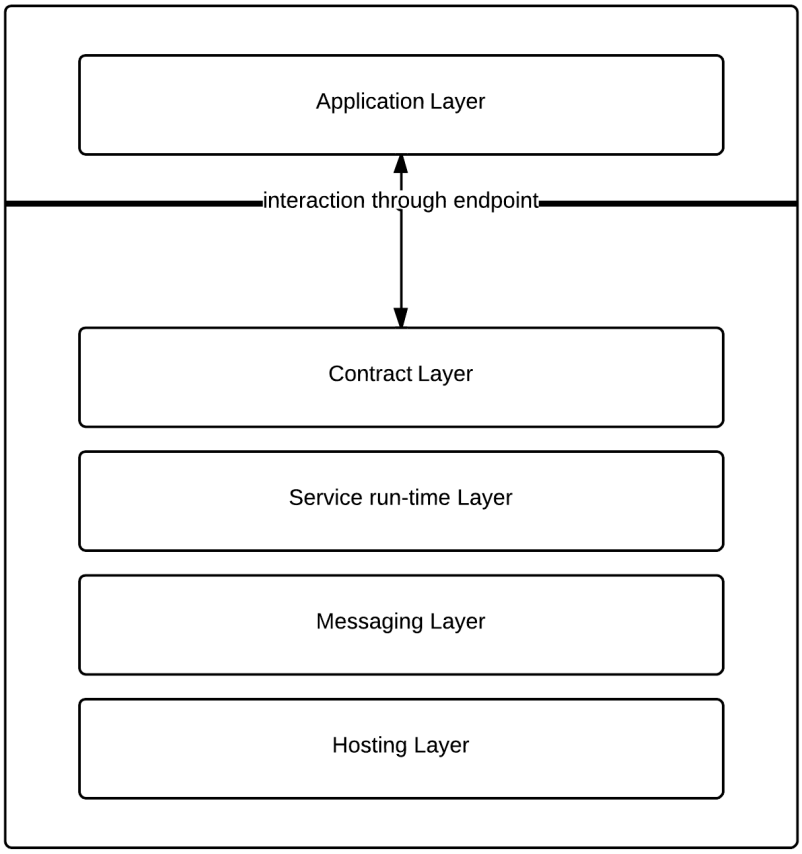


**Figure 2 Architecture of a WCF service [4]**

## 2.4 Integration with web pages

WCF services' endpoints can be accessed via AJAX requests from any web application. Having initially only set to be compatible with SOAP (Simple Object Access Protocol) WCF is now in hooks with REST (Representational State Transfer) architectures.

For instance, as shown in Figure 3, the WCF service exposes one of its operations via an endpoint. Now once the service is installed and is running, AJAX calls from a web page can send and receive data from the

WCF service. The service takes in the data and sends appropriate response. The response is converted to JSON and used by the web page.

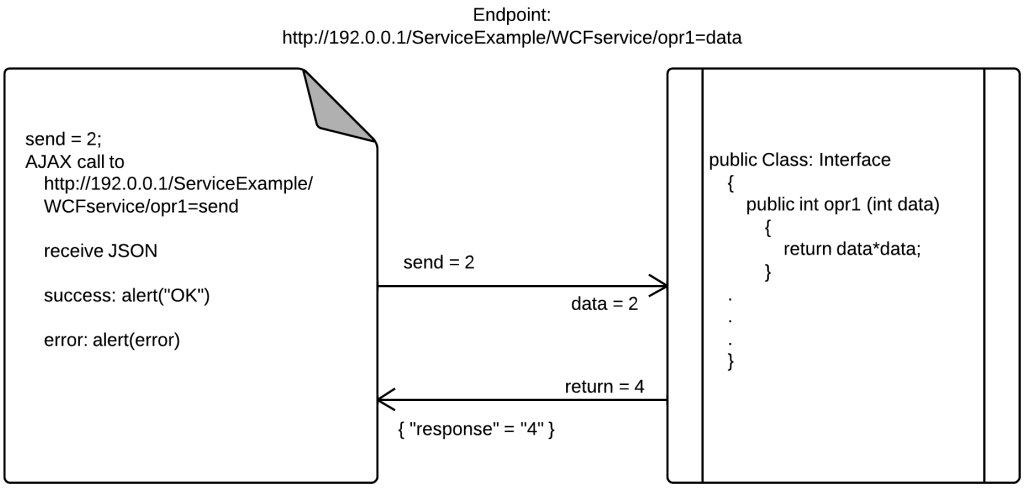


**Figure 3 An AJAX call to WCF service**

# Chapter 3

# Methodology

### 1. Background System Design

### 1.1 General Structure

As stated in Chapter 1, the main aim of this project was to design and implement a new and stable release of a communication channel between a privacy preserving social network and a client-side standalone application. A Windows service with Windows Communication Foundation service was chosen as the fabled communication bridge. The author opted for this strategy keeping in mind the delicate nature of the project and the limitations of the same as outlined in Chapter 1 section 5.

The system included a WCF service hosted in a Windows service. The Windows service would also act out the business logic of the operation. This Windows service was of Win32OwnProcess type. It thus was part of a single process and did not share its process with any other Windows service. It hosted the WCF service, which was accomplishing all the communication tasks.

As outlined in Figure 4, a webpage wanting to access the client-side standalone application would make an AJAX call to the endpoint of the WCF service. Once, the data was sent through the endpoint, the WCF service would write to a file and name it accordingly in a particular folder viz. “efb/in” on the client. The client-side standalone application monitoring that particular folder would then read the file’s contents and do the necessary encrypting or decrypting. Once the said process is

finished, the client-side standalone application would then write the new contents into another folder viz. “efb/out” on the client. This time, the

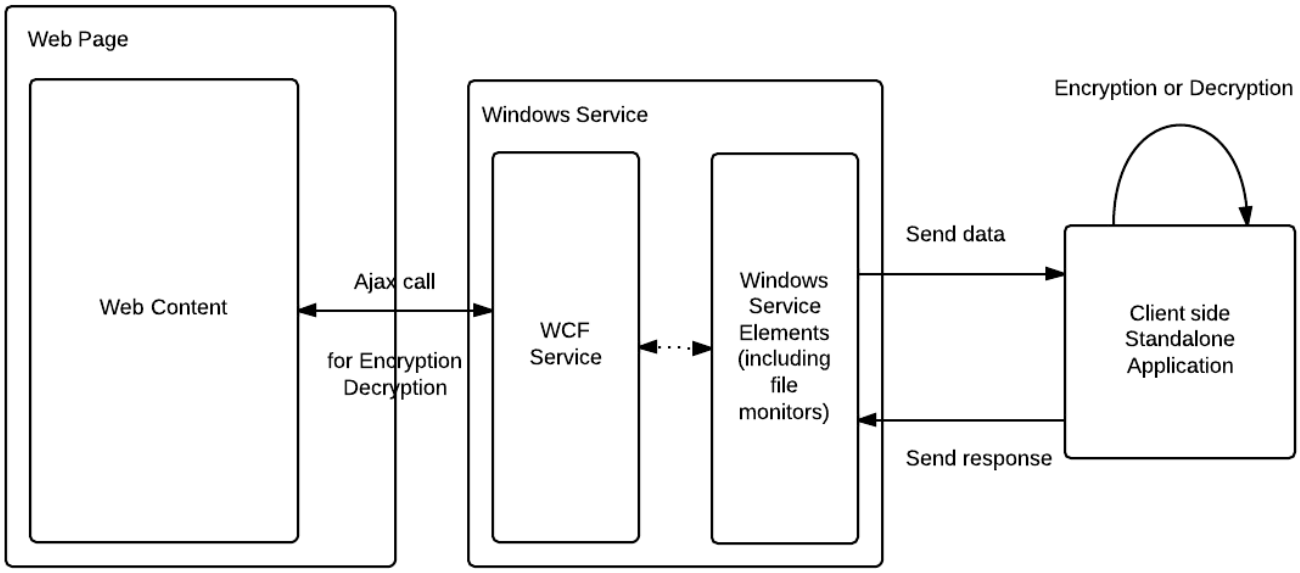


**Figure 4 Current Implementation’s Structure**

Windows service is monitoring the said folder. It reads the file, copies the contents into a local variable and passes it on to the WCF service, which then sends the data back to the webpage.

### 1.2 File Monitoring Design

While designing an apt file monitoring system in Windows, the author had to take certain considerations into his working. The three major initiations to be taken care of were as follows,

- Choice of design
    - There are two types of file monitoring systems, polling and event driven. Polling deals with continuous pings to the source in view of an apt reply whereas event driven, as the name suggests is designed to act only if a particular trigger event occurs. The author chose to utilize the event driven technique in order to monitor files in the folder. Polling is

memory expensive and can act as a deterrent to system performance. Also, the project demanded actions to be taken only when the webpages demanded so, therefore the service was supposed to listen for events rather than be an active entity asking for triggers.

- Choice of event trigger
    - Once the design dilemma was settled, the author had to choose an apt event trigger for the file monitoring system. A wrong choice of trigger could get the system out of place and have it render incorrect files to the system. The main battle was to choose between "file changed" and "file created" events. Despite the later event looking more likely, it would have proven to be the wrong choice. This is so because a new file created by the application would never be empty. So technically, a new file is created, closed and then opened in order to write the data down. The trigger would fire immediately when the file is created and the file will be opened even before it is written onto. This would result in no data being transferred and file system exceptions occurring all the time. Also, the "file created" trigger would not work for the case where a file is overwritten by the same command, which is the usual case. On the other hand the "file changed" trigger is supposed to work only when any file in a particular folder recently created or not is modified. By this logic, if a new file is created, the file monitor waits for the application to write the data down and close the file before firing up.
- Choice of file monitor
    - The system design happens to have a WCF service hosted in a Windows service in such a way that the later service runs for an indefinite period while the WCF service is triggered

only during the AJAX call from the webpage. Thus, in order to monitor a client side folder and read from the same, the Windows service came out as the stronger of the two choices. Also, without being directly accessible to the outside world the Windows service provided a logical layer of security to the design.

### 1.3 Data relay

Using the operation contract feature of Windows Communication Foundation, as explained in sub-section 2.2 of Chapter 2, the service can relay the data back to the webpage in either XML or JSON format. In the project though, the data is sent to the webpage by default means of XML whereas it is received in JSON. The data in general contains innerHTML text of a form or content with the name of action to be taken.

When the data is received by the WCF service, it is split into two parts, viz. action and content. The action part forms the name of the file the service needs to open/create and the content is the data that needs to be written onto the same.

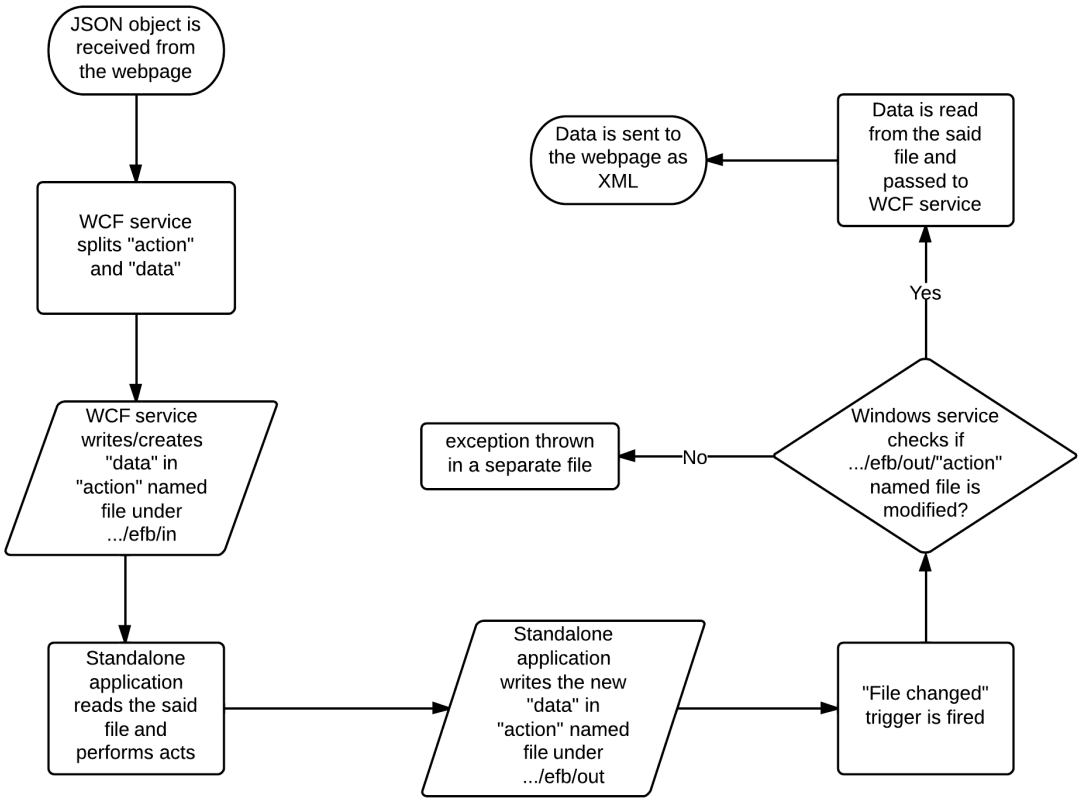


**Figure 5 Data Relay Flow**

When monitoring the folder, the Windows service looks for changes in the file named after the action previously received in the system. If the said file has been modified, the content is sent back to the webpage through the WCF service. A precise flow of the system is shown in Figure 5.

## 2. Webpage Design

### 2.1 AJAX call design

As mentioned previously, the author used AJAX calls to make the webpage communicate with the WCF service, which in turn communicated with the client-side standalone application. Initially, the data to be sent is converted into a JSON object with the action to be taken and the actual HTML content to be sent.

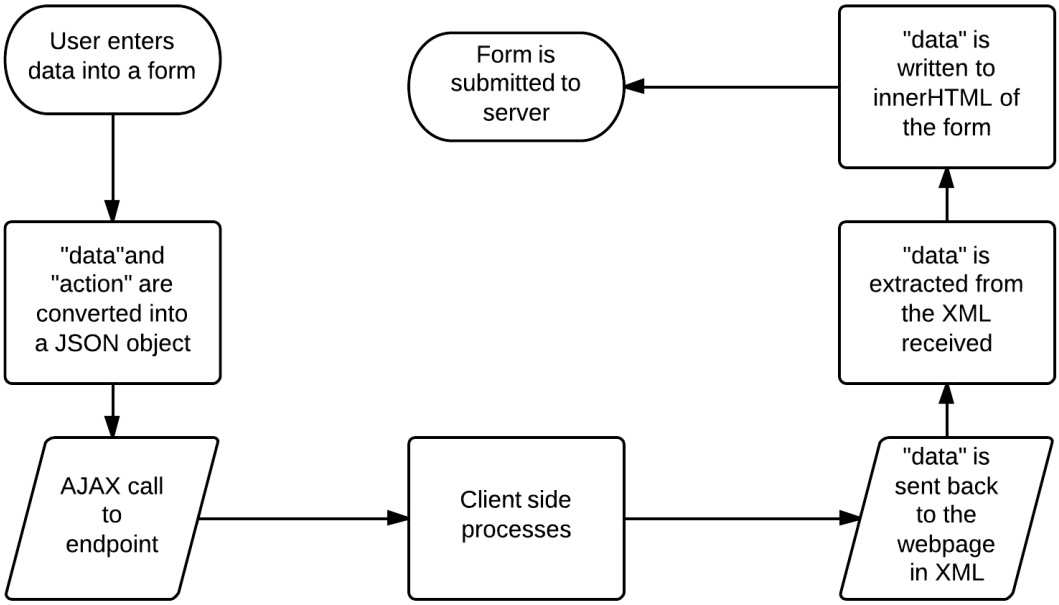


**Figure 6 AJAX call Flow**

Once that is done, the data is relayed via the call to the endpoint. After the data is received, the encrypted/decrypted data is extracted. By using the innerHTML property of forms, the new data is added to the pages and submitted to the server. Figure 6 displays this flow.

## 2.2 Comparison between designs

Table 2 describes the differences between the previous approach of designing the system, as explained in Chapter 1 and the current one.

**Table 2 Comparisons between designs**

| Previous Design | Current Design |
|---|---|
| The Java Applet had to load on every page before being of any use in acting as the communication bridge between the client-side standalone application and the website. | The service application does not "load", as the functionality of the same is exposed through AJAX calls. |
| Data was being sent over the protocol in raw form. | Data is exchanged in XML/JSON formats. |
| The system is valid until Java Update 7 Version 51 | This system is valid throughout on any Windows abled PC |
| The Java applet is a security lose end, as it is very easy to break up the play and infect the applet file present in the client system. | The service system is beyond visibility from a regular attack and cannot be edited easily once converted to an executable format. |

# Chapter 4

# Implementation

1. **Service implementation**

In order to capacitate the system described in the previous chapter, the author implemented a multi class model (.NET 4) explained in this section. The system is defined under the namespace of "eFacebook.Mediator.Source". The WCF service is declared under the class FocalService, which implements Mediator while the service hosting the same is named FocalWindowsService. The ProjectInstaller class helps in creating installation files for the Windows service. The class CommunicationManager handles the intra-service communication pathway.

- Mediator

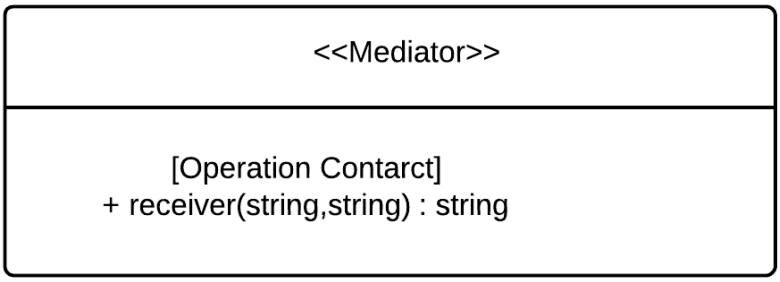


**Figure 7 Mediator Class**

  - The interface that outlines the method "receiver". This method is designed to receive the AJAX calls. The method receiver takes in two string arguments named "stuff" and "action". The former variable takes in the HTML data being transferred to the service while the later is involved in noting down the action that is supposed to be taken by the service. It is to be noted that the "action" converts into the filename, which is being filled with "stuff" under a particular folder.

- The Operation Contract as mentioned in the sub-section 2.2 of Chapter 2 is also utilized. Here the contract mentions the kind of method this function behaves as. In this case, the author has used "POST" as it is better than using "GET" when sending or receiving sensitive data. The method is logically an invoke operation and can be called by any WCF REST programming model. The response as well as received body are chosen to be "wrapped". This means that when the service or client serializes parameters and returns values to a message, it writes them within infrastructure-provided XML elements and is wrapped. Furthermore, the contract also mentions the URI template of the method. This is the suffix added to the endpoint for ease of differentiation from other methods. In this case, the URI template is "/form".

- ConnectionManager

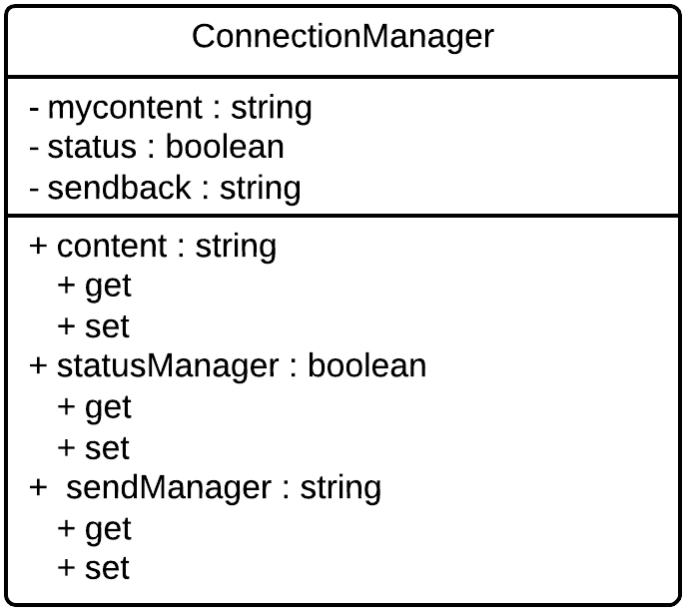


**Figure 8 ConnectionManager Class**

- This class acts as the messenger between the FocalService and the FocalWindowsService classes. The variable "content" is used for storing the value of "action" received by the method "receiver". The string variable "sendback" is

used to read in the data from the file created by the client-side standalone application and pass on to the WCF service. The Boolean variable, "status" is used for storing the readability of the variable, "sendback". Only when set to true, can the data read be sent to the webpage wanting the same.

- FocalService

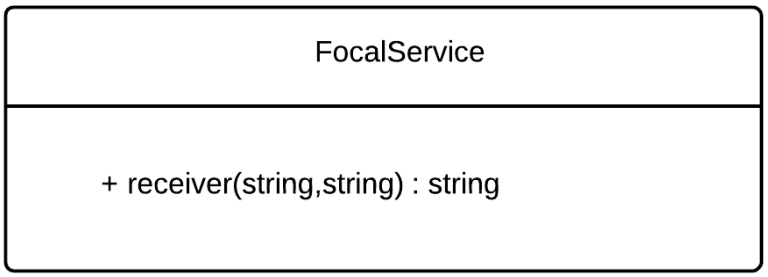


**Figure 9 FocalService Class**

  - This class implements Mediator and is the pioneer WCF service class. The method "receiver" is the only function in the class. It sets the "path" variable ("…/efb/in") for the service and concatenates the "action" variable received from the call as the filename. It also sets the value of the variable action to a "content" from ConnectionManager. It then utilizes the built in functionality of System.ServiceProcess.ServiceController's property viz. ExecuteCommand to access FocalWindowsService's OnCustomCommand method.
  - The method receiver also accomplishes writing variable "stuff's" contents to the file specified by "path". The method keeps polling for the data to be sent back i.e. "sendback" using the Boolean variable of "status". When the later becomes true, it returns the value as string in XML format.

- FocalWindowsService
    - This class implements System.ServiceProcess.ServiceBase class in order to act as the pioneer Windows service implementation. It contains a public "serviceHost" variable of type System.ServiceModel.ServiceHost. This type provides the hosting capability to any service and is needed in this case to host the FocalService class.

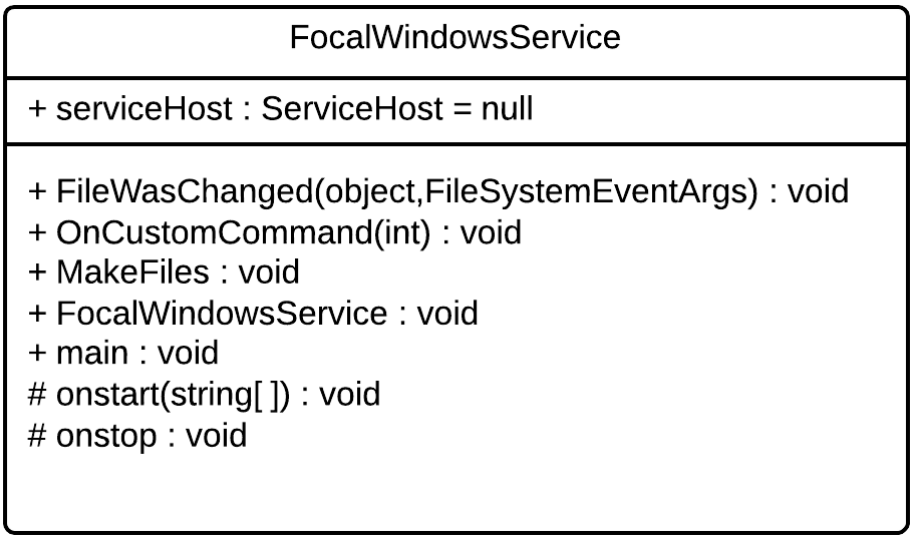


**Figure 10 FocalWindowsService Class**

    - Its protected method, "OnStart" is used for detailing out the tasks to be implemented when the service is started. First things first, this method initiates the FocalService by calling "ServiceHost.Open" method. It then creates a System.IO.FileSystemWatcher variable. This watcher listens to the folder "…/efb/out" for any "FileChange" events to happen. As mentioned in the sub-section 1.2 of Chapter 3, the author clearly chose event based monitoring technique over polling one. If the event does happen, it enlists the "FileWasChanged" method to act as the executer.

- The “FileWasChanged” method takes up event and waits for 20 milliseconds before checking if the event was raised by the file whose name is same as the variable “content” from the class ConnectionManager. If so, it reads the contents of the file into ConnectionManager’s “sendback” and toggles the “status” variable to true. It also checks for errors in any file input/output process.
- The “OnCustomCommand” menthod is supposed to take in requests from ServiceController’s ExecuteCommand. It relays the commands received to their proper handler functions defined in this class.
- The “makeFiles” method is added in as a debugger tool in order to help the future developers. It adds the current file name being written to by the FocalService and the current time, as a file to another folder called “C:/test/watched”. Its main purpose is to catch the filename and the timestamp of the same.
- The protected method, “OnStop” is used to stop the execution of the FocalService after the FocalWindowsService has been stopped.
- The “main” method of the class runs the FocalWindowsService class by calling ServiceBase’s method “Run”.
- The constructor of the said class is used to set the visible name of the Windows service, as displayed in the Windows Service Manager.

- ProjectInstaller

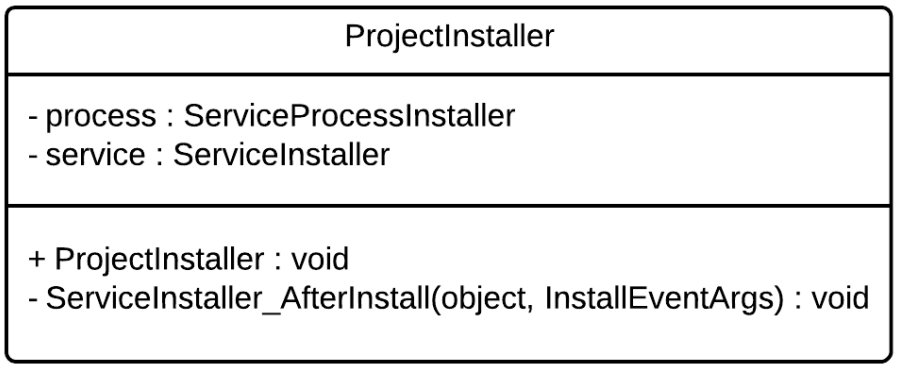


**Figure 11 ProjectInstaller Class**

  - This class implements System.Configuration.Install.Intaller. Its constructor adds in the files required for the InstallUtil.exe to help the system install the Windows service. It also adds in an event handler for after install scenario.
  - The said event handler is implemented by the "ServiceInstaller_AfterInstall" method. It starts the Windows service once it is installed onto the system.

## 2. Concept flow

The Windows service has been named eFacebookMediatorService. In order to install and uninstall the Windows service using .NET InstallUtil.exe, the author wrote scripts that can be run once the service is downloaded onto the machine in the future. This forms the Set Up Phase. Figure 12 shows the major flow of this operation and more.

Once the service is installed using the script talked about above, the ProjectInstaller class's method "ServiceInstaller_AfterInstall" starts the same. After that the "main" method of the FocalWindowsService class is executed, running the service completely. The "OnStart" method then runs the FocalService which is now ready to be called through AJAX by

any webpage. It also sets the file system watcher and file change event handlers up.

When an AJAX call to the endpoint with the correct URI template as mentioned in the previous sub-section, is made, the "receiver" function gets called. A file of the name "action" and the timestamp is created for debugging purposes. No content is copied there. The method then writes the content to the named file in "…/efb/in" and waits for the data to be sent back. This file is then taken up by the client-side standalone application which reads it and writes modified data into another file of the same name but in "…/efb/out" folder. This folder is being watched by the FocalWindowsService class. It triggers a file- changed event and the handler method (FileWasChanged) is called. This method checks if the said file is actually the one required by comparing its name to the action received previously. If so, the "sendback" variable is set and the "status" flag is set to true. This immediately triggers the "receiver" method's polling operation and the contents of "sendback" variable are sent back to the calling webpage. All of this happens in a time range of 30 to 70 milliseconds.

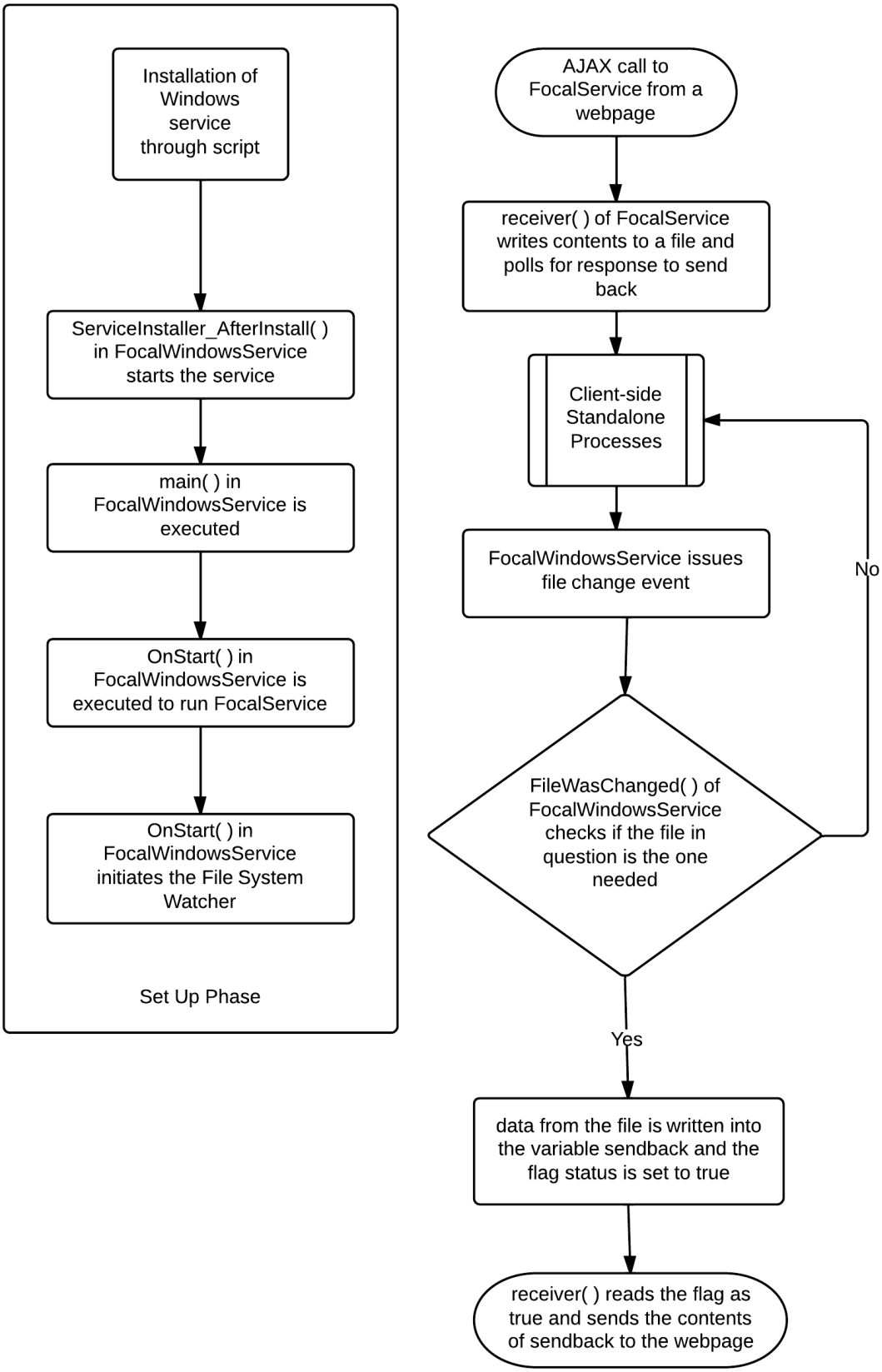


**Figure 12 Complete Run-through of the implementation**

**3. Peripheral implementations**

- Metadata for Windows service
    - As mentioned in the sub-section 2.2 of Chapter 2, for every Windows service, an endpoint is a complete necessity. The author created a file named App.config which describes the binding used, endpoint behaviors and exception handling triggers.
    - The endpoint chosen is http://localhost:8000/ServiceModelSamples/service with the URI template having "/form". This means that all AJAX calls are being made to http://localhost:8000/ServiceModelSamples/service/form
    - The webHttpBinding is used to define the behavior of the system in terms of protocols, encoding and transfer channels. Refer to sub-section 2.2 of Chapter 2 for more insights into this concept.
- AJAX implementation
    - An illustration of a common AJAX call is shown in Figure 13. First, the data to be sent is converted into a JSON object with stuff and action as shown in lines 4 and 5.
    - The JSON object is then sent as a string and the data expected back is marked as XML.
    - The URL mentioned is nothing but the endpoint of the WCF service/FocalService class.

```
1   var action = "decryptStatus";
2   var tosend = $("#encryptedStatuses").html();
3   var jData = {};
4   jData.stuff = tosend;
5   jData.action = action;
6      $.ajax
7       ({
8             cache: false,
9             type: "POST",
10            async: false,
11            url:  "http://localhost:8000/ServiceModelSamples/service/form",
12            data: JSON.stringify(jData),
13            contentType: "application/json; charset=utf-8",
14            dataType: "xml",
15
16            success:function(data)
17              {
18                test = $(data).find ('receiverResult');
19                var myhtml = test.text();
20                $("#status_display").html(myhtml);
21              },
22
23            error: function (xhr,error)
24              {
25                 alert(error);
26              }
27      });
```

**Figure 13 An implementation of AJAX call**

- If successful, the necessary data is extracted from the XML received and is utilized in the innerHTML of another content. Lines 16 to 21 illustrate this.
- If unsuccessful, the error caught is shown. Lines 23 to 26 show this.

# Chapter 5

# Conclusion

## 1. **Discussion**

During the course of this project, the author had the opportunity to deliver on his expertise in programming as well as pick up previously unknown paradigms. A Windows Communication Foundation service hosted in a Windows service was implemented in order to act as a communication channel between a privacy preserving social network and a client-side standalone application. This system replaces the previous installation of a Java Applet that acted as the communication bridge.

The current design utilizes the concept of endpoints in a WCF service. The webpage communicates via AJAX calls to these endpoints. The WXF service picks up the data and writes it to a file. This file is read into the client-side standalone application, which in turn writes the modified contents into a second file in another folder. This folder is watched for changes by the Windows service, the one hosting WCF service. If the file changed, is the one desired, the contents are read and sent to the WCF service. The WCF service packages it as XML and sends it back to the webpage that had initiated the call.

All of this process has the worse case time of 70 milliseconds. It is secure enough as no outside process can damage the client. Even if malicious script is sent through the AJAX call, the client will not recognize it as any action and hence not run it. Hence, all objectives have been met.

## 2. SWOT analysis

The current implementation has the following features being categorized into different parts,

- Strengths
    - Unlike the previous implementation, there is no load time for the Windows service to run. The service needs to be started only once. Every page loads without having to wait for the Windows service to run before.
    - WCF makes sure that third party creations such as this are always allowed in the system and will never go out of support.
    - All the previous Online Social Network functionalities have been kept alive in this implementation. This tool is totally backward compatible.
    - The worst-case time from the AJAX call received to the response sent back is negligible at 70 milliseconds.
    - No user interaction is required. Unlike the previous implementation, the user does not handle pop-ups for Java approval at all.
    - The file handling system is cleaner as the file watcher system included in the Windows service now listens for events instead of polling for changes.
    - The Windows service and WCF service interaction is bare minimum keeping in mind the security point of view.
    - On the webpage, AJAX calls have replaced setting content and action of Java Applets. This has made the code run smoother.
- Weaknesses
    - It is cumbersome to debug if the system is not performing up to the expectations.
    - If the future implementations on the eFacebook system involve loads of data being transferred in one AJAX call, it could lead to the file system coming under lot of scrutiny and the system may have to change to combat that.

- Opportunities
    - Currently the system works on any machine that is running Windows XP and above. This system can surely be extended to multiple Operating Systems such Unix (Daemons).
    - The whole data exchange through file system can be changed to a more secure and less open option.
- Threats
    - Dual Use of Concern has always been a major player in any new technology. If one can create such a complicated system, rest be assured that certain trapdoors can creep up in the system. We have to be wary of unwanted use of our technology.

**3. Future Works**

The following tasks can be completed in the future versions of this project,

- This system can be extended to other Operating Systems too. The idea can remain the same. Only differences that can happen are related to the language used and the compatibility of the same with the server-side technology.
- The file system can be replaced by a slicker option that does not expose data to the user at all. Probably have a system that directly passes the data to client-side standalone application. This can further decrease the time taken to communicate.
- Lastly, this implementation can be extended to cover eTwitter application created by Do Hoang Giang as detailed in his Final Year Project thesis, Privacy Preserving Online Social Network.

# Chapter 6

## References

[1] Do Hoang Giang, *Privacy Preserving Online Social Network*, page 24, 2013

[2] http://www.java.com/en/download/help/java_blocked.xml

[3] http://msdn.microsoft.com/en-us/library/ms731079(v=vs.110).aspx

[4] http://msdn.microsoft.com/en-us/library/ms733128(v=vs.110).aspx

[5] Nguyen Vu Tuan, *Privacy Preserving Facebook System*, 2013