SOFE 4790 Final Project Report: Resilient Weather Service

Group Members:

Devan Shah	100428864
Miguel Arindaeng	100394094

Submitted to:

Ying Zhu

&

Weina Ma

Due Date:

December 3rd, 2014

Table of Contents

Abstract

This report is intended to give the reader a detailed overview of the distributed system that this group is designing: a resilient weather service. Useful weather data is not always readily apparent to the user, and this project hopes to remedy this. The weather service will reflect on aspects that make systems resilient, such as what to do when there are unexpected server terminations, resource unavailability, or network/server failures. In addition to this, this report will go over the overall design of the system, such as listing the stakeholders and requirements. Design diagrams will also be included to give a sense of the overall structure of the resilient weather system. There will also be screenshots of the UI and the backend server system, as well as a short video demonstrating the capabilities of the system.

This report will also go over implementation details of the weather service. It will use existing XML and HTTP web services to retrieve data and construct feeds in order to quickly retrieve the current weather. The client will also register with the weather service server whenever there is a new client. The server will continuously check the registered clients to make sure they are getting the information that they requested. The server side will be able to handle unexpected process terminations, resource unavailability, and network/server failures through a variety of implementations. This will achieve the goal of creating a resilient weather service.

Project Introduction

There is a need for a resilient weather system that provides quick and non-complex updates of the weather for locations in Canada. This system would be intended for users that need reliable weather information instantly, as opposed to other weather services that overwhelm the user with a plethora of weather information. Examples of these users are: Canadian military personnel, meteorologists, boating personnel, and marine scientists. These users would not need a lot of information but quick, at-a-glance weather information that is reliable no matter the environment.

A resilient weather system of such nature can be implemented via a distributed system. This weather service distributed system would obtain weather information from the Canadian government website (http://weather.gc.ca/mainmenu/weather_menu_e.html) and read the corresponding ATOM feeds to obtain the weather for location the client has requested. According to the Canadian government weather website, there are many kinds of weather information that can be extracted from the website, such as local forecast information, weather alerts, marine alerts, but this weather service will only provide the weather information that is deemed important to the primary stakeholders (details in the requirements section). This will achieve the objective of providing a resilient weather service that provides quick and non-complex updates.

The purpose of the product is to present users with real-time and reliable weather data, which one can access quickly through a series of short interactions (i.e. via mouse clicks or touch clicks depending on the client device). If time allows the project scope would try to include alternative computing devices as clients such as mobile devices and OS's other than Windows. The following sections describe the design, implementation, and testing of the Resilient Weather Service.

Requirements Specification & Design

List of Stakeholders

The following are a list of stakeholders in the resilient weather system project. These consist of the main users of the system and would benefit the most from its use. This list was compiled via extensive research and interviews of various weather professions around the world. Most responded positively to the idea of a quick and easy to use weather system.

- Primary stakeholders would be:
  - Meteorologists
  - Canadian military personnel
  - National Park Staff
  - General public who need real-time, reliable, and quick weather information
- Secondary stakeholders would be:
  - General public
  - Government employees maintaining the ATOM feeds on the government weather website

Requirements

The resilient weather system requires a client, server and multiple forms of Web API to retrieve significant weather data. The main server needs to retain weather data and provide that data to clients that request it. It also requires some sort of "heartbeat" function to acknowledge if a server is online or down. According to the classification of a resilient system the weather service needs to account for unexpected process terminations, resource unavailability, network failures, server failures, and file storage permissions. The weather service requires a user interface that allows for setting user preferences such as city and location. This data will be presented in the UI and will be refreshed whenever the user clicks on the button for feeds or quick weather. Following are the features that are required to be implemented for the user:

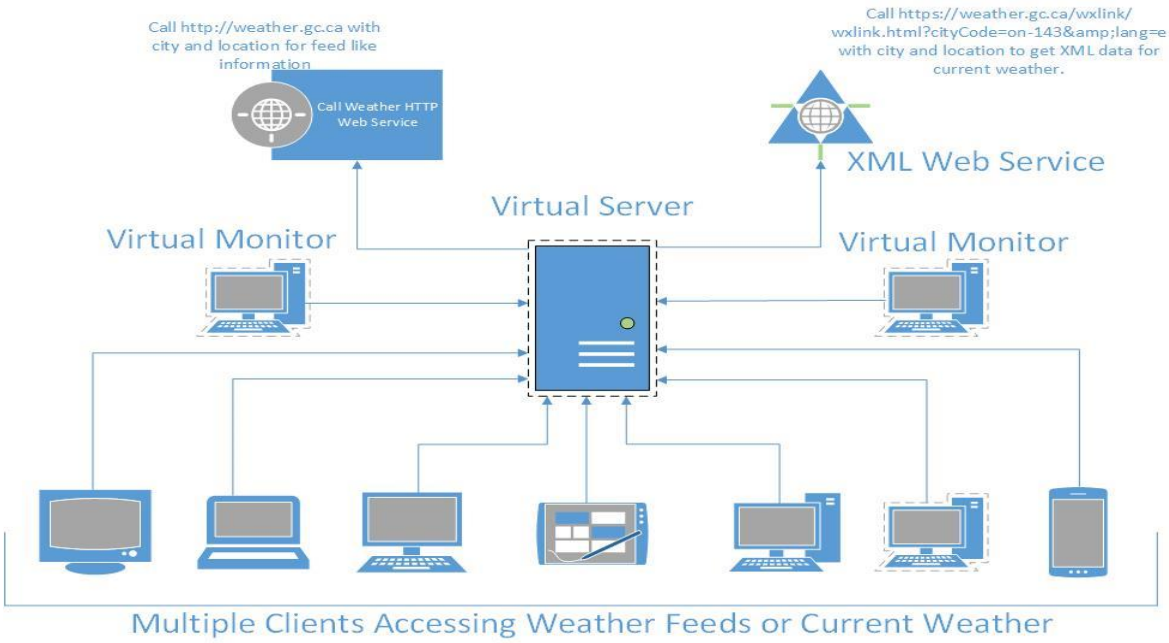
Weather Feeds

- Current Conditions & Forecast Feeds
  - Provide current weather conditions of the city specified (i.e. temperature)
  - Provide a 7-day forecast with a description of the weather for each day
- Warning Feeds
  - Provide public weather warnings from the Government of Canada
- Marine Feeds
  - Provide warnings for marine locations, for example ice warnings and/or snow squalls.
  - Provide weather conditions for marines, which would include winds, waves, visibility

Quick Weather

- Provide current weather details such as temperature, wind chill, mini-picture of weather for current city

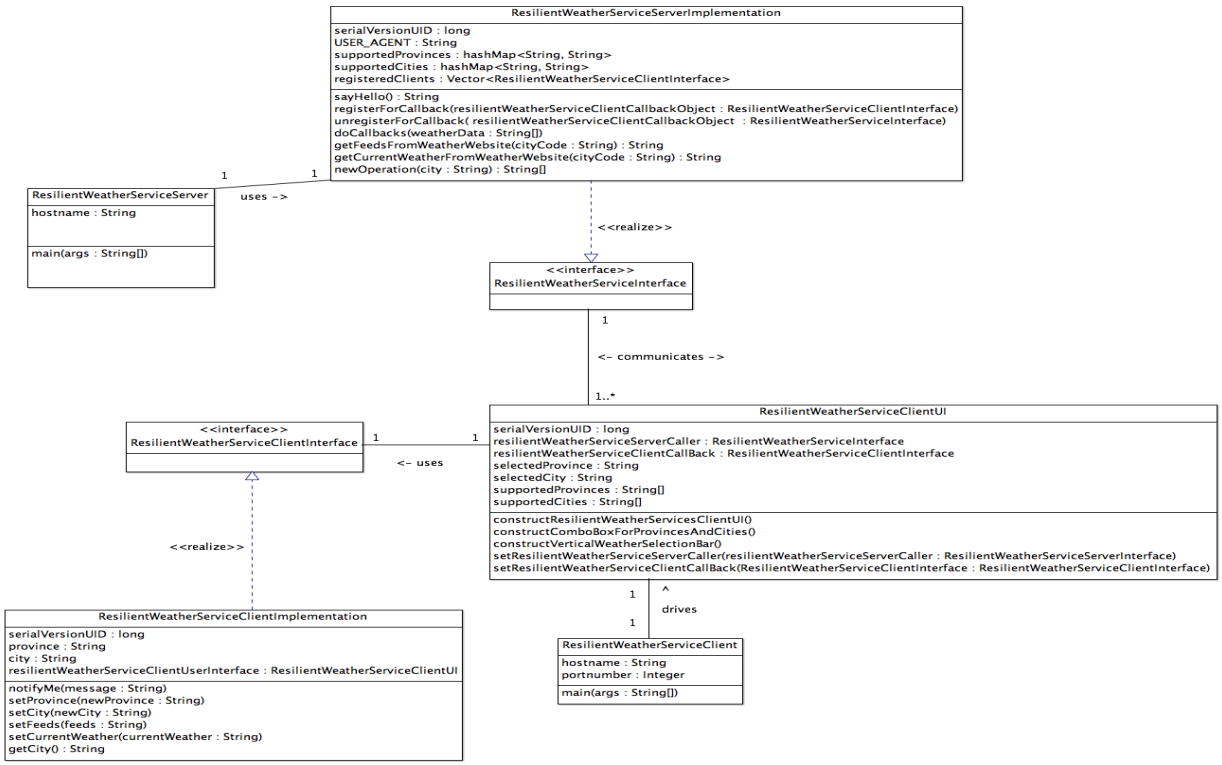
Architectural Design Diagram



(Clear image of the architectural design can be found in *Appendix A: Architectural Design Extended*)

This architectural design diagram above shows how our design interacts with all aspects of the system. The main server is the one that is performing all the requests to retrieve the data from the web services and parsing the data before providing it to the clients. We have implemented 2 virtual monitors that acts as monitors for the main weather server, these monitors are responsible for restarting the server when it crashes for any reason. This is the aspect that provides a heartbeat for the system allowing it to be resilient. All the clients are connected to the main weather server using Java RMI, and all the clients requests information from the server. Each of the clients have their own user interface that is used to store the weather information that is retrieved from the server. The user interface on the client side also provides the option to configure a weather for the city of their choosing.

Class Diagram



(Clear image of the class diagram can be found in *Appendix B: Class Diagram Extended*)

This class diagram represents how the connections between the client, server and user interface work. From the diagram we can see that we have 3 main classes ResilientWeatherServiceClient, ResilientWeatherServiceMonitor and ResilientWeatherServiceServer that are driving the main interaction between the server and the clients. The ResilientWeatherServiceClient class is responsible for constructing the ResilientWeatherServiceClientUI object which is used to setup the user interface with default weather settings and also registering with the server with the information that the server needs. The ResilientWeatherServiceMonitor class is responsible to monitor the Java RMI ports that the server is initialized on and make sure that there is a server connected to the port at all times. Once the monitor has detected that the register is available it restarts the server again on the same host and port. This makes sure that the system is resilient in the sense that the server will always be online and able to retrieve and send data. There are also two interfaces that are used to interact with the server and client to send the data from one to the other. The data that is sent are the feeds and the current weather that the server provides. The server retrieves this information from the government websites based on the city code that is passed in.

Overall User Interface Design

Figure 3Shows the user interface, with the current weather tab active

Figure 1Shows the user interface, with the settings tab active

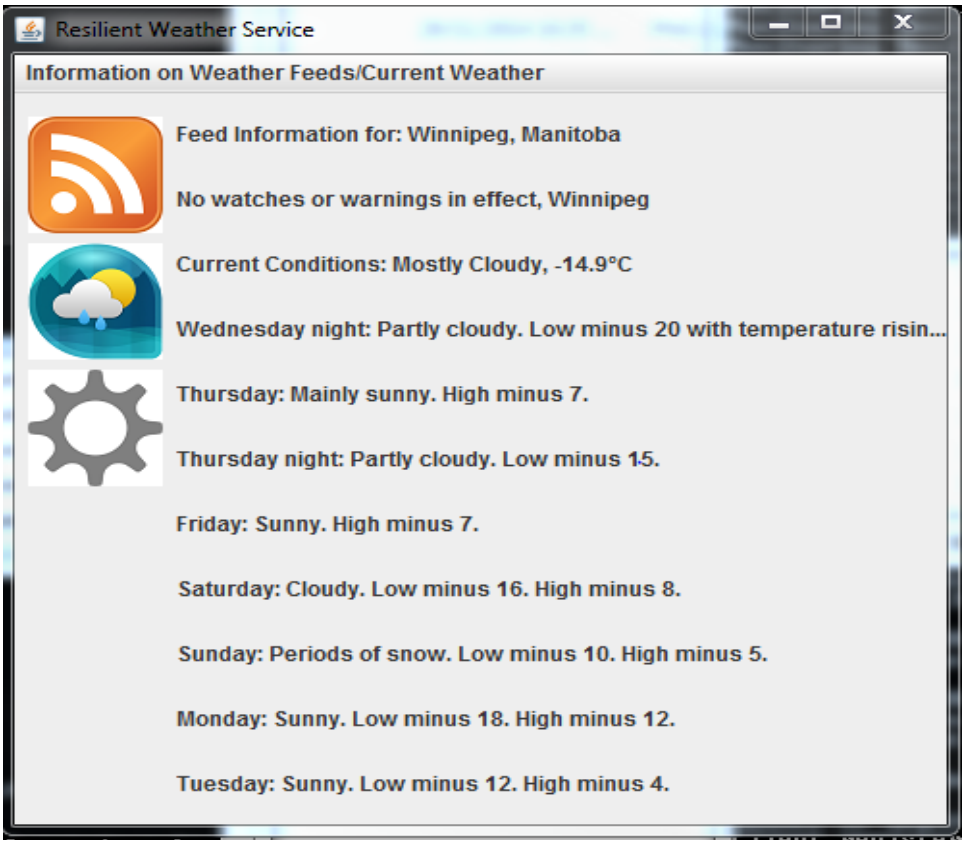


Figure 2Shows the user interface, with the feeds tab active

Figures 1, 2 and 3 show the user interface that the user will be provided when the client is started, this user interface allows the user to navigate through the feeds and current weather information. The settings window shows the configuration that the users are capable of setting to retrieve the weather for. When theses settings are changed all the feeds and current weather windows on the user interface are updated with the latest data of the city that it was changed to. Also when the feeds or current weather tabs are clicked the information is refreshed providing the most up to data information for the user.

Overall Resiliency Design

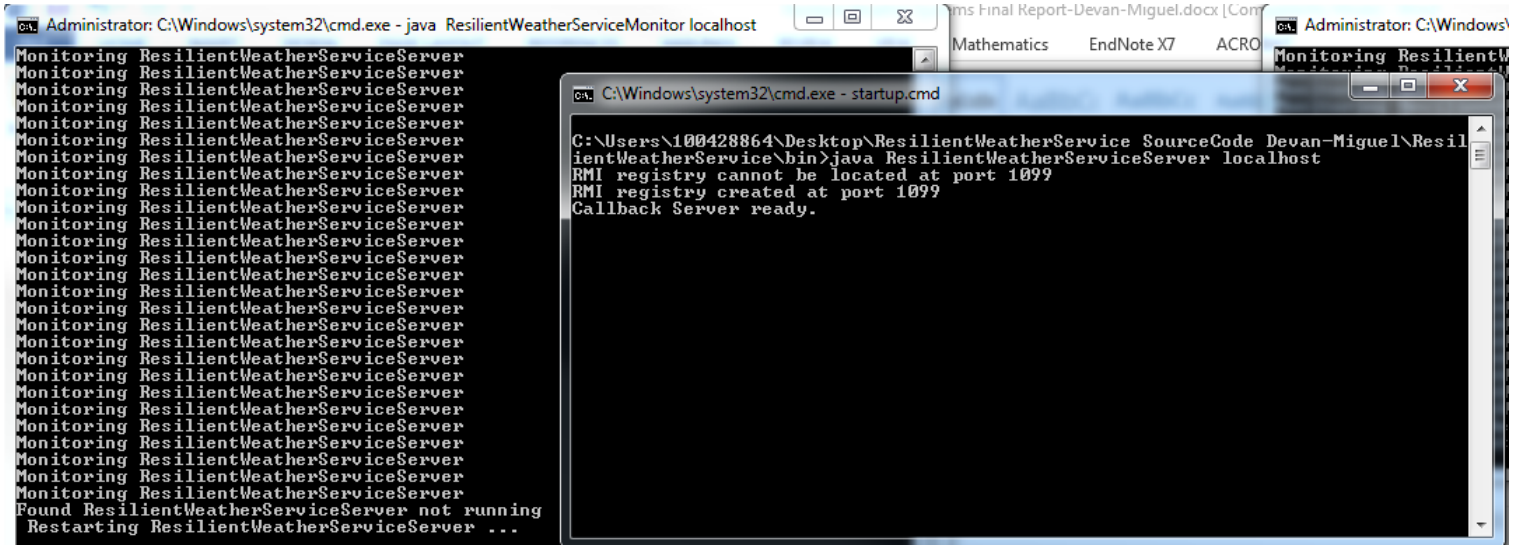


Figure 4Shows how the system is resilient, this shows that when the main crashes or is terminated the monitor will bring the server back online and the server will be able to start taking requests from the same clients that were previously active on the old server.

Figure 4 represents how the weather system is making use of the resiliency design. In the case above the main weather service server was terminated to show how the monitors would react to this. As you can see from the image one of the monitors detected that the main server was not running and restarted the server again. At this point the clients would now start to communicate with the new server that was started and would retrieve data from the new server for the weather feeds and current weather data. The way that the monitor detection works is with the use of Java RMI, the monitor continuously checks that the host and port that the main server is running on are register to the Java RMI. When it detects that there the registry is empty it start up a new instance of the server using a scheduler. More details on the full implication can be found in the source code that is provided.

Implementation

Description

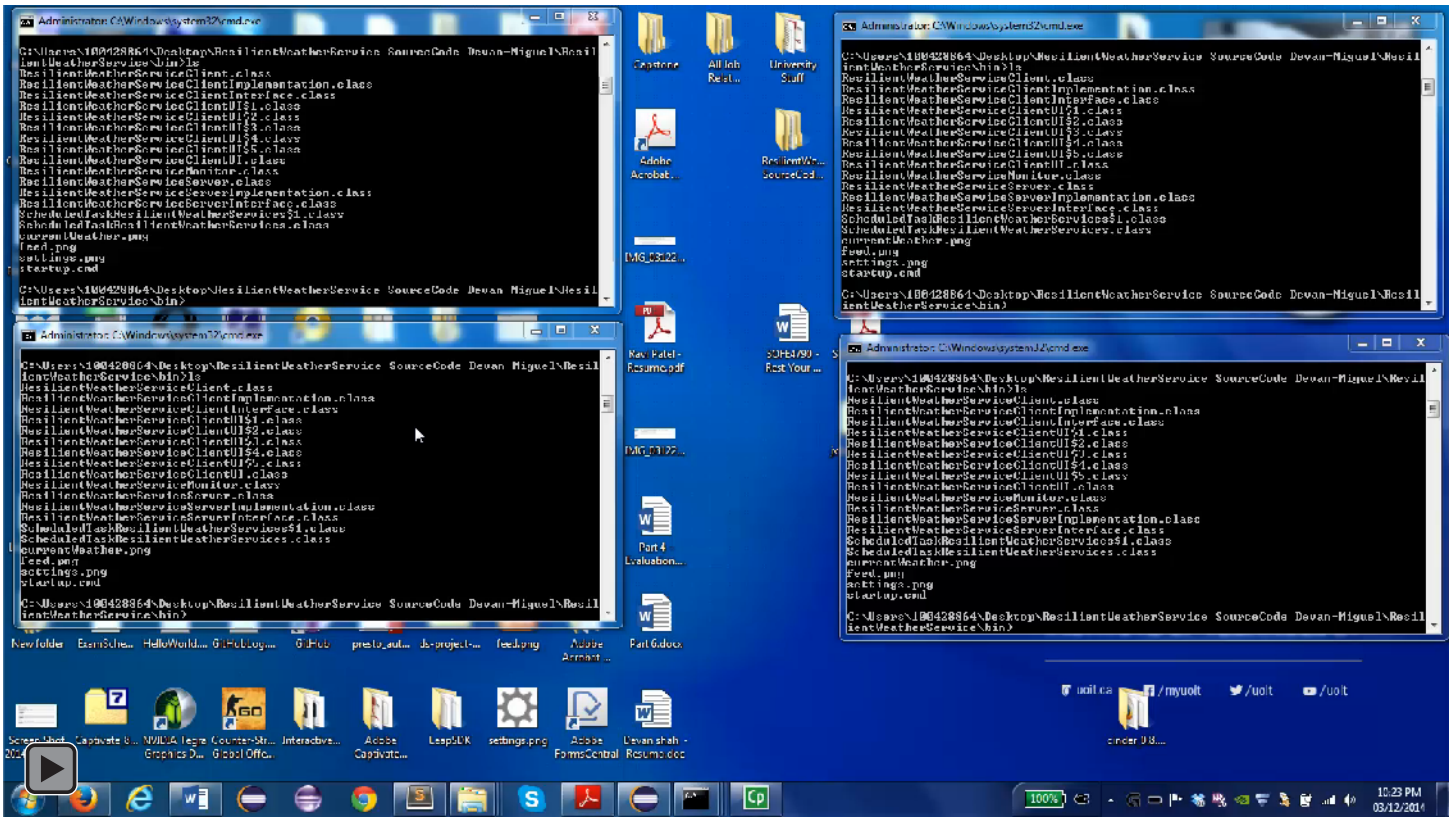
Implementation of the weather service will closely follow the defined preliminary architecture. Our proposed solution will utilize a distributed system architecture of a pervasive nature and leverage service oriented architecture technology with XML, ATOM feeds for receiving data from already existing weather XML web services, and HTTP web services. Our weather service will use embedded Java HTTP protocols to retrieve weather data from the existing XML/HTTP web services. This will provide information for constructing feeds and gathering current weather data for specific locations. Feeds are setup by users for specific locations and will be updated every time there is a user request for new information. This provides users with the most up to date weather data possible.

The client would register with the server and the server would reply back with the location information that the client needs. This way the server can send back information to the client whenever there is an update for the location the client is registered for. On the client side there will be a validation step to make sure that the settings that are present in the client match the information that the server is sending back (location matches). This will be done by the use of a data structure on the client side that will store the settings for the feeds that users require. The feed settings contain the location for which to receive weather data. This will be done by making use of a call back client/server setup (i.e. Java RMI), as the client needs to stay registered to the server to receive up to date data on feeds.

The server would request new data whenever registered clients receive the prompt from the user to retrieve new data (i.e. button click on Feeds or Quick Weather) in the UI. The server will be set up as a virtual server and 2 virtual monitors on the same workstation as the server. The 2 virtual monitors will continuously check the main virtual server of the weather service every few seconds to make sure it is online and able to accept transactions. In the case that the server has crashed because of unexpected process terminations, resource unavailability, network failures and/or server failures one of the monitors will be able to bring the weather service server and weather service back online. This is possible because the virtual server and the 2 virtual monitors are on the same main workstation (physical computer).

The server/client architecture for the weather service will be making use of a Java RMI to accomplish the task of constructing the connecting with multiple clients over the Internet. The implementation will be making use of multiple open source java libraries for XML/HTTP Call data parsing. For XML parsing we are planning on using the Java Apache XML parser Xerces (open source) to parse the data that retrieved from the web API's. We will be making use of the build in java libraries for HTTP calls (i.e. get), this will allow a connection to the website to get the data for the weather in specific locations. This fully outlined solution will be resolving the issues of users requiring quick weather data in a timely manner, and through the use of heartbeat monitoring via two virtual monitors, the weather system will achieve the objective of being resilient.

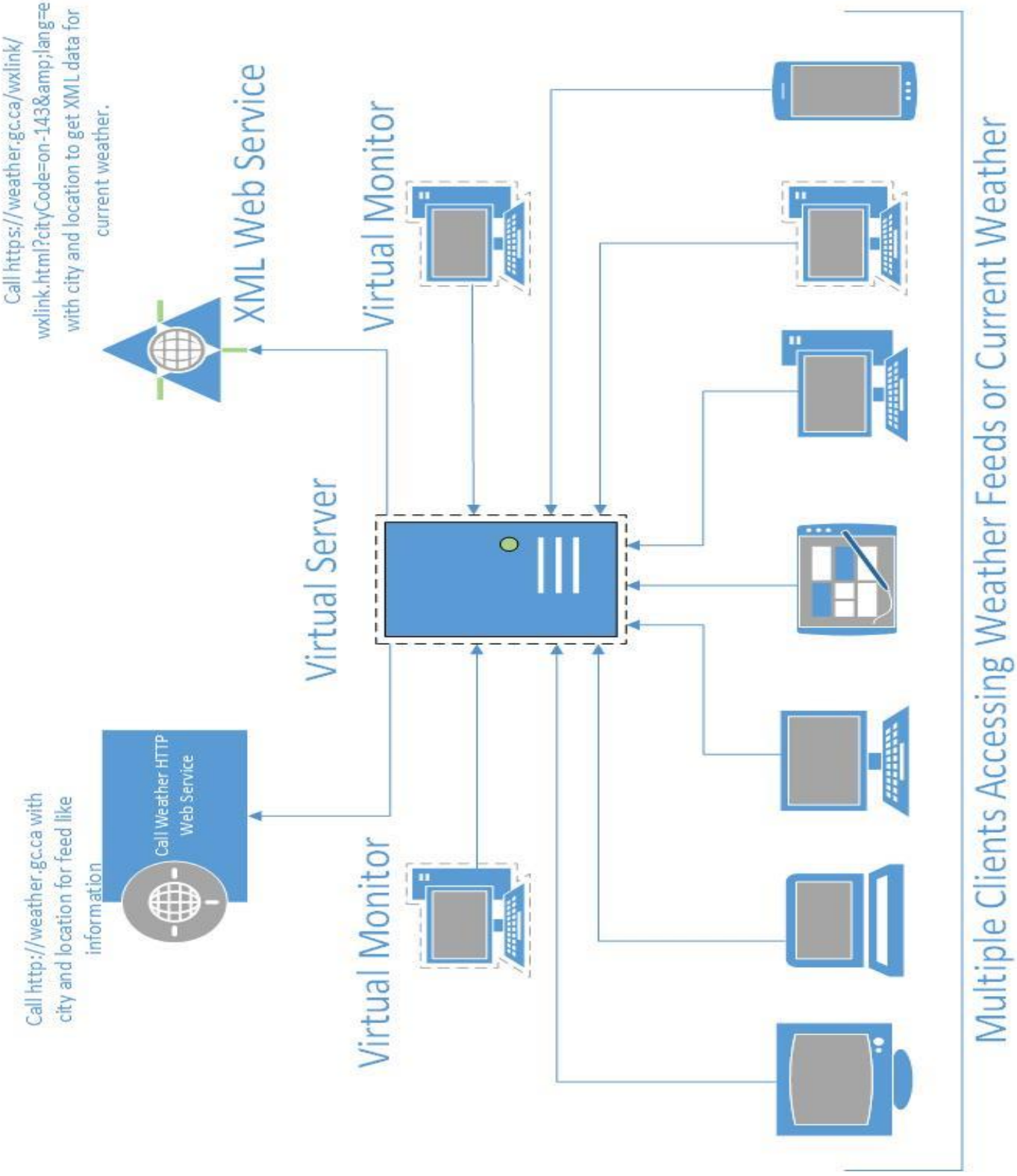
Video of Prototype



Conclusion

Overall, the system as described above achieves the main objectives of being a resilient weather system. It provides users with useful feeds for weather warnings, weather forecast, and marine warnings. It also provides users with quick weather updates for the selected city. This is a great solution for the primary stakeholders as described above, and reduces “information clutter” present in current weather systems. It has been tested extensively for resiliency and usability, and it will be a useful weather system for the future. If there was more time allocated for this project, more focus would be allocated to refining the user interface and more network testing to allow for more clients.

Appendix A: Architectural Design Extended



Appendix B: Class Diagram Extended