Chapter 3

Methodology

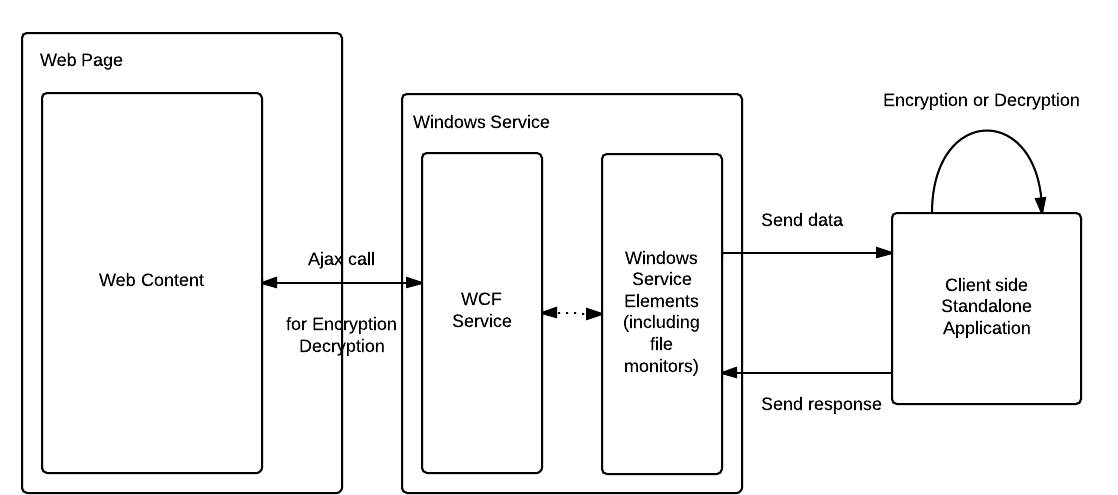
1. Background System Design

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 sub-section X.

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 X, 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 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

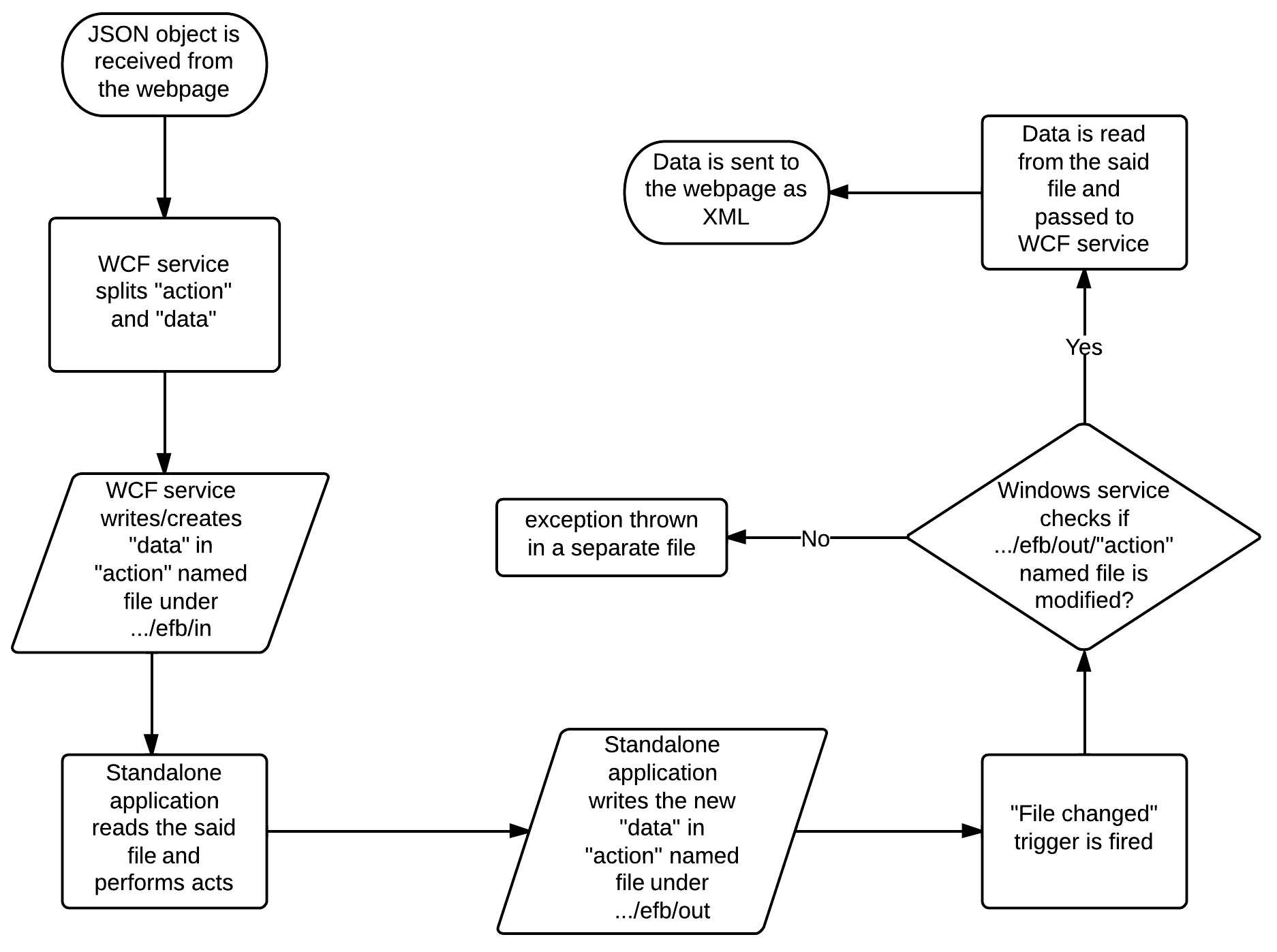
While designing an apt file monitoring system in Windows, the author had to take certain considerations into his working. The three major initiations to be taken care of were as follows,

* 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 an also 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 section X, 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.

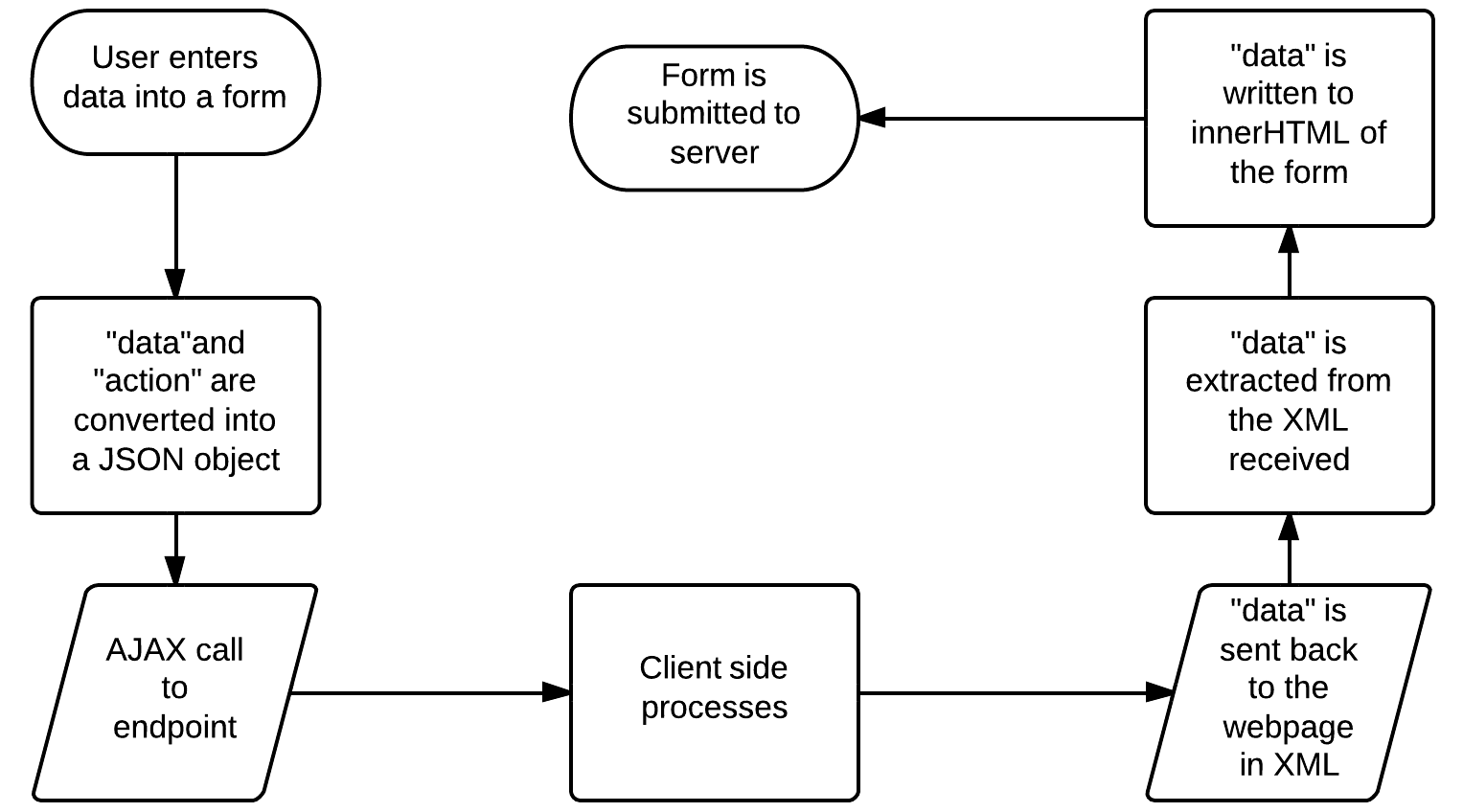
When the data is received by the WCF service, it is split into two parts, viz. action and content. The action part forms the name of the file the service needs to open/create and the content is the data that needs to be written onto the same.



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 X.

2. Webpage 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.



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 X displays this flow.

2.2 Comparison between designs

Table X describes the differences between the previous approach of designing the system, as explained in Section X and the current one.

|  |  |
| --- | --- |
| 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. |