SOFE 4790 Project Proposal: Resilient Weather Service

Group Members:

Devan Shah	100428864
Miguel Arindaeng	100394094

Abstract

This proposal is intended to give the reader an 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. This service will be able to sift through and extract important weather data, as there is a demand that this type of information be presented accurately to the user. Weather data such as the current day's weather, the seven-day weather forecast, and wind speeds are some of the types of information that will be presented. This data will be presented in the form of ATOM feeds and service updates to provide current weather data for users. Users will be able to setup personal ATOM feeds in order to tailor what information is presented to them.

This proposal 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 Statement

Substantial growth and increase in the field of web-based technology has impacted the timeliness of valuable data being transmitted to users. In the context of weather data, there has been a tremendous degradation in the amount of useful data presented to the user. Most of the weather websites that exist give substantial amounts of data at one time that it is hard to sift through and extract what is important. Weather data such as the current day's weather or a 7-day forecast are examples of more important data to be considered. Since 7-day weather and hourly trends change rapidly over time, this data would require frequent updates to display their information – this can be done efficiently through an auto-update method. It would allow the user to retrieve quick updates on the changing weather for interested locations, as well as providing updates on the weather through feeds and real time updates. This saves the user time from accessing the website repeatedly to get updates on the weather. The purpose of this product is to present users with real time weather data, which one can access quickly with a mouse hover or single click. Our implementation of rapidly accessible weather will start as a Windows application and then will eventually be ported to smartphones, tablets and other operating systems. If time allows our implementation will also provide weather warnings/watch, small geographic warnings, large geographic warnings, marine warnings/watch and marine forecast.

Requirements

A resilient weather service 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 visually appealing UI that allows for setting user preferences such as city and location. The UI should be able to stay hidden in the background to avoid interfering with the user's normal tasks. The user will have a wide range of functionality ranging from feeds to current weather data. This data will be presented in the UI and will need to be refreshed in real time. Following are the features that are required to be implemented for the user:

Constantly Updating Feeds

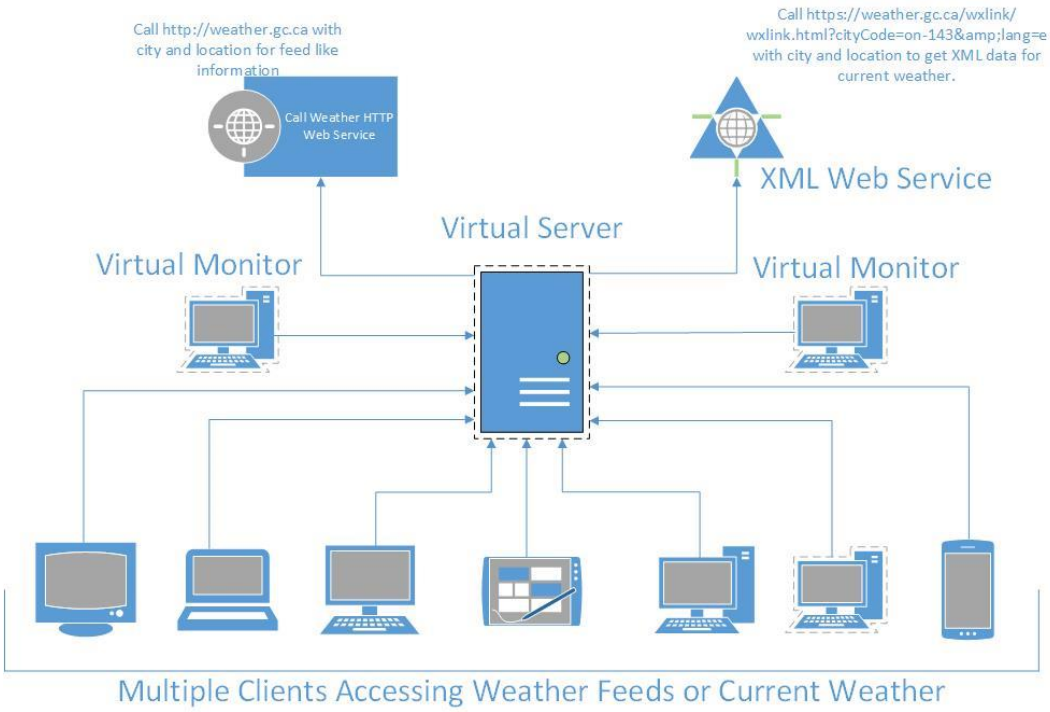
- Current Conditions & Forecast Feeds
  - Provide warnings/watch for town/city
  - Provide current weather conditions (weather condition, temperature, humidity, wind speed and wind chill)
  - Provide a 7-day forecast with images representing the weather.
- Warning Feeds
  - Provide warnings for small geographic locations; coverage is same as mentioned in "Current Conditions & Forecast Feeds"
  - Provide warnings for large geographic locations; this would include public warnings.

- 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 and 5 day out looks.

Quick Weather

- Provide the current weather only when users asks for snapshot of the day.
  - Provides the user with the weather for the day, (morning, afternoon and night)

Preliminary Architecture



(Clear image of the preliminary architecture can be found in ***Appendix A: Preliminary Architecture Extended***)

Proposed Solution

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 imbedded 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 or reconstructed repeatedly over time by the server. 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 for and the type of data for the feed (current conditions & Forecast feeds, warnings feeds and marine feeds). This will be done by making use of a call back client/server setup, as the client needs to stay registered to the server to receive up to date data on feeds.

The server would request new data continuously for all of the registered clients that exist in the data structure on the server side. The server will be set up as a virtual server and 2 virtual monitors will be setup on the same workstation as the virtual server. The 2 virtual monitors will continuously check the main virtual server of the weather service every 1 sec 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 will also store each of the transactions (weather retrieved from the web services) to a file as they happen in the case there is a crash on the server for any reason, the already retrieved client information will still be accessible after restart.

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. Our 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, the implementation is planned be a resilient system.

Appendix A: Preliminary Architecture Extended