CIS 4911 - Senior Project (U01)

Web Dashboard for Addigy

Javier Carmona

Francisco Marcano

October 27, 2014

Jason Dettbarn

**Design Document**

# Copyright and Trademark Notices

All Addigy specific logos and slogan material is copyright Addigy, Inc. All other materials developed for this project is available open source under MIT License:

http://opensource.org/licenses/MIT

# 

# Abstract

This document will help to outline and explain all of the functional components for the Web Dashboard for Addigy project. The document will be broken down into different chapters each of which will be in charged of explaining different aspects.

The first chapter includes an introduction, which will serve to explain the problem definition, the scope of the project, any terminology used and an overview of the rest of the document. Chapter two will explain the current system that is implemented. The third chapter talks about the project plan which helps to define the project’s organization, work breakdown, and cost. The fourth chapter talks about system requirements and functional dependencies, as well as any models in the system. Chapter five includes a glossary of any terminology used in this document. Finally, chapter six includes all of the different diagrams used to visualize the system.

# Table of Contents

|  |  |
| --- | --- |
| **Chapter** | **Page** |
| **1. Introduction** | **3** |
| 1.1 Problem Definition | 3 |
| 1.2 Design Methodology | 3 |
| 1.3 Definitions, Acronyms, and Abbreviations | 4 |
| 1.4 Overview of Document | 4 |
| **2. System Design** | **4** |
| 2.1. Overview | 4 |
| 2.2. Subsytem Decomposition | 5 |
| 2.3. Hardware and Software Mapping | 6 |
| 2.4. Persistent Data Management | 7 |
| 2.5. Security/Privacy | **9** |
| **3. Detailed Design** | 9 |
| 3.1. Overview | 9 |
| 3.2. Static Model | 10 |
| 3.3. Dynamic Model | 12 |
| 3.4. Code Specification | 14 |
| **4. Glossary** | **17** |
| **5. Appendix** | **18** |
| 5.1. Appendix A | 18 |
| 5.2. Appendix B | 19 |
| 5.3. Appendix C | 24 |
| 5.4. Appendix D | 25 |

# 

# 

# 1. Introduction

The Dashboard for Addigy is an application that facilitates the illustration and management of computer data provided by the underlying Addigy framework. This document outlines all of the design methodologies used, as well as all of the underlying models and system architectures created for the successful completion of this software.

First we will talk about the problem this technology is trying to solve and how it is trying to solve it. Finally, we will talk about the different design methodologies used in order to successfully implement the software solution to the problem. At the end of this document, acronyms, definitions, and abbreviations used throughout the document for certain words can be found. Along with the acronyms, definitions, and abbreviations, an overview of the system can also be found.

## 

## 1.1. Problem Definition

Many industry professionals in Information Technology fields lack the tools necessary to successfully manage mac computers in their networks. The Addigy product aims to solve that problem by providing statistical data on any machine that is present in their network.

Currently the Addigy product does not have any user friendly way of showcasing their statistical data to these high level professionals in the industry. The Web Dashboard for Addigy aims to solve that problem by showing the aforementioned statistical data in a nice and clean format, that is easy for any professionals in the industry to follow.

## 1.2. Design Methodologies

The way this project was approached was a little different. Due to the shifting of design methodologies in general in favor for a more agile development approach, this project has both a waterfall approach (when it comes to documentation) along with an agile approach (when it comes to implementation).

Due to the differences of both methods, documentation needed to become as flexible as possible, mainly due to the fact that most implementations were not concretely defined until it was time to implement them (due to the agile nature of agile development), this made creating design documents difficult from the base system requirements.

A number of models were used to represent the system. UML Use Case Diagrams were created in order to represent all interactions between the system administrator, the single machine user, and the Addigy Dashboard. UML Class Diagrams were created in order to represent the different objects that exist in the Dashboard. Each scenario was modeled using UML Sequence Diagrams.

## 1.3. Definitions, Acronyms, and Abbreviations

Some of the acronyms or abbreviations used throughout this document can be found detailed here:

* **WDA**: Web Dashboard for Addigy
* **REST**: Representational State Transfer
* **API**: Application Programming Interface

## 1.4. Overview of Document

This document will serve to outline and describe all of the project’s System Implementations and Design. Things that can be expected include the following:

* High Level description of the system
* Decomposition of all of the subsystems that make up the main system
* Hardware and Software mapping
* Data manipulation and transformation
* System security and privacy
* The static, as well as the dynamic model of the system
* Code specifications (constraints, conditions, invariants, etc.) for the main control section in each system

# 

# 2. System Design

This chapter explains how the whole system was broken down into its different components. It will explain their purpose and how they work together.

## 2.1. Overview

The Addigy Web Dashboard is a web based dashboard developed using a number of different systems and services. It followed a Three-tier architecture in order to separate the Presentation Tier from the Logic Tier and the Data Tier, this is shown in the System Design diagram of **Appendix C**.

The presentation tier consists of:

* The Web Dashboard Multi-Machine View has cummulative information for all the machines of an organization.
* The Web Dashboard Single-Machine View contains real-time data regarding one specific machine as well as ways to interact with it.
* The Agent Chat Interface provides a two-way communication channel for the user administrator and the person using the machine being remote controlled.

The Logic Tier consists of systems that work behind the scenes in order to update and handle interactions for the presentation tier. The systems in it are:

* The Multi-Machine controller is in charge of updating the dashboard and responding to user interactions such as filtering machines by their state.
* The PagerDuty Integration allows the Multi-Machine controller to show PagerDuty alerts and allows the controller to act upon them.
* The Single Machine Controller handles user interactions with the single-machine dashboard and communication with the PubNub services.
* PubNub provides a channel through which the Single Machine Controller and the Single Machine Agent can communicate in order to exchange data and commands.
* The Single Machine Agent allows the administrator to extract information from the target machine and allows the user of the machine to communicate with the administrator.

The Data Tier contains services that are in charge of maintaining persistent information about the application. These are:

* The Addigy Data API holds cummulative information about all the machines of an organization.
* The PagerDuty Data API contains alerts that have been raised by the system and provides ways of acting upon them.

## 2.2. Subsystem Decomposition

The **Web Dashboard Multi-Machine View**and**The Web Dashboard Single Machine View** were developed using HTML and CSS. It takes advantage of frameworks such as BootStrap and JQuery to display functionalities and data processed by the Logic Tier. These systems require the files to be hosted in a web server, preferably a LAMP or WAMP server with the latest version of PHP.

The **Agent Chat Interface** provides a User Interface for the user of one computer to communicate with an administrator that is using the machine remotely. The User Interface is an extension It was developed using Java 8 and therefore requires JRE8 or greater in order for it to run.

The **Mutli-Machine Controller** serves as way for the administrator to retrieve and interact with the data offered by the Addigy and PagerDuty Web Services. It was developed using the AngularJS Framework. The controller is responsible for connecting to the Addigy Web Services and retrieving information regarding all the machines for an organization. It also retrieves any alerts that may have been raised through the PagerDuty services in order to be shown through the Web Dashboard Multi-Machine View. The controller handles administrator’s interactions such as filtering machines by groups, acknowledging, or escalating alerts.

The **PagerDuty Integration** provides the Multi-Machine Controller with ways of retrieving and interacting with alerts raised through the PagerDuty services. It uses a Javascript API provided by PagerDuty in order to abstract away the API functionalities to the Multi-Machine Controller.

The **Single Machine Controller** provides information and functionality to the Web Dashboard Single-Machine View by handling communication with the PubNub services. The controller is in charge of listening to a PubNub channel for any information published by an individual machine. It interprets the messages published on the channel and takes appropriate action based on the content of the messages. Some of the messages require the controller to update the view such as a new screenshot that has been taken of the target machine, and some other messages require the controller to publish messages to the channel regarding its current status. In addition, the controller handles user interactions such as running a command by publishing a message to the PubNub channel to be interpreted by the Single Machine agent. This controller requires a user account to be setup in PubNub and API keys assigned to the account.

**PubNub** is a service that allows online communication using a Publish-Subscribe architecture over a highly scalable distributed system. It allows machines wishing to communicate to open a logical channel and register/subscribe to it. By subscribing to the channel, a machine connects a TCP Socket to the PubNub services through which it will receive any message published to the channel. A machine can also publish messages to the channel and these will be delivered to all the subscribers.

The **Single Machine Agent** is in charge of collecting information about a machine and executing tasks on behalf of the administrator. It subscribes to a PubNub channel in order to communicate with the Single Machine Controller. Whenever an administrator subscribes to the channel, the agent starts actively monitoring the machine for machine specific data such as memory usage and cpu utilization, and publishing it to the channel. The agent also interprets messages received from PubNub that have been published by the Single Machine Controller such as a command to be executed, and acts upon the message received. The agent was developed using Java 8 and therefore requires JDK8 or greater installed on the machine for it to work.

The **Addigy Data API** remotely accesses the machine of an organization at time intervals in order to collect machine specific information about them. It stores all the information in databases and catalogues the data by organization. This service makes the data collected available through different Web Service API calls.

The **PagerDuty Data API** allows different applications to create alerts and provides different ways of interacting with them through Web Service Api Calls.

## 2.3. Hardware and Software Mapping

The hardware and software that were used in this project were integrated in the following ways:

* **Server Machine**
  + A linux virtual machine was used to host all of the files and folders needed to successfully run the application. The server framework used was an Apache server running PHP. The apache server acts as the link between our virtual machines virtualized hardware and the application software code. Both the **Multiple Machine View**  and the **Single Machine View** use this server in order to server the files to the client machine.
* **Client Machine**
  + All of the users that request the pages from our server run a copy of our application in their machine. Thanks to modern browser technology, they act as the link between the client machines hardware and applications JavaScript logic. Both the **Multiple Machine View** and the **Single Machine View** subsystems and their respective **Controller** subsystems utilize the client machine in order to run the logic necessary to drive the application.
  + The **Agent Chat Interface** and the **Single Machine Agent** subsystems which run directly on the client machine also depend on the hardware resources of the client machine. This machine could be a different machine from the one that the Multiple and Single Machine views depend on.
  + The **PagerDuty Integration** subsystem depends on the machine that the Multiple and Single Machine views run in. Since this subsystem is directly tied to the Multiple Machine View and Controller subsystems.
  + The **PubNub** subsystem creates a link between the software that the **Single Machine Agent** depends on and the software that the **Single Machine View** and **Controller** depends on.

## 2.4. Persistent Data Management

The application currently doesn’t handle any data persistence directly. The only persistent data that is utilized is through the use of the **PagerDuty** **Integration** subsystem. The application currently stores data based on certain machine error events that can happen at any point. The application also polls the persistent data from the PagerDuty API and makes it available to the user.Below is a data dictionary for all of the data types that are retrieved:

***--Error*** ***Data Type***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Length** | **Type** | **Rule** |
| Title | 120 | varchar | None |
| Description | 500 | varchar | None |
| Incident Key | 120 | varchar | Primary Key |
| Assigned User | 120 | varchar | Foreign Key |

***--User Data Type***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Length** | **Type** | **Rule** |
| Name | 120 | varchar | None |
| Email | 500 | varchar | unique |
| User Id | 120 | varchar | Primary Key |

## 2.5. Security/Privacy

The application currently does not handle authentication as this was not a feature that our mentor needed for the initial version of the project. Currently the application is keeping track of the username that get used when logging in without authenticating against a persistent data system. This is quick feature to add in future revisions as the application already has a notion of what a user is.

For data encryption, is not being implemented in this version, mainly because is not needed. Since the application is not not using any Data Persistent systems, it is not handling any user or system sensitive information.

The default Linux firewall settings were used in the Virtual Machine that was provided to run the application. No other modifications to this were done to the Virtual Machine. Security servers were not used in this application.

# 3. Detailed Design

This section explains the specific details of the subsystems that make the Addigy Web Dashboard.

## 3.1. Overview

The **Multi-Machine Web Dashboard** and **Single-Machine Web Dashboard** follow a Model-View-Controller design with the Multi Machine Controller and Single Machine Controller. By displaying the data in a user interface, Dashboards serve as the View in this paradigm. The dashboard show their information by accessing HTML documents through a browser and applying CSS files to it in order to achieve the desired look and feel.

The **Multi Machine Controller** and **Single Machine controller** are in charge of interacting with Web Services and APIs in order to retrieve information for the Dashboards or handle interactions. This makes the systems controllers for the Model-View-Controller paradigm of the Dashboard.

The **Single Machine Agent** run on the machine being serviced and acts on behalf of the administrator on the machine. The agent acts as a publisher and subscriber for the PubNub Publish-Subscribe architecture.

**PubNub** uses a distributed system to handle one-to-many communication between different machines. It follows a Publish-Subscribe architecture where individual machines subscribe to a channel and publish messages to it while delivering published messages to all the subscribers.

The **Addigy Data API** and the **PagerDuty Data API** are APIs used to store and retrieve information about the machines. Addigy stores auditing data and PagerDuty stores alarms raised by the system.

## 3.2. Static Model

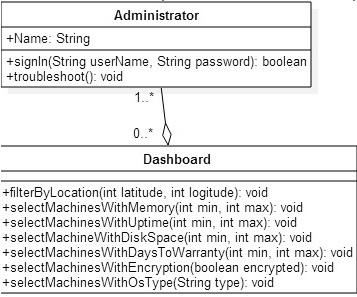
In this section, we would be looking at different parts of the system with subsystem specific class diagrams, refer to **Appendix C** for a complete Class Diagram of the System.

The Multiple Machine Dashboard - Administrator Class Diagram shown below shows the relationship between the **Multiple Machine Dashboard** and the Administrator.

The administrator has a name that serves as an identifier and can login or troubleshoot machines on the system by looking and interacting with the dashboard.

The dashboard supports a number of operations based on the machines it collects from the Addigy Data Api:

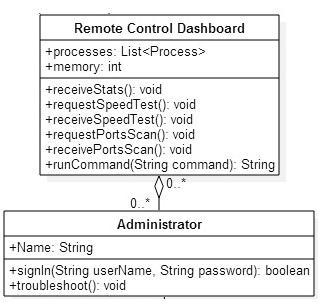
* filterByLocation - Selects the machines that are within a small distance of the specified latitude and longitude.
* selectMachinesWithMemory - Selects the machines that are within the minimum and maximum range of memory specified.
* selectMachinesWithUptime - Selects the machines that are within the minimum and maximum range of uptime specified.
* selectmachineWithDiskSpace - Selects the machines that are within the minimum and maximum range of disk space specified.
* selectMachinesWithDaysToWarranty - Selects the machines that are within the minimum and maximum amount of days to the day that the warranty will expire.
* selectMachinesWithEncryption - Selects the machines that is encrypted if a true value is passed to the operation or the ones that are unencrypted otherwise.
* selectMachinesWithOSType - Selects the machines that are currently running on the specified OS type.



*Multiple Machine Dashboard - Administrator Class Diagram*

As can be seen in the Remote Control Dashboard - Administrator Class Diagram below, the **Single Machine Dashboard** contains a list of processes currently running in a machine and how much memory is being used by the system. The dashboard could contain any number of system administrators. It supports the operations:

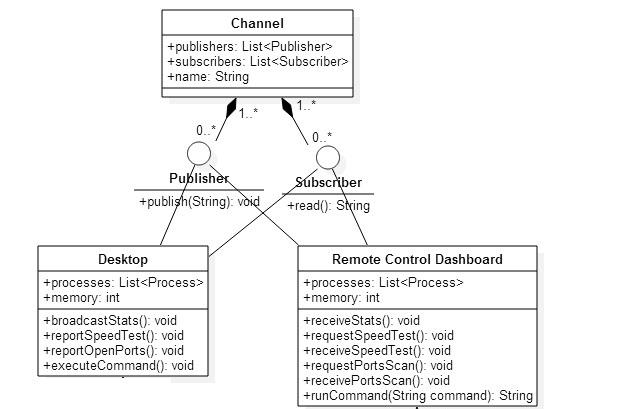
* receiveStats - Receives memory usage among other machine specific data being reported by the agent running on the remote controlled machine.
* requestSpeedTest - Requests the agent running on the remote controlled machine to run and report a speed test.
* receiveSpeedTest - Receives the results of a speed test that has been executed by the Single Machine Agent running on the machine being serviced.
* requestPortScan - Requests the Single Machine Agent to scan and report what ports are being utilized by the system and their protocols.
* receivePortScan - Received the results of a port scan that has been done on the machine being serviced.
* runCommand - Allows the dashboard to run a command on the target machine through the agent.



*Remote Control Dashboard - Administrator Class Diagram*

The Remote Control Dashboard and theDesktop use a Publish-Subscribe method modeled in the Publish-Subscribe Class Diagram below in order to communicate with each other.

The channel is provided by **PubNub** and it allows two machines to publish and subscribe to a channel. Messages published to the channel will be delivered to all subscribers. As seen in the class diagram, both the Desktop and Remote Control Dashboard are Publishers and Subscribers of the channel. The agent running on the desktop uses the channel to publish machine specific data, speed test results and open port scans results. The agent can also execute commands as received from the dashboard by the administrator.

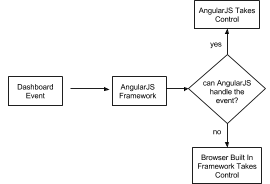


*Publish-Subscribe Class Diagram*

## 

## 3.3. Dynamic Model

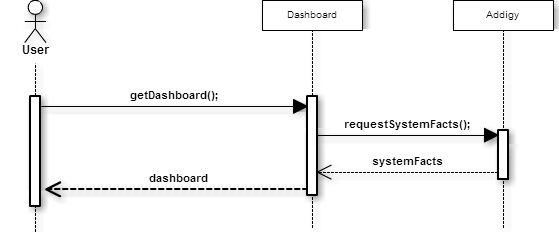
### Multiple Machine View & Single Machine View

The web dashboard Multiple Machine View andSingle Machine View both utilize the same control object. In this case the main control is handled by the AngularJS framework. 

Anytime there is an event or a state change in the view, the AngularJS framework intercepts it and acts upon it based on what the user configurations. If the event cannot be handled by AngularJS, then it lets the browser built in default controller handle it.

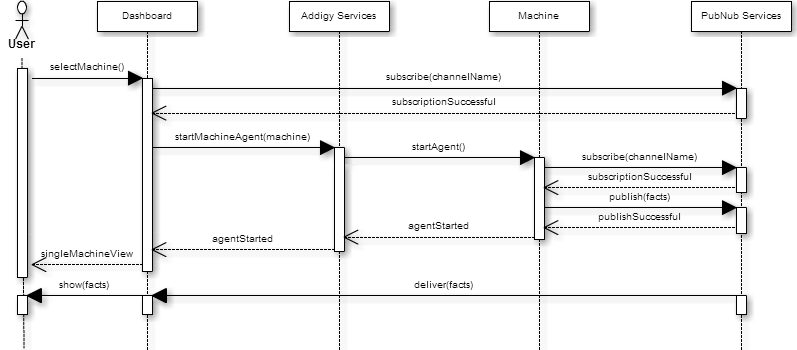
### Multiple Machine Controller

This is the controller that drives all of the flow for the Multiple Machine view. The main algorithm that this controller performs is the ability to filter machines based on the data that is currently available. It does this by checking to see if the machine data has changed in any way. If it has, then goes ahead and runs all of the different helper methods that modify the view based on the the new data that is available.



### Single Machine Controller & PubNub

This is the controller that drives all of the flow for the Single Machine view. The main algorithm that this controller performs is the ability to update all of the information for the view depending on the type of data that is returned from the **Single Machine Agent**. It does this by first checking an upcode style string which determines what type of information the Single Machine Agent returned with. This data exchange is performed by using the **PubNub** subsystem to communicate between the the Single Machine Agent and the **Single Machine View** subsystems.



### PagerDuty Data API & Addigy Data API

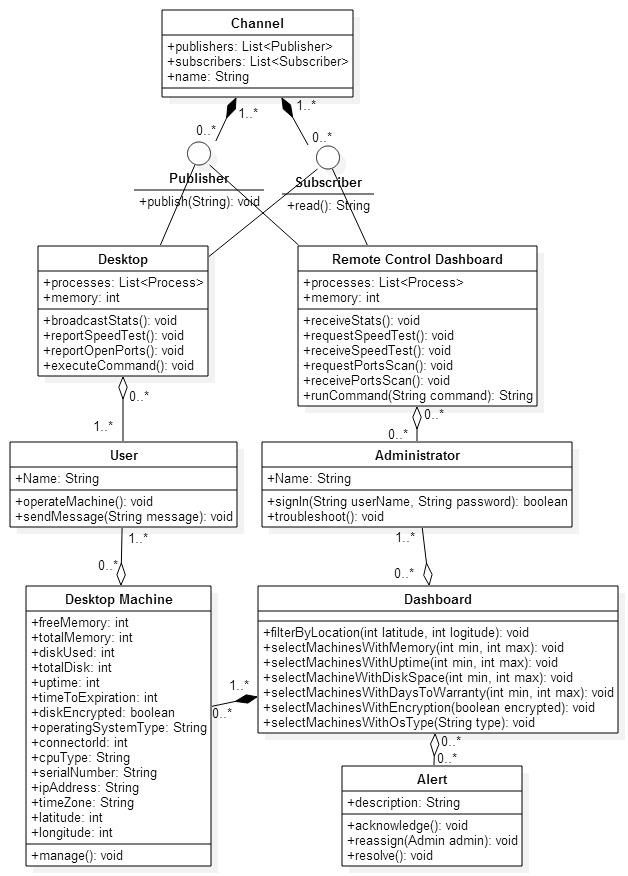
These subsystems use the **Multiple Machine Controller** to get data retrieved from them. This is done by calling a series of webhooks which return data depending on the subsystem that the Multiple Machine Controller is trying to call.

## 3.4. Code Specifications

### Multiple Machine View & Single Machine View

This views do not directly have any logic or code aside from HTML and CSS. The main control object for these two views is the AngularJS framework.

### Multiple Machine Controller



The Multiple Machine Controller makes use of three main class interfaces.

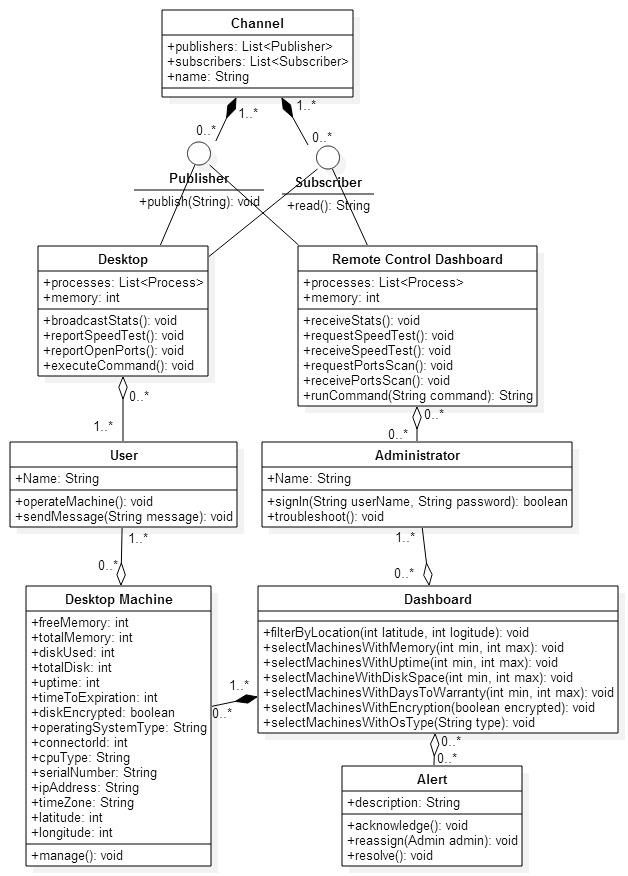
The first one is the Desktop Machine class interface. This class is used to describe all of the properties that a machine can have. This class interface will provide all of the necessary information to the rest of the class interfaces to provide a complete implementation of the Multiple Machine Controller.

The Dashboard class interface defines all of the functionality that is possible while on the Multiple Machine view. Most of the functionality outlined in the class definition is the ability to filter and select machines based on a type of machine property defined in the Desktop Machines class interface.

Finally,the Alerts class interface defines what and alert is and can do from the Multiple Machine view. The most important are the functional wrapper built around the **PagerDuty Data API** which allows the application to interact with the PagerDuty platform without any 3rd party support.

### 

### Single Machine Controller



The Single Machine Controller is made up of 3 class interfaces.

The first interface is the Desktop. This interface defines what a target machine is composed of and can perform. This essentially makes up the **Single Machine Agent**, in which the agent acts as the Desktop interface for the Single Machine controller.

The Remote Control Dashboard is the interface that defines what the web view would behave as from the perspective of the Desktop interface. The purpose of the Remote Control Dashboard is to, through the use of its functionalities, to collect information for the machine that the Desktop class interface is acting as.

Finally the Channel interface behaves as the link between both the Desktop interface and the Remote Control Dashboard interface. Facilitating the exchange of information between the two.

# 4. Glossary - define terms used in document, especially domain specific terms.

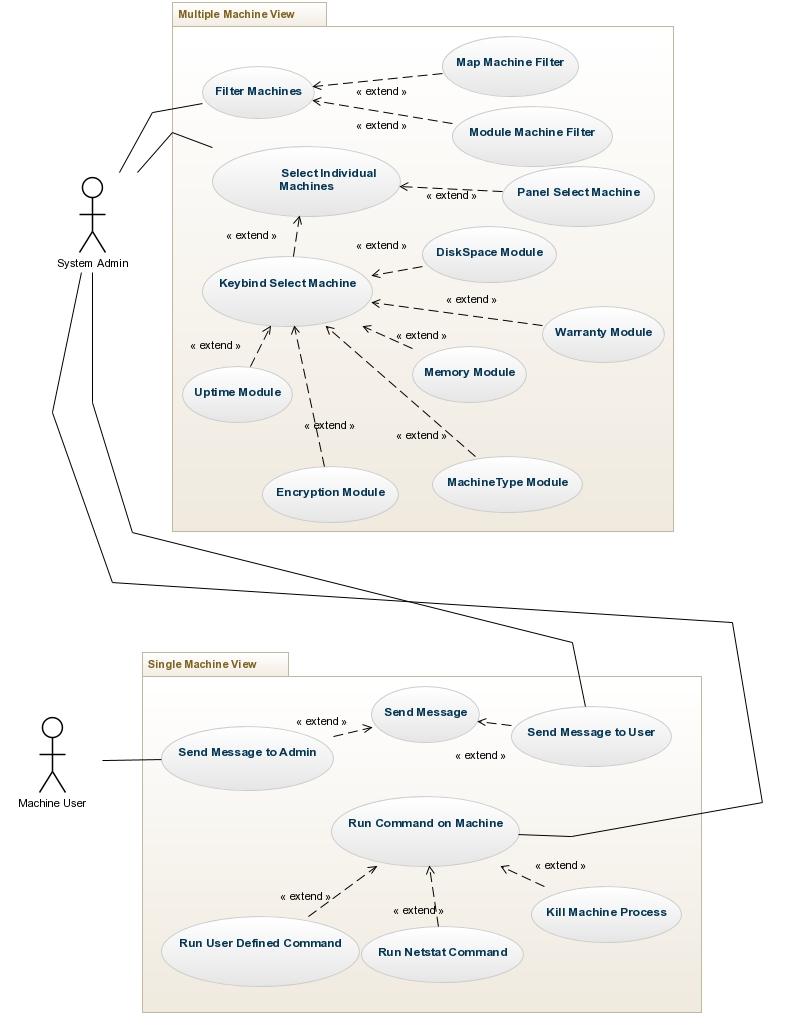
**Fact**: A piece of information about a machine.

**Agent:** A program that acts on the user’s computer on behalf of the administrator.

# 

# 5. Appendix

**5.1. Appendix A - Use Case Diagram**



**5.2. Appendix B - Use Cases**

***Log In***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is standing in the Login page.

**Flow of Events**

The user enters his username in the username field, his password in his password field, and clicks on the login button.

**Exit Condition**

The user is successfully logged in.

**Exceptions**

An error occurred and an error message is being displayed.

**Nonfunctional Requirements**

Login form must validate user’s input.

***Examine Dashboard***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is logged in

**Flow of Events**

The user logs in and is redirected to the dashboard page where the different statistics of the network can be seen.

**Exit Condition**

The user saw the status of the network.

**Exceptions**

None

**Nonfunctional Requirements**

The information of the dashboard must not be older than 30 minutes.

***Examine Machines in Location***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user has logged in and is looking at the main dashboard.

**Flow of Events**

User finds the location of the machines he wants to see in the map and clicks the specific location. The rest of the dashboard now displays information about the computers in the selected locations only.

**Exit Condition**

The user sees the information about the group of machines he is interested in.

**Exceptions**

None

**Nonfunctional Requirements**

None

***Acknowledge Machine Alerts***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at the errors in the main dashboard

**Flow of Events**

The user is presented with the alert in the main dashboard, the user then clicks on the acknowledge button for the alert of their interest

**Exit Condition**

The system received confirmation that the alert was acknowledge

**Exceptions**

The alert is not successfully acknowledge on the PagerDuty framework

**Nonfunctional Requirements**

The alert information should be polled every 10 seconds

***Resolve Machine Alerts***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at the errors in the main dashboard

**Flow of Events**

The user is presented with the alert in the main dashboard, the user then clicks on the resolve button for the alert of their interest

**Exit Condition**

The system received confirmation that the alert was resolved

**Exceptions**

The alert is not successfully resolved on the PagerDuty framework

**Nonfunctional Requirements**

The alert information should be polled every 10 seconds

***Re-Assign Machine Alerts***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at the errors in the main dashboard

**Flow of Events**

The user is presented with the alert in the main dashboard, the user then clicks on the assign dropdown and selects a the user he wishes to re-assign the alert to.

**Exit Condition**

The system received confirmation that the alert was re-assigned

**Exceptions**

The alert is not successfully re-assigned on the PagerDuty framework

**Nonfunctional Requirements**

The alert information should be polled every 10 seconds

***Examine Single Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user has logged in and is looking at a group of computers in a specific location.

**Flow of Events**

The use identifies the machine he wants to zoom into and clicks it. The dashboard now displays real time information about that specific machine.

**Exit Condition**

The user sees specific information to one machine.

**Exceptions**

Could not connect to the machine.

**Nonfunctional Requirements**

Shall not affect performance of the dashboard or the target machine.

***Start Process in One Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user has logged in and is looking at the specifics of one machine.

**Flow of Events**

The user enters a command in the command input field and submits the command for execution. The command is executed in the target machine.

**Exit Condition**

The user starts a desired process in the target machine.

**Exceptions**

Fail to deliver command.

Failed to connect to the machine.

**Nonfunctional Requirements**

The command should be executed on the target machine within 10 seconds.

***Terminate Process in One Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at a specific machine and its processes. User decided that one process must be terminated.

**Flow of Events**

User clicks on the process he desires to terminate and clicks on the Terminate button for that process. The target machine executes a command to kill the target process.

**Exit Condition**

The process is terminated on the target machine.

**Exceptions**

Fail to deliver command.

Failed to connect to the machine.

**Nonfunctional Requirements**

Process must be terminated within 10 seconds of the button being clicked.

***Get Machine Bandwidth***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The user is looking at a specific machine and need to know the download and upload speeds of the machine.

**Flow of Events**

User expands the Speed Test module of the single machine dashboard. The dashboard requests a speed test from the target machine and reports the results to the user.

**Exit Condition**

The user sees the download and upload speeds of the machine he is monitoring.

**Exceptions**

Fail to deliver speed test request.

Failed to connect to the machine.

**Nonfunctional Requirements**

None

***Chat With Machine User***

**Actors**

Admin: The network administrator inspecting the user’s machine.

User: A user is the user of the machine.

**Entry Condition**

The administrator is looking at a specific machine and needs to communicate with the person using the machine.

The user is working on his machine.

**Flow of Events**

Administrator expands the Chat module of the single machine dashboard. The dashboard reveals a chat window for the administrator to type and send a message. The administrator types a message and clicks on the send button. The dashboard sends the message to the machine. The machine opens a chat window with the administrator’s message and capability for the user to enter and send a message for the administrator. The user types and message and sends it to the administrator. The machine sends the message to the dashboard. The dashboard shows the message to the administrator through the chat module.

**Exit Condition**

The user and the administrator have exchanged messages.

**Exceptions**

Fail to deliver message.

Failed to connect to the machine.

**Nonfunctional Requirements**

None

***See Active Ports in the Machine***

**Actors**

User: A user is the administrator of a network.

**Entry Condition**

The administrator is looking at a specific machine and needs to know what ports are being used.

**Flow of Events**

The user expands the Netstats module of the dashboard. The dashboard requests a request for the ports of the machine to be scanned. The machine scans its ports and reports what ports are in use and the protocols being used. The dashboard displays the results of the scan.

**Exit Condition**

The user sees what ports are open in the machine and what protocols are being used.

**Exceptions**

Fail to deliver Netstat request..

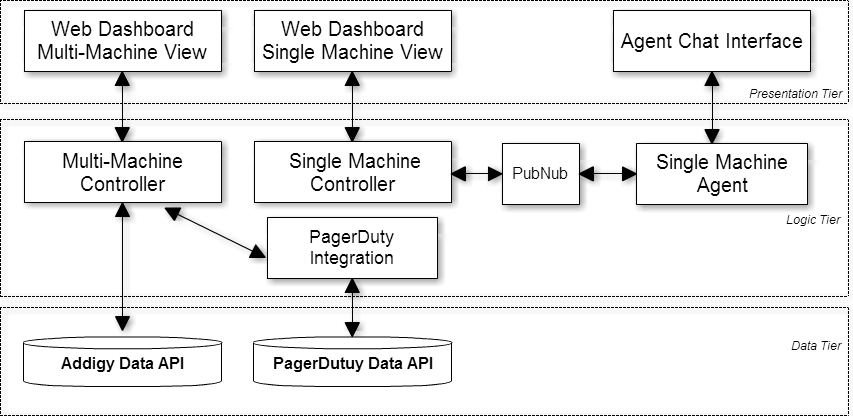
Failed to connect to the machine.

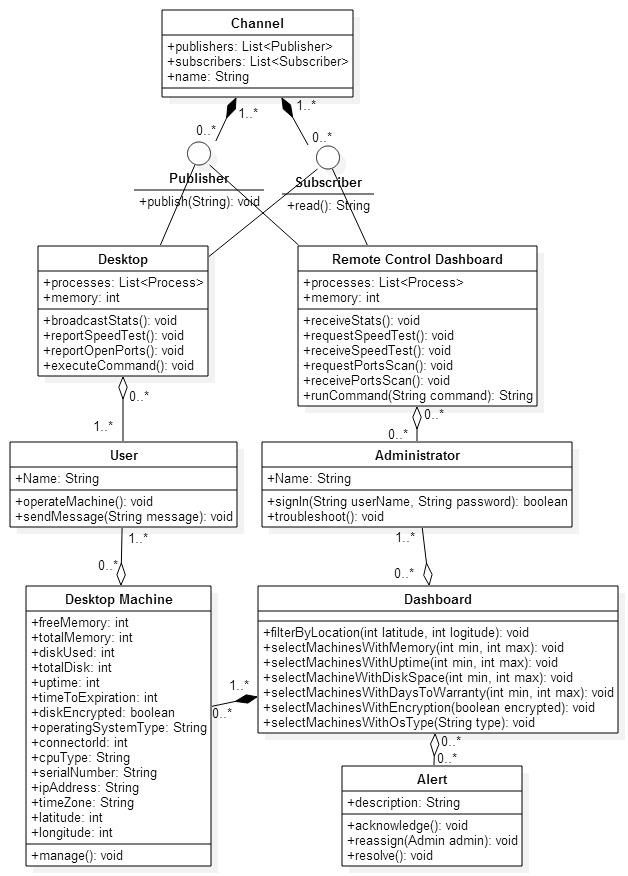
**Nonfunctional Requirements**

None

**5.3. Appendix C - Class Interface**

**System Design**



**Class Diagram**

**5.4. Appendix D - Diary of Meetings and Tasks**

**Diary of Meetings**

|  |  |
| --- | --- |
| **Meeting** | **Date** |
| Meeting 1 | 9/4/2014 |
| Meeting 2 | 9/9/2014 |
| Meeting 3 | 9/16/2014 |
| Meeting 4 | 9/23/2014 |
| Meeting 5 | 9/30/2014 |
| Meeting 6 | 10/7/2014 |
| Meeting 7 | 10/14/2014 |
| Meeting 8 | 10/21/2014 |
| Meeting 9 | 10/28/2014 |
| Meeting 10 | 11/4/2014 |
| Meeting 11 | 11/11/2014 |
| Meeting 12 | 11/18/2014 |
| Meeting 13 | 11/25/2014 |
| Meeting 14 | 12/2/2014 |
| Meeting 15 | 12/9/2014 |

**Project Schedule**

|  |  |
| --- | --- |
| **Task Name** | **Deadline** |
| Dev and Product servers are set up | 9/8/2014 |
| First Iteration: Scenario-Based Implementation - Unit Testing | 9/8/2014 |
| Second Iteration: Scenario-Based Implementation - Unit Testing | 9/22/2014 |
| Third Iteration: Scenario-Based Implementation - Unit Testing | 10/6/2014 |
| Fourth Iteration: Scenario-Based Implementation - System Testing | 10/20/2014 |
| Fifth Iteration: Scenario-Based Implementation - System Testing | 11/3/2014 |
| Sixth Iteration: Scenario-Based Implementation - System Testing | 11/17/2014 |
| Final deliverable | 12/11/2014 |
| Final posters | 12/5/2014 |
| Showcase | 12/12/2014 |