2014

Decision system for incremental upgrades of the Power Distribution System for Electric Vehicles

Syed Mohammad Sami
University of Windsor

Decision system for incremental upgrades of the Power Distribution System for Electric Vehicles

by

Syed Mohammad Sami

A Thesis

submitted to the Faculty of Graduate Studies

through the Department of Electrical and Computer Engineering

in Partial Fulfillment of the Requirements for the

Degree of Master of Applied Science at the

University of Windsor

Windsor, Ontario, Canada

Decision system for incremental upgrades of the Power Distribution System for Electric Vehicles

by

Syed Sami

APPROVED BY:

_____

Dr. Z. Kobti

School of Computer Science

_____

Dr. E. Abdel-Raheem

Department of Electrical and Computer Engineering

_____

Dr. K. Tepe, Advisor

Department of Electrical and Computer Engineering

29 April 2014

AUTHOR'S DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

The growing demand of Electric energy powered vehicles has some unwanted side effects on the current power grid system. Some of these side effects are wires not being able to sustain such high currents for prolonged periods of time, transformers not being able to sustain stable voltages [1], to sustain the added energy demands. Thus, we have to solve how and where we can provide a feasible incremental upgrade approach for the energy distribution companies. This requires us to monitor voltage and current output of various nodes in the electrical distribution system, which is able to collect data and identify patterns. Thus, a real-time monitoring and decision making system is proposed that is able to forecast future needs as well. The simulated environment created in this thesis verifies that such monitoring system can provide valuable information to distribution companies and identify problems in the grid for immediate and future upgrades.

DEDICATION

I would like to dedicate my work to my parents Dr. Syed Mohammad Saddique and Raheela Naz, without whom I would not have been able to pursue my education and realize my true potential.

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Dr. Kemal Tepe for his support and guidance. I have undoubtedly learnt a great deal from Dr. Kemel, which is evident in my work.

Moreover, I would like to thank my committee members Dr. Ziad Kobti and Dr. Esam Abdel-Raheem for their valuable input and guidance throughout this research process.

I would like to express my sincerest gratitude to my fellow lab members for their involvement and feedback as well, especially Kazi Atiqur Rahman.

Last but not least, I would like to thank my family and friends for their support in my quest for attaining my degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Statement of the Problem

With the increasing prices of gasoline and environmental concerns related to carbon emissions, car manufacturers are looking at employing electric energy for modern vehicles [2], i.e. Electric Vehicles (EVs). Currently, environmentally cautious consumers are purchasing such vehicles and governments are providing tax and other incentives to increase demand for them. As of 2014, EVs, especially, plug-in and plug-in-hybrid, do not constitute large percentage of car market, however technological improvements in speed of battery charging is expected to propel these sales significantly. This scenario will significantly challenge the electrical grid since the demand in electricity can easily exceed the transmission capacity of the grid, and create major problems for electrical distribution companies. This demand can only be met with heavy investments and upgrades in the grid. However upgrading entire grid is not economically feasible as well as can create costly inefficiencies. That is why distribution companies needs to know where the EVs are mainly be located and charged to effectively distribute their resources and gradually upgrade the infrastructure to optimize their return for the investment as well as provide healthy grid for all their customers. This thesis investigates one of the viable options for distribution companies, which is a grid monitoring sensor networks. This network will monitor the condition of the grid and identify problem areas in the grid for distribution companies to help them in their decision making process in immediate and future upgrades.

1.2 Thesis Contribution

In order to make cost effective and reliable upgrade decisions, such as when should various nodes of the grid such as wire, power grid, distribution wire, etc. be upgraded, we need to attain data such as voltage, current , timestamps readings and process the collected data reliably and effectively. Moreover, the decisions have to be made on a multitude of data points. To be able to make timely upgrade decisions the system should be able to compile the current condition information and be able to forecast future growth in the demand. In order to demonstrate the

decision process, virtual sensors were developed and deployed in a randomly generated city scenario that pushes data to the decision making and analysing system.

A complete system including the virtual sensors and associated monitoring system was designed, implemented and demonstrated in software.

An online HTML5 based graphical user interface is developed to demonstrate the current status of the grid along with forecast information.

Once a strong sense of the performance abilities are achieved, an independent decision making system is developed and a complete description of its functionality is given in Chapter 3.

1.3 Background

Electric vehicles have been around for a century now. The first electric powered passenger carriage was built between 1832 and 1839 [3]. The definition of an electric vehicle arose to be:

"An Electric Vehicle will, by definition, use an electric motor for propulsion rather than being powered by a gasoline powered motor" [3].

1.3.1 Electric cars versus Gasoline engines

Karl Benz was the first person to introduce the automobile to the society in 1885. Although, there were numerous cars designed, those were not as close to the automobile as we know today, other than Karl's. Nevertheless, the first electric motor vehicle was actually designed and implemented in 1835 by an engineer, Thomas Davenport of Brandon, Vermont. [4].

Electric cars have been around before gasoline engines. The only reason it was not a commercial success till recent, was because of major political issues regarding the oil industry and the lack of research to make electric vehicles a feasible mode of transportation [2].

1.3.1.1 Performance

Some of the major factors that relate to vehicles, and the comparison between electric vehicles and gasoline vehicles are shown in Table 1.3-1.

	Electric Vehicles	Gasoline Vehicles
Torque	Electric motors produce more torque than its counterpart.	
Pollution	Electric motors do not produce any pollutants.	Although harmful exhausts by gasoline engines are being reduced; but they still do exist.
Driving Range	The only thing holding back the electric vehicle is its driving range on a whole charge.	
Price	Electric vehicles are far more expensive than the average gasoline engines.	
Travelling costs	Electric vehicles are much cheaper to travel on than gasoline engines.	
Efficiency	Electric cars are the most efficient source of transportation as of yet.	Although gasoline powered engines are getting more efficient every day. They still are being given a run for their money by the electric vehicles.
Refueling cycle	It takes about 8 hours to charge an electric car.	Refueling a gasoline powered car is comparatively instantaneous.

Table 1.3-1 Differences between Electric Vehicles and Gasoline Vehicles

1.3.2 Existing power distribution system

The existing power distribution system comprises various nodes or units that distribute energy to houses, offices and various end nodes in the system, this is shown in Figure 1.3-1.

Electric power is generated at the power plants by converting another source of energy into electricity; then the power is transferred over to substations where it is stepped up from a couple of thousand volts to extremely high voltages for long distance transmission on the transmission grids. After that the power is brought to the power distribution grids where the power is stepped down to a few thousand volts; and is distributed over the distribution bus/wires. Smaller transformers step down the input to usable voltage, from where it is distributed to houses, offices and other structures using regular wires.

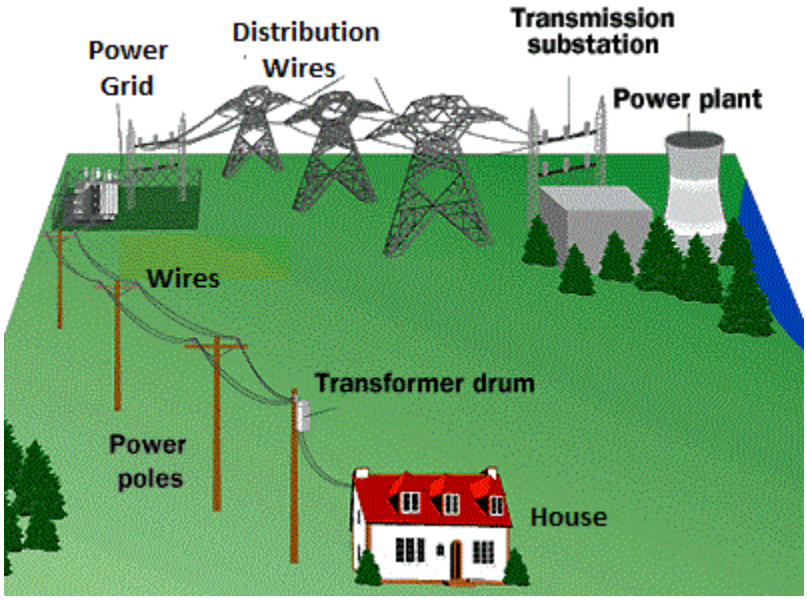

Figure 1.3-1 Typical components in a power distribution system in a city

1.4 Thesis Organization

A description of the remaining parts of this thesis is as follows: Chapter 2 will discuss some related work done around the system implementation. In Chapter 3 we will discuss the various components of the system, and the respective algorithms will be discussed. Moreover, in Chapter 4 the performance capabilities of the designed system will be analyzed and we will go in depth on the results for determining the performance of the system in several different environments, followed by Chapter 5 with the project conclusion and ideas for future work.

2 REVIEW OF THE LITERATURE

This chapter investigates relevant research, to aid in the design of various components needed for a complete monitoring system. One major point of debate for any such architecture is usually:

- Software As A Service (SAAS) on each node that is polled for data acquisition by a Business Intelligence (BI) server.
- Minimal HTTP based push capabilities on each node that send data to an independent Central Server (CS).

A significant part of the research work was done in designing the implementation.

2.1 Embedded Web Server for Wireless Sensor Networks

In [5], a research was conducted in implementing a mini-web server for the purpose of household consumers. The goal was to allow members of a household to monitor their domestic energy consumption, while being able to read integral data from the implemented device. Being a distributed design, the author had built an embedded web server as an expansion module to keep his design modular. It allows authorized users to establish two-way communication with the sensor network deployed in the house.

The author suggests that this system can be utilized for data collection and monitoring, which previously required physical presence at the research site. The author indicates that such a web server design would have limited computational, storage and energy resources available. This was one of the shortcomings we tried to overcome in this thesis. That is why a central web server concept is proposed in order to run multiple calculations repeatedly to allow for a minimum delay in publishing the results on the control panel. Moreover, to make the required decisions expected from the system; it requires information from all the sensory nodes.

In [5], the author also states that the Embedded Web Server (EWS) was primarily built to serve as a remote access and control interface for various nodes in the network; contrary to this, in our design, we have allowed the administrator of the system to have a global view of all the nodes in the system in real-time. The belief here is that, a user would not need to explore each sensor in a network individually; rather a user would like consolidated data, which we have tried to accomplish with this project. Nevertheless, if there is a need to micro-manage various nodes in the system, system can handle this as well.

The author also describes various embedded hardware, designed for deploying mini-web servers such as TargetWeb, Tmote, Webchip and a few more.

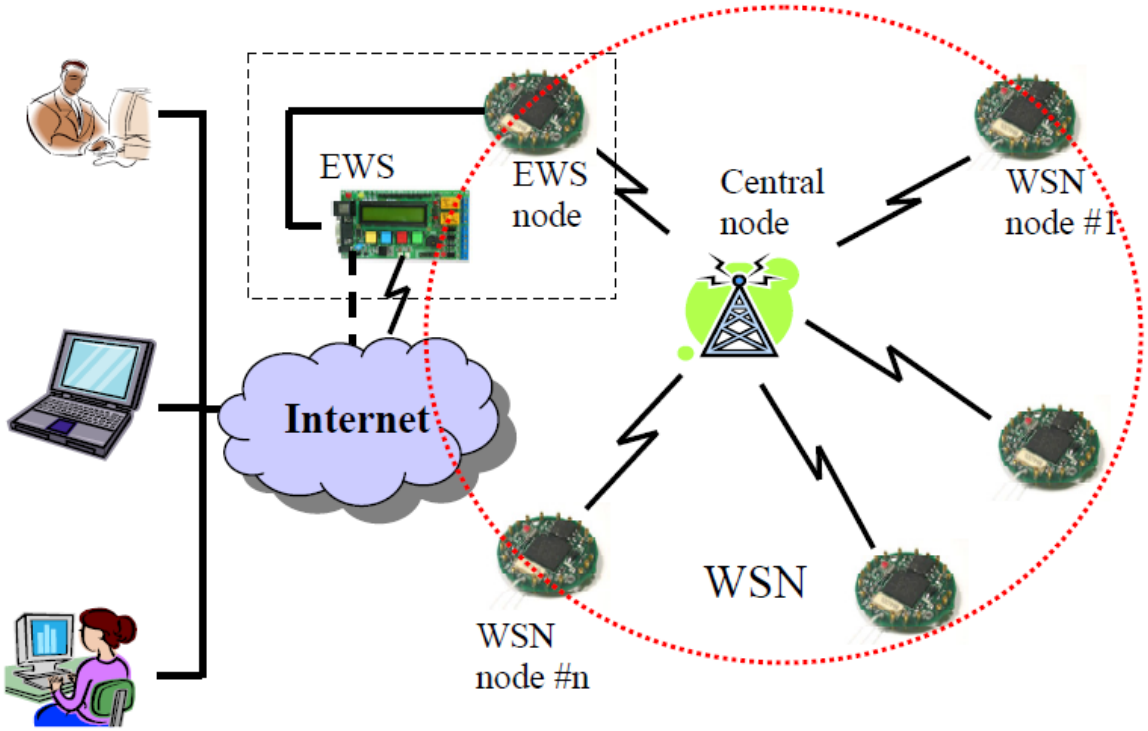

Figure 2.1-1 Implementation of Wireless Sensor Network for Data Acquisition purposes

A network topology presented in [5] is given in Figure 2.1-1. The system in this thesis has a larger undertaking than what was provided in [5], in terms of being able to monitor nodes all around a power grid. For this reason, instead of deploying our own network; we are relying on the internet; where data is gathered by the central server.

In [5], the author describes his method of communication; each sensor node after processing the data acquired by them, sends it to the central node in an assigned time slot. The problem with such a system is scalability and reliability. When the data has to hop through multiple relay-nodes to reach the central server; the latency experienced is more significant, thus causing unreliability. Such unreliability due to multi-hop network was discussed in [6].

In [5], the author also describes how forking different processes to handle various requests from the servers is important and used Boa which supports integration with Common Gateway Interface (CGI) . One important design consideration to be made here is that a server will have to handle numerous requests. If the requests are not managed properly, it may lock the database and restrict further Create, Read, Update and Delete(CRUD) operations and may not allow data sent by other sensor nodes to be logged. In our implementation, FastCGI was employed which is far more superior to CGI. To avoid table locking, we have used the innoDB engine which allows row level locking.

2.2 Service Oriented Process Monitoring and Control

A design approach, usually considered is, Service Oriented Architecture (SOA). As an example of a distributed web server design, in [7] some researchers attempted such an architecture; and stated that all current and future systems will be able to share information in a timely and open manner, enabling an enterprise-wide system of systems that will dynamically evolve based on business needs.

This thesis shares the same vision, and enables the user to add additional nodes seamlessly. As technology prospers, and more parameters, components or sensors are to be added to the system; the user can do so with minimal alterations.

2.3 Predicting energy measurements of service-enabled devices

In [8], the authors share a coherent goal with this thesis project, but using a different implementation. The authors present methodologies for mining data gathered from devices, deployed for energy monitoring purposes; i.e. web service enabled smart meters and home appliances. The authors expect to evaluate the mined data in a timely manner and feed the results

to various business intelligence tools that might provide a business advantage.

The way their implementation differs from this thesis, is that the central node polls all the nodes in the domain, and makes it viewable to the user and then processes the data in a real-time fashion; whereas we allow the sensor nodes to push the data to the central node whenever they have an update and we use the information previously pushed to generate data for upgradability in an asynchronous yet timely manner.

Moreover, the authors in the paper have suggested that if a prediction model is properly implemented, it may allow the devices to make accurate decisions based on the dynamic nature of the events. Considering their architecture for this project, the central server or Business Intelligence (BI) server has to first poll each node for the required information, process, make decisions and then send the data back to the nodes to be able to make dynamic decisions. This system will be more viable, if self-healing is being integrated into the smart grid as discussed in [9], but when it comes to monitoring purposes, it does not make sense to redistribute the BI information back to the devices.

According to [9], in the future it is expected that the existing centralized generation of energy will be more distributed, for example the households and factories will not just consume energy, but will also product energy making them Producers and Consumers (ProSumers). This is a concept that we have not considered for this thesis, but the system implemented in this thesis is readily scalable to meet this solution.

Moreover, the author states that in the future there will be many heterogeneous devices that will want to provide their information to a service for business intelligence purposes. Thus, the system described in this thesis uses the HTTP standard to communicate between devices which is readily implementable on most devices.

In [9], some common algorithms used for event prediction, sequence matching, compressions based prediction, prediction using Markov models, and episode discoveries were presented. Although, for the purpose of this thesis, these algorithms were researched; due to the lack of time and resources it has been deemed out of scope and has been added as a future enhancement to the proposed system.

In the end the author classifies various events generated by the sensory nodes as occasional events and heartbeat events.

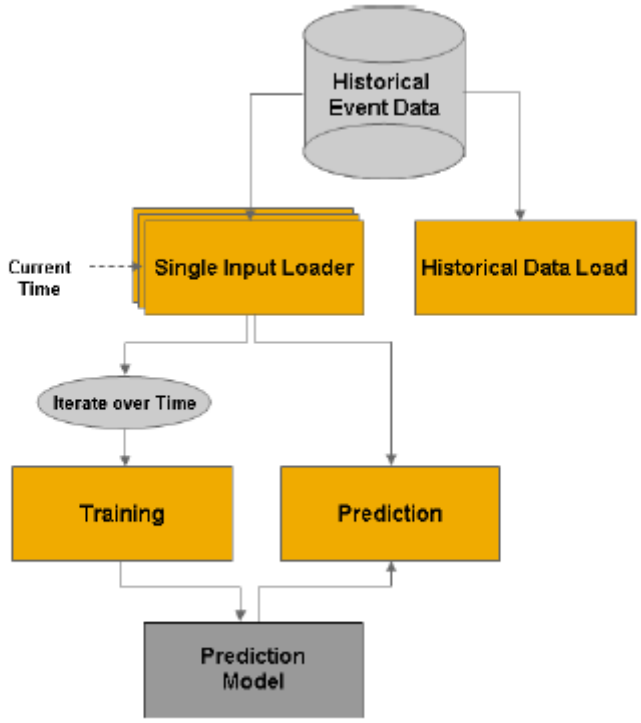

Figure 2.3-1 Prediction model overview

Event classification as described earlier have been incorporated in the algorithms described in this thesis. The system proposed by this thesis uses historical data and recent data, to predict growth and upgrade requirements in the power distribution system. Such system is illustrated in Figure 2.3-1.

2.4 Simulations of smart grid Cities

A major chunk of research done in the field of smart grids is done using simulators. In [10] the author has used a Multi Agent System (MAS) to analyze, design and build a simulator that attempts to create the dynamic behavior of a smart city. The author believes that the current power distribution system is expected to change in the near future; which makes the work done by this thesis even more vital, as it allows for efficient, timely and cost-effective upgrade decisions.

The author discusses that the best way for an enterprise application to get data from other entities in the network is using web services. In this thesis, the same concept has been employed and the various nodes in the system communicate using web services.

The author of this paper also discusses how the real-time analytics generated by their system shows the current overall consumption and generation of energy. This information, when generated by each node, is saved into a database for the server to interpret. The component of MAS system is given in Figure 2.4-1. The system described in this thesis can also be employed to use the data generated by the various MAS used in this paper by tapping into the database.

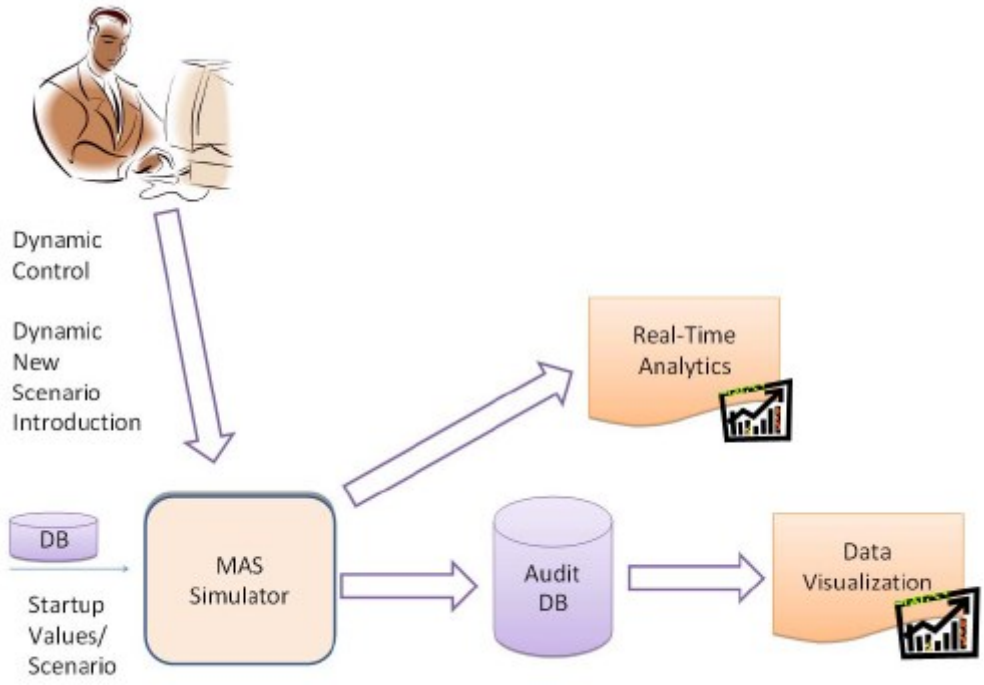

Figure 2.4-1 MAS system for simulation of various components in a city

The author further suggests that prediction algorithms should be incorporated to make the data generated above more valuable, which is the premise of this thesis.

To conclude, the author suggests that for simplicity and as a proof of concept, the implemented system will turn consumer devices on and off to allow the current power distribution system to sustain itself. In this thesis, we believe that it is not feasible to expect consumers to cut

back due to increasing energy demand; rather the current power distribution system should be upgraded to meet the future demand. At the very least the cost of using appliances at different times should be regulated to allow the consumer to estimate the need against the cost of the need; thus showing energy providers the urgency of the upgrade.

2.5 Cyber Physical Systems

Along with the above research, another major topic of interest is the ability for various sensory nodes to perform physical actions based on commands sent from a server. In [11], the author says that radical changes are expected to occur in the next years in the electricity domain and the grid itself. Re-enforcing the premise of this thesis, the author of the paper states that smart grids will heavily rely on IT technologies at several layers for monitoring and controlling. Advanced business services will take advantage of the near real-time information flows among all participants.

The author also suggests that in order to realize the SmartGrid promise, we will have to heavily rely on Cyber-Physical Systems (CPS) to monitor, share and manage information and actions on the business and the real world. In doing so, the SmartGrid network will become "100 or 1000 times larger than the Internet" [12]. This concept is also reinforced by [13]. As an example of a SmartGrid CPS the author considers the Smart Meters.

The author further asserts the topic that Business systems nowadays heavily depend on real-time and accurate information from various sensors and CSPs.

To elaborate on the issue of electric vehicles invading the current power distribution system the author states that mobility is a prominent issue in SmartGrids, i.e. we will have many mobile objects tapping into the grid at various locations in the SmartGrid network. This thesis acknowledges this fact and has considered this for the end solution.

Security and Privacy is also of utmost importance in a SmartGrid network. According to the author the "Show Stopper" in actually deploying these infrastructures to be able to monitor data from various points in the grid, is lack of security and privacy, which is well justified by [14].

2.6 Development Frameworks

In [15], it can be seen that among the top frameworks for PHP, Yii has grown a high acceptance rate due its superiority in integrating with third party tools. Some important considerations as a developer, when choosing a framework to work with are the following:

- Rapid application development cycles
- Following the latest and greatest coding standards (MVC etc.)
- Lightweight or no overhead
- Able to integrate with third party applications easily
- Object Relational Mapping (ORM)
- Large enough community
- View separation
- Database support
- Good documentation

Taking the above points into consideration we decided to work with the Yii framework as it does justice to a good stable development release [16].

2.7 Web Server

Web server is an application that sits on a computer or a Virtual Instance that listens to requests on port 80 or 443 using the Hyper Text Transfer Protocol or the Hyper Text Transfer Protocol with Secure sockets layer (HTTPS) protocol respectively; and replies to those request. It basically, translates a Universal Request Identifier (URI) that is requested using the above said protocols, to a file location; and returns the contents of the file. The file may be a static file such as a simple Hyper Text Markup Language (HTML) file or may even be a dynamic file such as a Personal Home Page (PHP) file; or may even be a program that executes and generates a response HTML such as a Web application ARchive (WAR) file. An example of a common Web Server stack is shown in Figure 2.7-1.

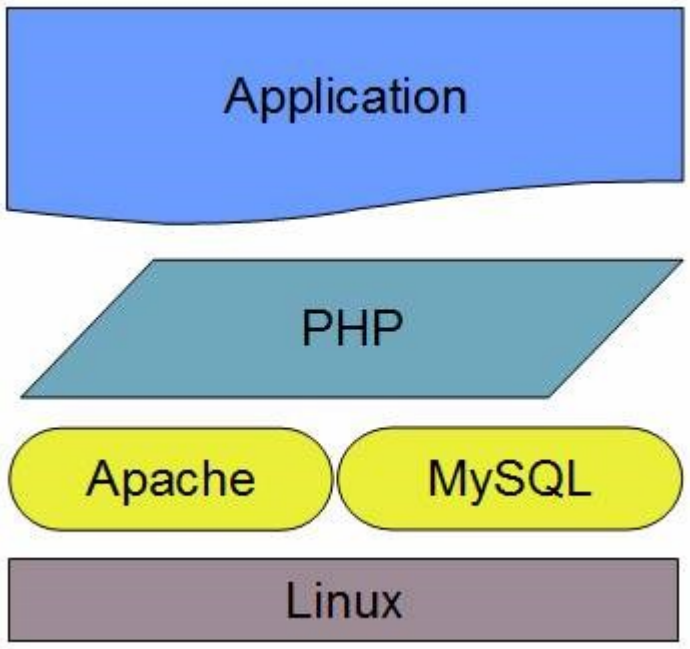

Figure 2.7-1 Various layers in a LAMP stack implementation

2.7.1 Apache Web Server

Apache Web Server is a popular web server as stated in [17]. Moreover, it is very popular among the developer community for being an open source free alternative to an enterprise level web server.

Apache Web Server generally comes as an integral part of the Linux, Apache web server, MySQL and PHP (LAMP) stack. It allows a user to create multiple Virtual Hosts (Vhosts) and readily integrates into PHP to provide dynamic web pages for each Vhost.

2.7.2 Apache Tomcat Server

Apache Tomcat Server is a part of a Java setup that allows developers to create Java Server Pages (JSP) servlets to provide static or dynamic web pages that are based on Java. Similar to apache web server, it is an open source alternative for developers to use in place of glass fish and Java Boss (JBoss), which are a part of the Java 2 Enterprise Edition (J2EE) stack [18].

Apache Tomcat readily integrates with Java applications and can deploy multiple WAR file instances. It supports resource finding and various database integration drivers through the Java DataBase Connector (JDBC) driver.

2.8 Server-side language

The server-side language in assistance with the web server dynamically handles requests and allows the server to preprocess given requests before serving them to the requesting client. The following are some options for this:

2.8.1 PHP

PHP is a popular language among new developers and is one of the most popular choices for website development in general [19]. It is deemed to be a weakly typed programming language, which actually is its strength as well. Weakly typed language means that it is a very flexible language, and it relies on "common sense" in how a task is completed. Explicit definition of function return types and input types are not required. This characteristic of PHP translates into fast development time; which is ideal for research projects.

PHP has no costs associated to it, in term of plugins, development environments, platforms, etc. It is also available on virtually any platform and is used for large scale as well as small scale applications.

2.8.2 Java

Java as opposed to PHP and RoR, is a compiled language, which results in a faster execution time, but a longer build time. In many production environments this is feasible; but on a development machine long build times and having to compile code and deploy it every time a change is made can increase development and QA cycles quite significantly.

Moreover, Java is deemed a strongly typed language, which means it requires explicit statements of intent to function and that is backed by a compiler. This means, that the language has a strict expectations from its programmers who have to explicitly declare the types and size of various inputs and outputs for functions and variables.

2.9 Database

The database is the basic building block of any scalable and useable web service that wants to store data.

Originally databases were flat, which meant that the information was stored in one long text file in either a Comma Separated Value (CSV) format or a tab delimited files. It was difficult to search or retrieve data efficiently through this form of database, without actually traversing the entire file sequentially.

This quickly grew as a tedious task and formed the basis for modern database systems. The following are some of the modern options for databases:

2.9.1 Relational Databases

Relational databases allow a user to easily query information and also allows one to sort data in the database with a simple query. Relational databases store data in an object called a Table which is composed of Columns or Fields and Rows or Records.

The claim to fame of Relational databases is the way the information is organized on the lower end and can be used to relate tables together. The “relational” part of the name comes from the fact that the records are mathematically related to each other. A typical database would have multiple tables with multiple records which when “related” to each other provide meaningful data.

Relational Databases are communicated with using a special language called Structured Query Language (SQL); which is the foundation for most of the database applications available today.

2.9.1.1 MySQL

MySQL being an integral part of the LAMP stack stands upright is one of the most favorable database servers by the research and development community [17]. MySQL for its price and performance, when applications leverage architecture; is very well known. It has a big resource community and even strong integration manuals/plugins with the various programming languages. MySQL excels from other RDBMS when high speed reads are required. Although it lacks in database features compared to SQL and Oracle, but generally speaking, middleware products by third party companies develop those features in their applications and don't require them in the Database engine.

2.9.1.2 SQL

SQL server also offers a free edition similar to MySQL called SQL Express Edition. One noticeable advantage of SQL server is its support for scalability. However, SQL Express Edition is limited to a database size of 4GB, but does allow a developer to promote the server to an Enterprise Edition when required. SQL server unlike MySQL has an elegant GUI for all its management and support. It works closely with other Microsoft products and is in fact usually the option to go with if people are using the rest of the Microsoft Suite, i.e. .NET stack as it readily integrates into it. A contrary point that Microsoft SQL server has is that it can only be installed on a Microsoft computer and cannot be ported to a different Operating System (OS).

2.9.1.3 Oracle

Oracle is known for having an immense enterprise hold. It is one of the most stable and scalable enterprise ready Database System out there. Oracle has tons of features from XML, user-defined types and numerous database management tools. It has an immense business suite that it leverages which includes Oracle Business Suite, Siebel, Oracle Fusion Middleware, and many more.

2.10 Summary

An understanding of some of the system components needed for a good central server design and their current progress in research has been established. We have also established an understanding of the various methodologies and implementations used to deploy a reliable sensory network.

Additionally, an appropriate technology must be selected for a good web server design. The fact that many researchers are working with these types of frameworks and languages in alternative applications demonstrates their usefulness.

A good understanding of the web server application, database and the other parts of a central server are vital to a stable and durable web server which will be discussed ahead.

3 METHODOLOGY

3.1 Proposed monitoring algorithm

Now that we have our background understanding related to the subject, we can dive in to currently how our power systems are organized and what we have implemented to accompany the current solution.

For the sake of a proof of concept, we have simplified our solution and made a minimum viable product (MVP). In this system we have only considered the following nodes:

1. City
2. Power Plants
3. Distribution Wires
4. Power Grids
5. Wires
6. Houses
7. Offices
8. Charging Stations

These nodes are just created for demo purposes, in a real life scenario, this can be simplified or one can add more nodes to this system as they desire. Every node is associated to a sensor that reads the data going into the node. A visual depiction of the assumed system is shown in Figure 3.1-1.

For the purposes of the proof of concept, we are randomly generating values from the sensor, as the sensor implementation is out of the scope of this thesis.

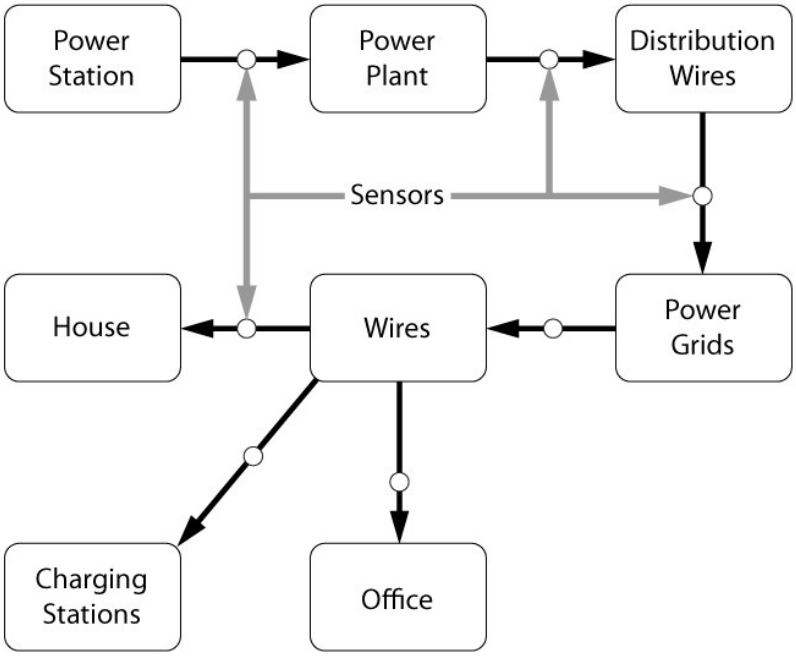

Figure 3.1-1 Overview of Power Distribution Grid as assumed in this thesis

3.2 Implementation of the proposed monitoring system

3.2.1 Database Schema

The most vital part of this project was designing the database and how to normalize it. Coming from a database engineering background, this tasks wasn't too hard.

Some considerations for the database design were as follows:

- Data has to be normalized so that it is scalable.
- There has to be a log of all previous sensor data for forecasting purposes and to be able to generate daily information.
- Sensors table should have data independent of the central server information

- There should be ease of adding new nodes in the system
- Parent-child relationship should be maintained.

A UML diagram of the database schema is shown in Figure 3.2-1.

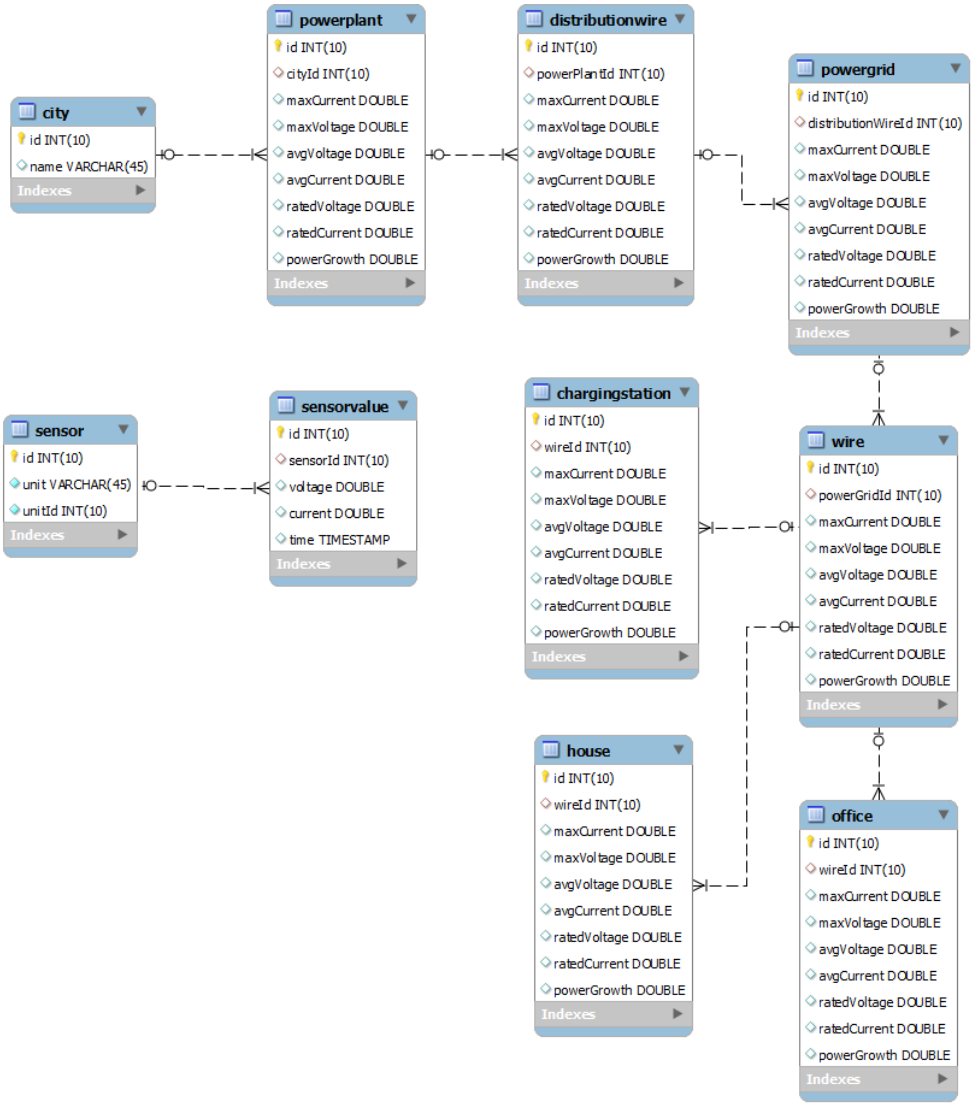

Figure 3.2-1 UML diagram illustrating the database structure of the implemented system and relationship among tables

3.2.1.1 Discussion of Database

To maintain the inter table relationship, we have established foreign key relationships between the various tables. Let's take an example of the relationship between the city and power plant tables. As can be seen the power plant table has a key cityId which actually refers to the unique identifier in the city table "id". This maintains the parent-child relationship for processing purposes.

Another thing, of great importance is the sensorValue table. This table stores all the sensor values as pushed by the sensors. It has a sensorId which refers to which sensor sent the information, and then the voltage and current sent by the senor and what time the data was sent.

The sensor table has all the sensors listed. When we query all the data for each sensor we can generate various values such as average voltage on a daily basis and average power consumption, power growth etc. and assign it to the respective node as can be seen in the database diagram in Figure 3.2-1.

The processed data is then used by the central server for forecasting information.

To establish the relationship between the sensors and the various nodes of the power distribution system, the unit and unitId values are used. Unit refers to the table the sensor refers, i.e. "house", "chargingStation" etc. and the unitId refers to the id of the node in that respective table.

3.2.2 Virtual Sensors

As discussed earlier, to help with the project focus and to make the debugging task easier, we designed virtual sensors that can generate random data.

3.2.2.1 Limitations

To help with the debugging process, we created virtual sensors to generate random data. The flaw in the virtual sensors generating data is that, the data shown in the city real-time viewer

is never valid, and does not follow the various laws of electronics, as it would make the project to complex.

Thus, the random data was collected from the sensors as is and decisions were made based on it. So the following issues existed:

- Power consumed by children does not add up to power supplied by parent
- Power growth rate is random
- Current fluctuate randomly
- And many such other random behaviors may be noticed

Using real sensors to input data into the system would eliminate the above issues.

3.2.2.2 Separation between Virtual Sensor and Central Server

In a real life environment, the sensors and the central server would not share the same code base; i.e. they will be two independent systems. To mimic this behavior in this project, I did the same and had two different code bases and limited the sharing of information between them to only such information which a sensor would have in a real life scenario. A hierarchical structure of the how the sensors work with the central system is shown in Figure 3.2-2.

One limitation is that to maintain a one to one relation between sensors audited by the central server and the sensors deployed; we have to assign each deployed sensor with a unique identifier and match that to the sensors the central server recognizes. For this reason, the best methodology that can be brought forth is that each sensor should generate a hash key based on its current geographic location (they have to have GPS sensors on them to be able to know where they are deployed, which is usually unique). This will allow the central server to uniquely identify each sensor in the current deployment.

The algorithm for these virtual sensors is discussed below:

3.2.2.3 Algorithm for virtual Sensors

- Generate a random value to correspond to voltage
- Generate a random value to correspond to current
- Knowing the central server DNS information, send a GET request to the central server with the generated data and a timestamp of the time of generation.
- Loop through the above steps.

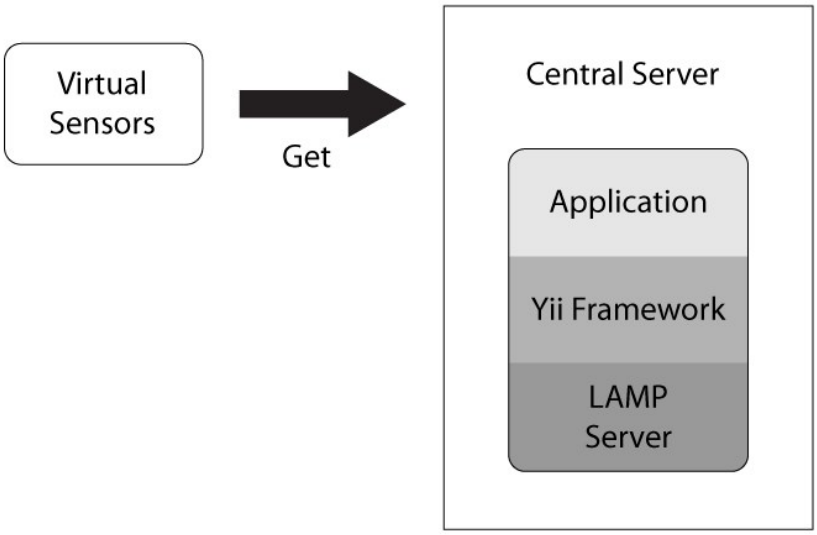

Figure 3.2-2 Communication between a Sensor node and Central Web Server

3.2.2.4 Gathering data from sensors

The first step in actually making any decisions, is gathering the raw data. For this purpose we have made an externally facing service called “Sensor Update”. Each sensor is supposed to have internet accessibility and queries the following URI to pass information to the central server:

http://mastersthesis.local/father/Father/index.php?r=site/sensorupdate?id={id}&v={volta

ge}&i={current}&time={YYYY-MM-DD}

The central server takes the values from the above query and saves it in the database in the "SensorValue" table. The id of the sensor and the other values are all saved in that table.

3.2.2.5 Processing RAW data

From the saved values above the maximum readings, average readings, and growth rates are calculated for each sensor. The processed data and the growth rate is then used to calculate after how long the specific element needs to be upgrade (based on the rated maximum values).

3.2.3 City Generator

In a real case scenario the city will be preloaded into the system using a file/excel sheet with all the nodes listed. The only reason the city generator was created in this project was so that we can test out the system properly.

The city generator will deploy random elements with a random number of children. A preview screen for the city generator is shown in Figure 3.2-3.

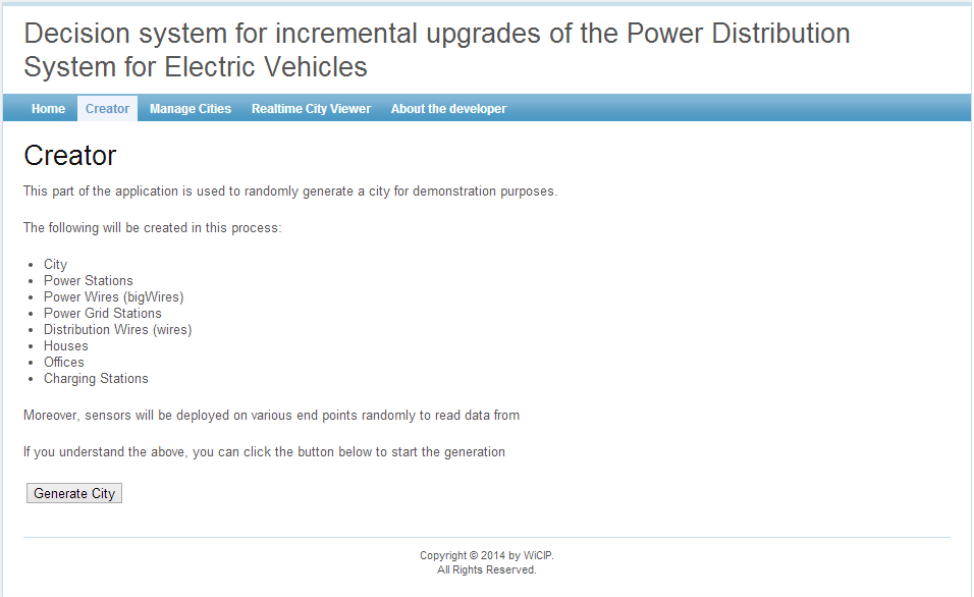

Figure 3.2-3 City generator home screen

3.2.3.1 City Generator Algorithm

To be able to generate a city that is close to reality, some information needs to be gathered first. Number of charging stations, houses, offices, power plants, etc. need to be decided as a function of a randomly generated population.

On doing a general survey of 20 houses we found that the average number of people in a house was anywhere from 4 to 6. Student houses usually had 6 people, whereas family houses usually had 4 people. Taking the average, we assumed a value of 5 people per household in a generic city.

Considering a whole commercial building as an office, taking a general survey of some of the medium sized and small sized businesses, we found out that most medium scale businesses had anywhere from 200 to 400 people. Whereas small businesses had 10 to 50 people. Taking a value in between the two we assumed the value 100 as it was most suited to a small generated city and as close to reality for larger cities.

We assumed a random value for chargingStations as we did not want to incorporate the growth in this sector. A randomly generated value for a proof of concept was sufficient.

The algorithm is given below:

```
assign population a random value between 1 and 5000
assign houses = population / 5
assign offices = population / 100
assign chargingStations a random value between 1 and (population / 1000)
create a city object
create a new powerPlant object
make the powerPlant a child of the city

for (i = 0; i <= (random value (1, (population / 500)); i++) {
    create a new powerWire
    make the powerWire a child of the powerPlant

    for (i=0; i<= (random value (1, (population / 1000)); i++) {
        create a new powerGrid
        make the powerGrid a child of the powerWire
            for (i=0; i<= (random value (1, (population / 2000)); i++) {
                create a new distributionWire
                make the distributionWire a child of the powerGrid

                for (i=0; i<= (random value (1, (houses)); i++) {
                    create a new house
                    make the house a child of the distributionWire
                }

                for (i=0; i<= (random value (1, (offices)); i++) {
                    create a new office
                    make the office a child of the distributionWire
                }

                for (i=0; i<= (random value (1, (charginStations)); i++) {
                    create a new chargingStation
                    make the chargingStation a child of the distributionWire
                }
            }
        }
```

Once the city is deployed the administrator has the option to deploy sensors; for without the sensors we would not be able to test the system. The algorithm is as follows:

3.2.3.2 Sensor deployment Algorithm

The sensor deployment script basically makes sure there are sensors deployed on each node in the grid. If a node does not have a sensor assigned to it, a sensor is created for it. The sensors table is then used to create virtual sensors for that node and to store historical data related to each sensor node.

The algorithm is given below:

```
Initialize nodes as an empty array
Merge all house nodes in database with nodes array
Merge all office nodes in database with nodes array
Merge all chargingStation nodes in database with nodes array
Merge all distributionWire nodes in database with nodes array
Merge all powerGrid nodes in database with nodes array
Merge all powerWire nodes in database with nodes array
Merge all powerPlant nodes in database with nodes array
Merge all city nodes in database with nodes array

foreach (nodes as node) {
    check in sensors table if there is a sensor for this nodeId and nodeType
    if exits then continue
    else {
        create a new sensor with this nodeId and nodeType
        store sensor in database
    }
}
```

3.2.3.3 Sample generated request:

```
GET
http://mastersthesis.local/Father/Update/index.php?r=site/sensorUpdate&v=99.1&i=76.5&t=2014-01-01
HTTP/1.1
Host: mastersthesis.local
Connection: keep-alive
Accept: text/html, */*; q=0.01
```

X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
Referer: http://unifycontacts.local/profile/editContact?contactId=12611
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cookie:
7e2f892cbe482e85202320d50df83653=d1f5aef37bbaca4d2be3c2e529ed511de258be3ba%3A4%3A%7Bi%3A0%3Bs%3A1%3A%224%22%3Bi%3A1%3Bs%3A14%3A%22samithebadshah%22%3Bi%3A2%3Bi%3A2592000%3Bi%3A3%3Ba%3A0%3A%7B%7D%7D;
PHPSESSID=hefpqih9fv8il1lvuqj588r5e5

The cookie in the above generated request is irrelevant and is generated due to development scenarios. In a real life scenario the generated request would be anywhere between 500 to 600 bytes based on our experimentation.

3.2.4 City Real-time Viewer

The real-time city viewer shows live data from each sensor and updates the page every 10 seconds. Moreover, it shows some general statistics and forecasting information as well, as shown in Figure 3.2-4.

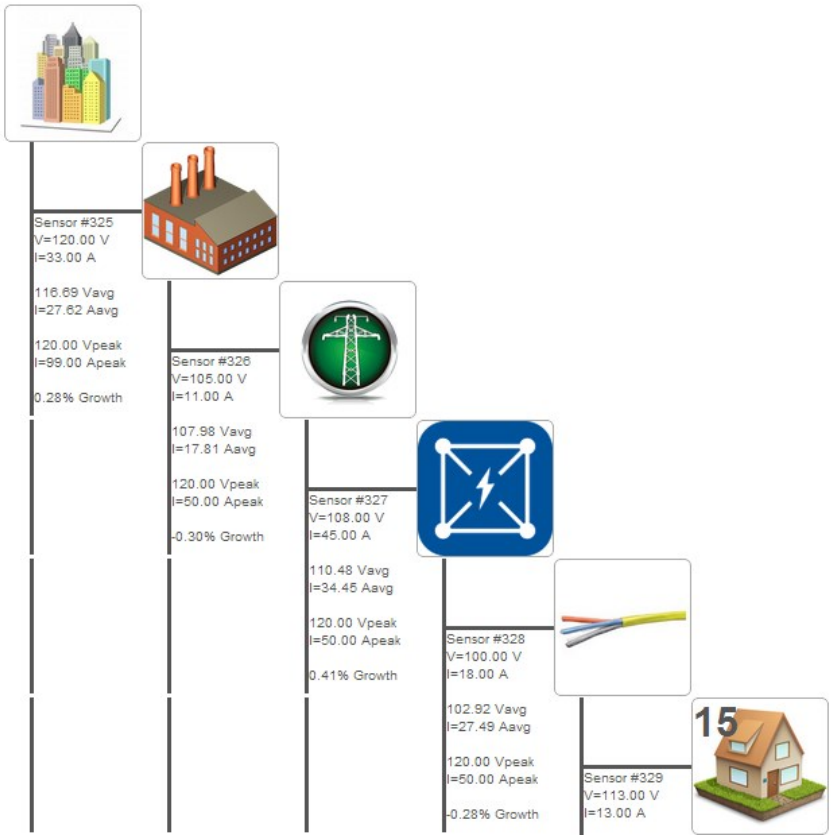

Figure 3.2-4 Real-time overview of the city with sensor values

Note: Please don't mind the values as they are randomly generated by the sensors. Actual values from a proper deployment of sensors will show more probable values.

The data shown, are as follows:

- Voltage read before sensor
- Current flowing through node
- Average Voltage reading
- Average Current reading
- Peak Voltage for the day
- Peak Current for the day
- Power growth on a daily basis

From this gathered data, a technical savvy engineer can see that if any of the following criteria is met, the node should be upgraded:

- If the peak voltage is greater than the maximum voltage handled by the node.
- If the peak current is greater than the maximum tolerable current handled by the node.
- Using the power growth, one can estimate when the node needs to be changed.

If any of the above criteria are met, the node should be upgraded and monitored for further growth.

3.3 Cases

As one may have already understood, the parameters we have gathered above are essential in defining a system completely. Some things we can estimate are as follows:

3.3.1 Inefficient Node

As we know for any node in the system the following should be true:

$$P_{in} = P_{out} + P_{dissipated} \tag{3.3-1}$$

Where P_{in} is input power, P_{out} is output power and $P_{dissipated}$ is power dissipated due to friction and other unwanted circumstances.

In an ideal case the following should be true:

$$P_{in} = P_{out} \tag{3.3-2}$$

Since, the dissipated power in the above equation is not desirable, we can come up with a term efficiency (e) that should be maximized as follows:

$$e = P_{out} + P_{in} \tag{3.3-3}$$

This gives us a percentage value of the Input Power that is dissipated in the node. In the system above, since all the sensors are placed before the nodes we have, the input power going to the system in the form (V_{avg} * I_{avg}), but we do not have the output power. The output power can be calculated as follows:

$$P_{out} = \sum_{i=0}^{i=m} P_{in\ of\ child-i} \tag{3.3-4}$$

Where m is the number of children.

Thus, our equation for efficiency in the system becomes:

$$e = \frac{\sum_{i=0}^{i=m} P_{in\ of\ child-i}}{P_{in\ of\ current\ node}} \tag{3.3-5}$$

Using the equation above, we can calculate the efficiency of each node in the system and of the overall city as well. We can calculate the efficiency of our power distribution system by taking the city as the node and the consumption of the end nodes (houses, charging stations, offices) as follows:

$$e = \frac{\sum_{i=0}^{i=m} P_{in\ of\ child-i}}{P_{generated}} \tag{3.3-6}$$

We can thus, monitor and make sure each node in the system does not inefficient as it will defeat the purpose of us upgrading the system to begin with.

3.3.2 Faulty nodes

Similarly, we can use our real-time city viewer to show us any nodes that are not operating properly. We can basically check and see which nodes have an input power of zero or very less than nominal and inspect the parent node for faults.

Moreover, we can generate alerts when there is a faulty node and send the work force to resolve the situation.

3.3.3 Days to upgrade

We can similarly see which node needs to be upgraded and when it should be upgraded. Each node in the real-time city viewer is associated with a power growth rate calculated on a day by day basis. The power growth (g) is calculated as follows for any specific "day":

$$\%g = P_{avg}(x)/P_{avg}(x-1) \tag{3.3-7}$$

The average power for the day is calculated as follows:

$$P_{avg}(x) = V_{avg}(x) * I_{avg}(x) \tag{3.3-8}$$

Where x is the day the required value is for.

After the percentage growth rate is calculated, we can estimate the time when upgrading the node will be inevitable by forecasting based on the current growth rate using the following equation:

$$P(x+1) = P(x) * (1+g) \tag{3.3-9}$$

$$P(x+2) = \big(P(x) * (1+g)\big) * (1+g) = P(x) * (1+g)^2 \quad (3.3\text{-}10)$$

$$P(x+n) = P(x) * (1+g)^n \quad (3.3\text{-}11)$$

Since we know the node has to be upgraded before the maximum rated power is exceeded:

$$P_{max} = P(x) * (1+g)^n \quad (3.3\text{-}12)$$

Solving for n we have:

$$\log P_{max} = \log(P(x) * (1+g)^n) \quad (3.3\text{-}13)$$

$$\log P_{max} = \log P(x) + n \log(1+g) \quad (3.3\text{-}14)$$

$$n = (\log P_{max} - \log P(x)) / \log(1+g) \quad (3.3\text{-}15)$$

$$n = \log(P_{max}/P(x)) / \log(1+g) \quad (3.3\text{-}16)$$

Using the current average power consumption (P_t), the growth rate and the maximum rated power for the node (P_{max}), we can calculate using Equation (3.3-16) how many days ahead of time we have to upgrade the above mentioned node.

3.3.4 Time scheduling

We can also explore the idea, whether time scheduling of various nodes will help in the sustainability of the current power distribution system. For example, the following steps could be taken to manage time scheduling of power consumption:

- Raise per unit prices of electricity within specific hours for various nodes
- Lower per unit prices of electricity within specific hours for various nodes

To know if the above schemes will help the power distribution system sustain, the following must hold true:

$$P_{avg} < P_{max} \tag{3.3-17}$$

If the above equation holds for the system then we can explore the idea of time scheduling. The need for time scheduling can be assumed if the following equation holds true:

$$P_{peak} > P_{max} \tag{3.3-18}$$

We can basically see which nodes consume more power within the specified time when the power consumption had reached its peak and apply regulations to help resolve the situation.

4 FINDINGS

To demonstrate the goal of this thesis, we decided to generate a city that we named "Windsor", and simulated the city shown in Figure 3.2-4.

The system was evaluated by running various scenarios, and seeing if the system is able to point out the state of various nodes in the city. Moreover, we would like to evaluate and see if the system would be able to recommend when to make updates to various nodes in the city.

To execute the various instances, values of the various sensors were hardcoded, as to mimic real life scenarios. The details of each scenario and the results are shown below:

4.1 Instance 1 (Happy Path)

For the happy path, we set the values of each sensor to more realistic life like values under normal conditions. A little inefficiencies have been assumed in the system to mimic actual power loss factors.

In Figure 4.1-1, it can be seen that all the nodes are not expecting massive growth in the power consumption. There are no urgent upgrades required and the nodes are sustainable for a long time. Thus, the city is "Happy"

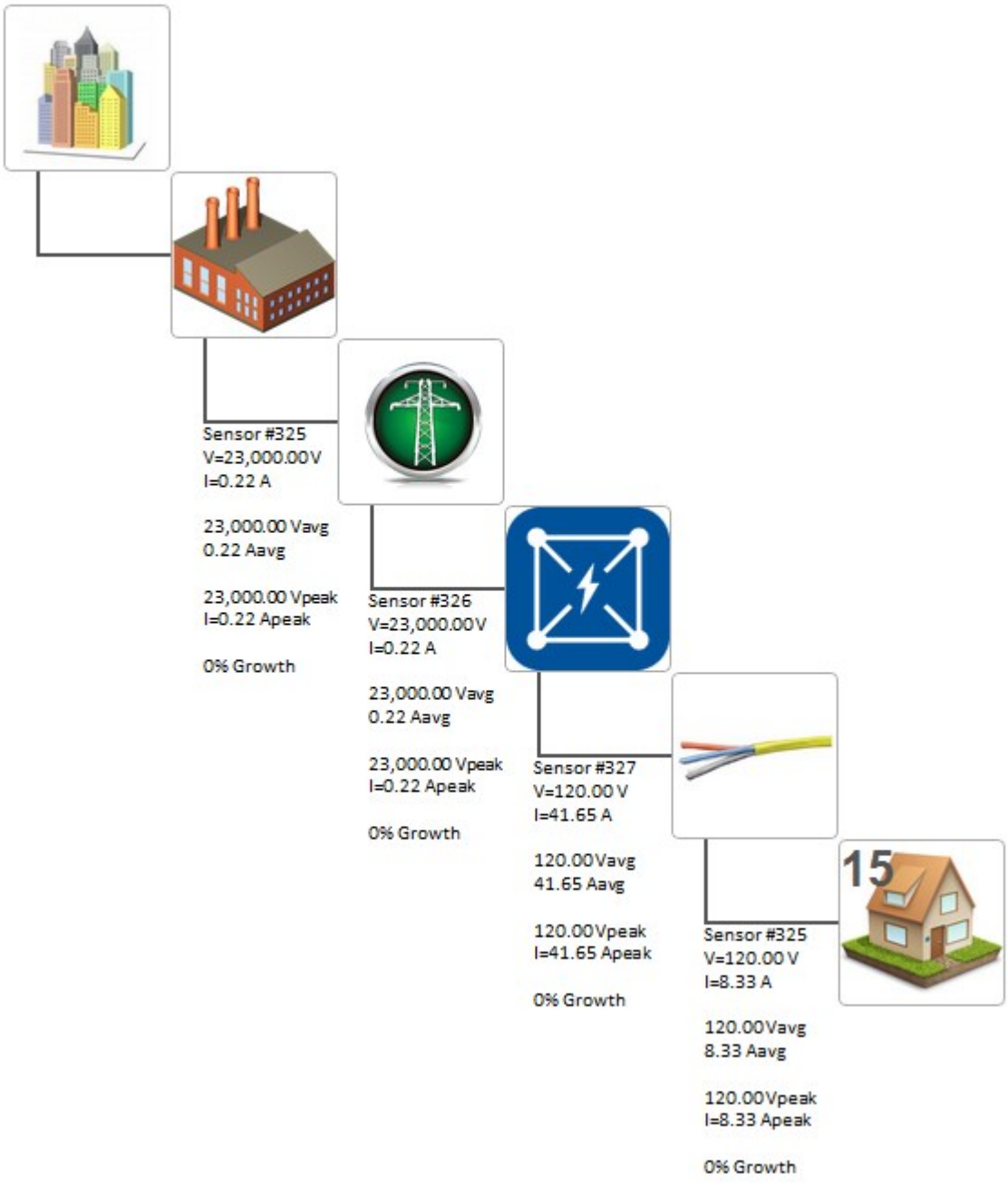

Figure 4.1-1 City on a happy path (i.e. no upgrades required)

4.2 Instance 2 (Inefficient and to be upgraded nodes)

By setting different threshold values; and changing the values read by the sensors, we explore if the system can pick out candidates for upgrades.

4.2.1 Threshold set to 90%:

Setting the threshold to 90%, we try a different set of values for sensor #326, pertaining to the distribution wires. Each of the instance has been shown below.

First setting the value of current to 0.21 A, i.e. putting distribution wires at a 95% efficiency we get values as shown in Figure 4.2-1.

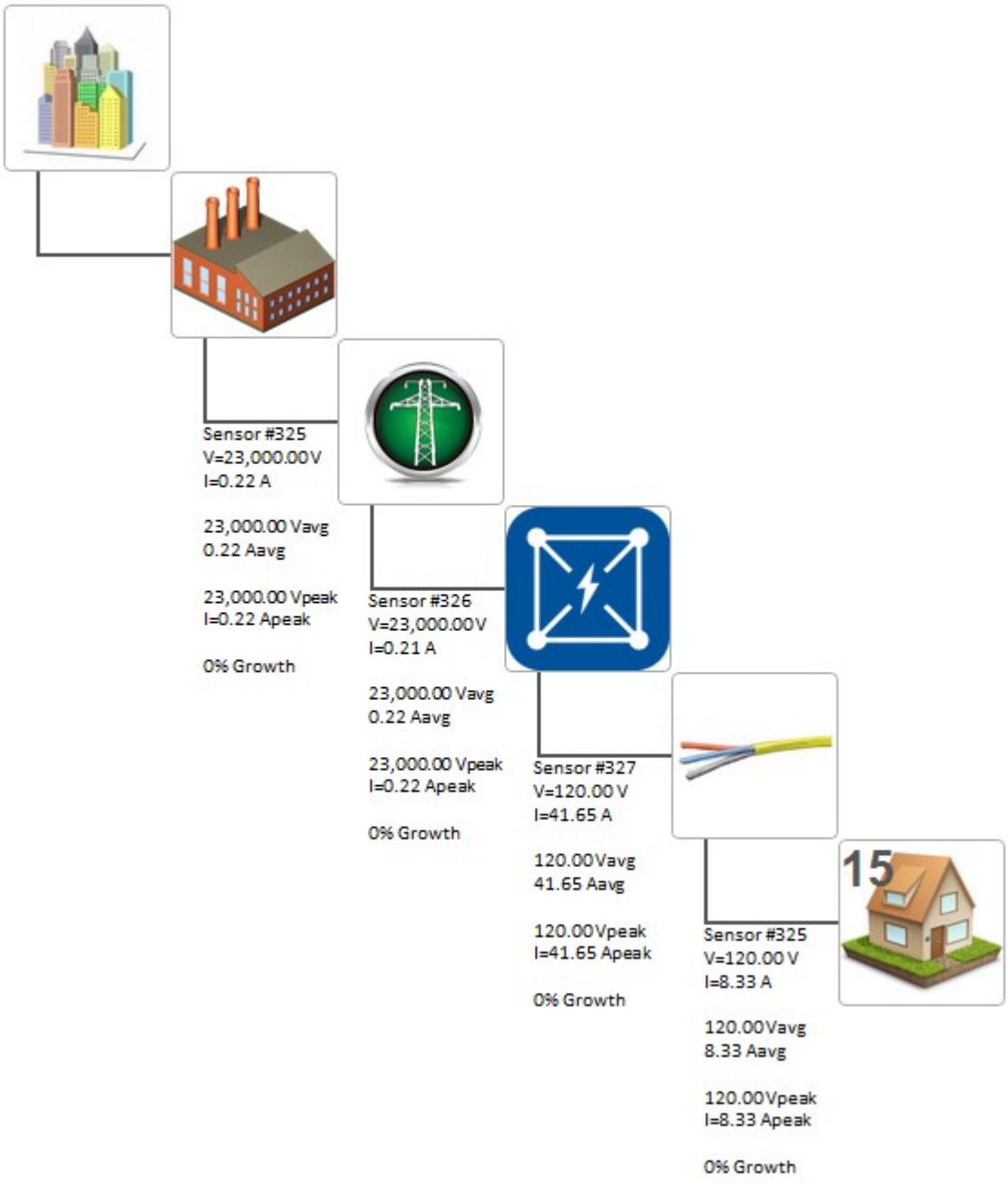

Figure 4.2-1 Inefficient nodes pointed out by the system

Now setting the value of current to 0.19 A, i.e. putting the efficiency down to 86.4%, we get what is shown in Figure 4.2-2.

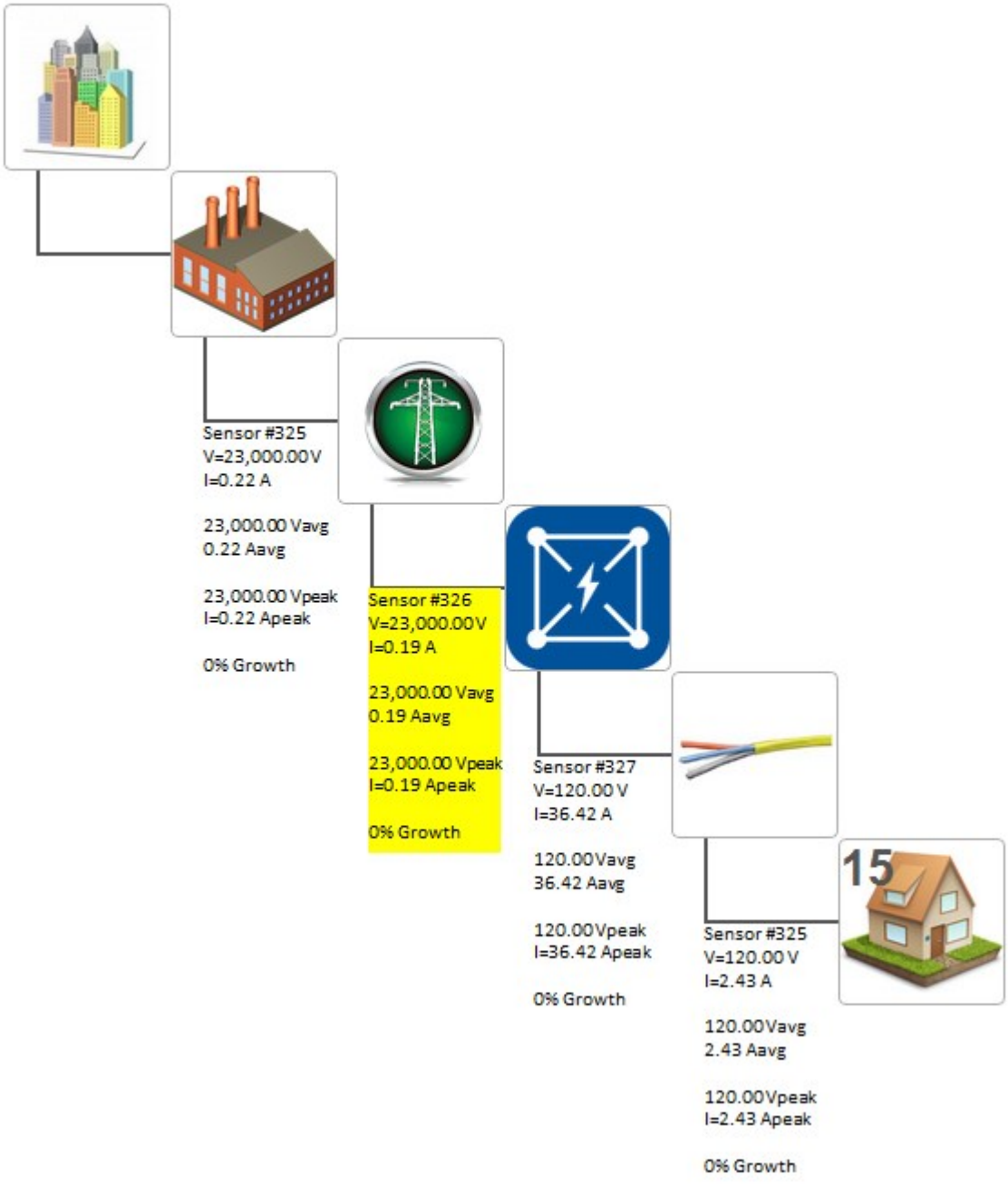

Figure 4.2-2 Inefficient nodes pointed out by the system

4.2.2 Threshold set to 60%:

First setting the value of current to 0.132 A, i.e. putting distribution wires at a 60% efficiency we get as shown in Figure 4.2-3.

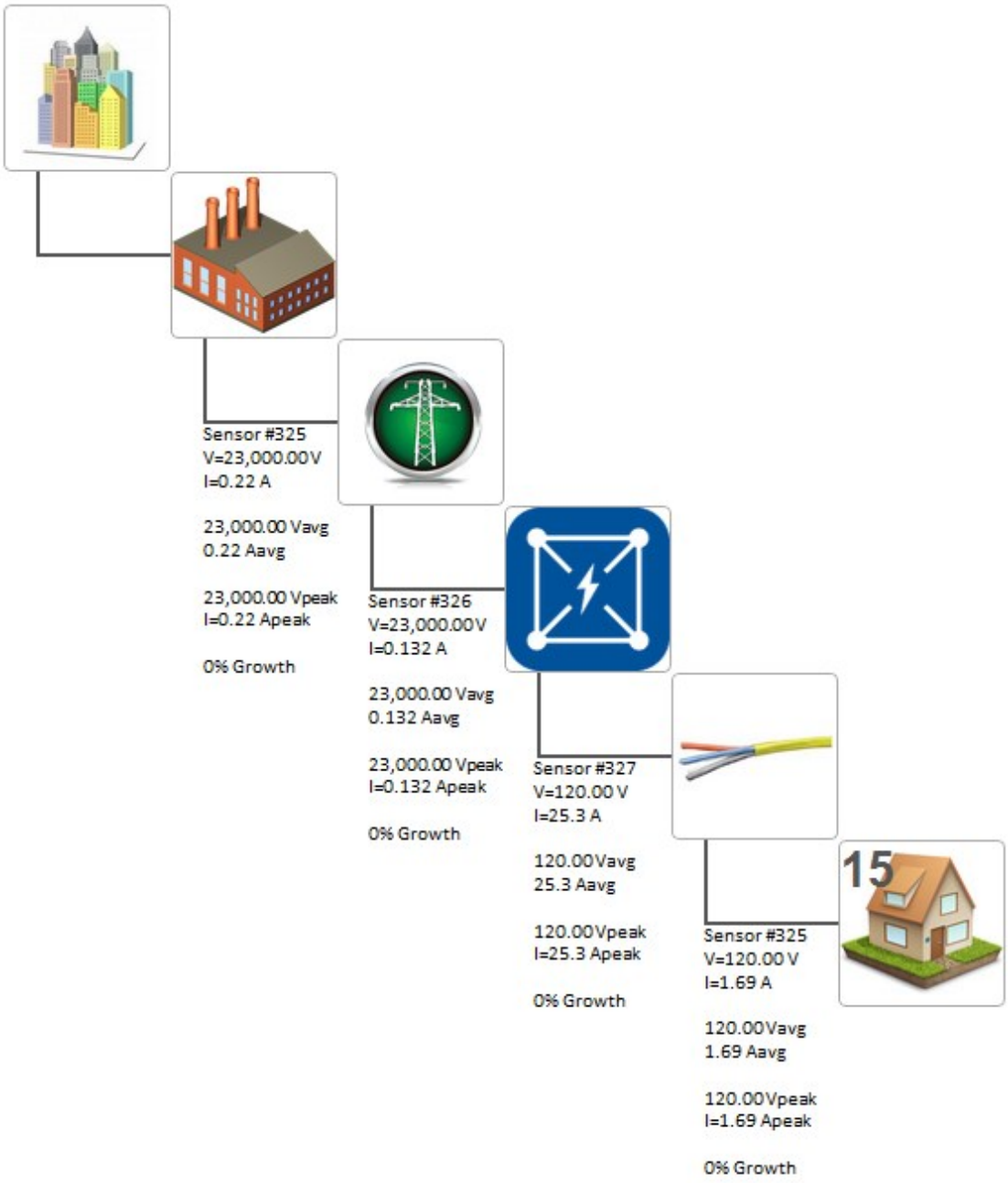

Figure 4.2-3 Inefficient nodes pointed out by the system

Now setting the value of current to 0.11 A, i.e. putting the efficiency down to 50%, the results are shown in Figure 4.2-4.

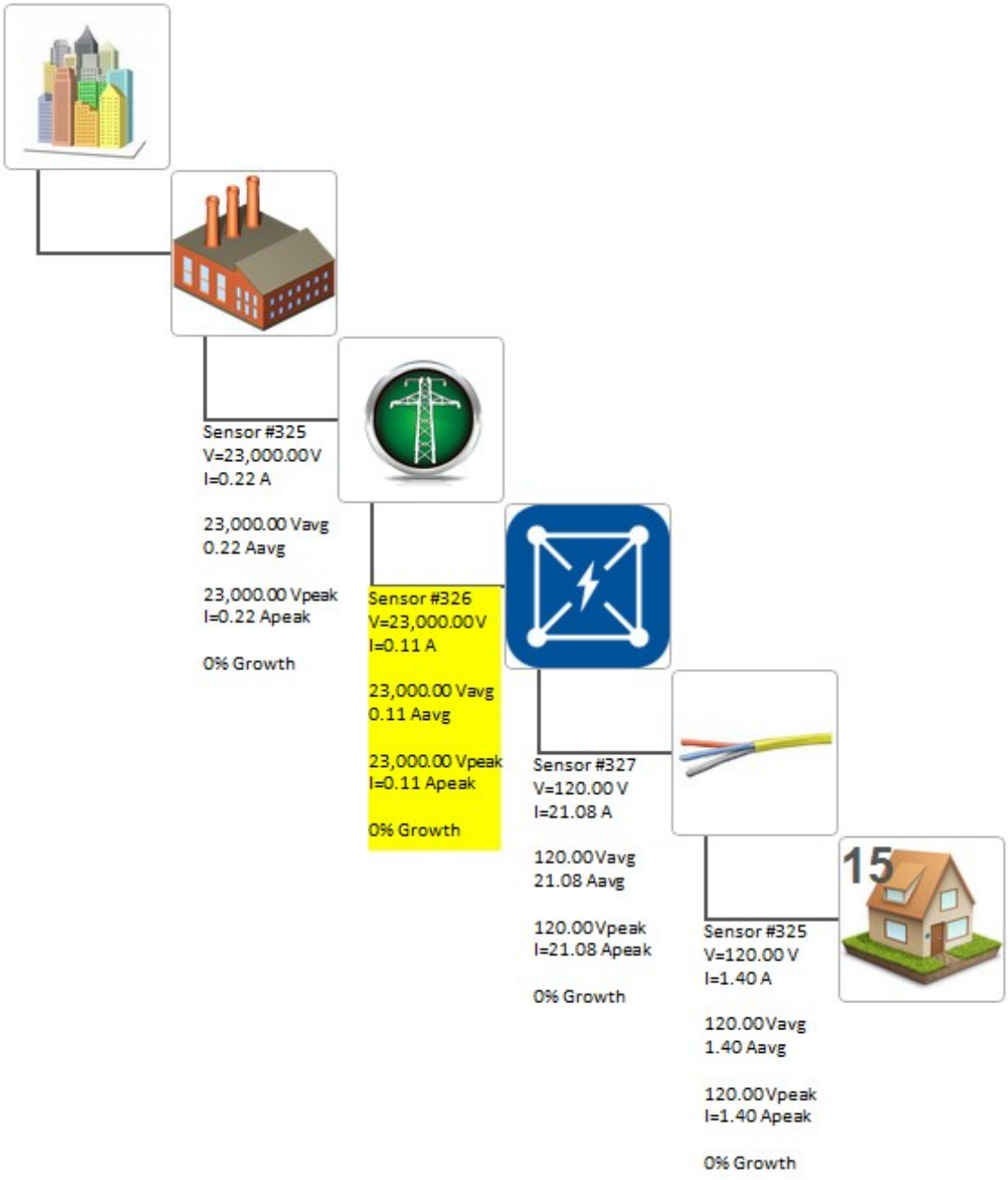

Figure 4.2-4 Inefficient nodes pointed out by the system

4.3 Instance 3 (Faulty Nodes)

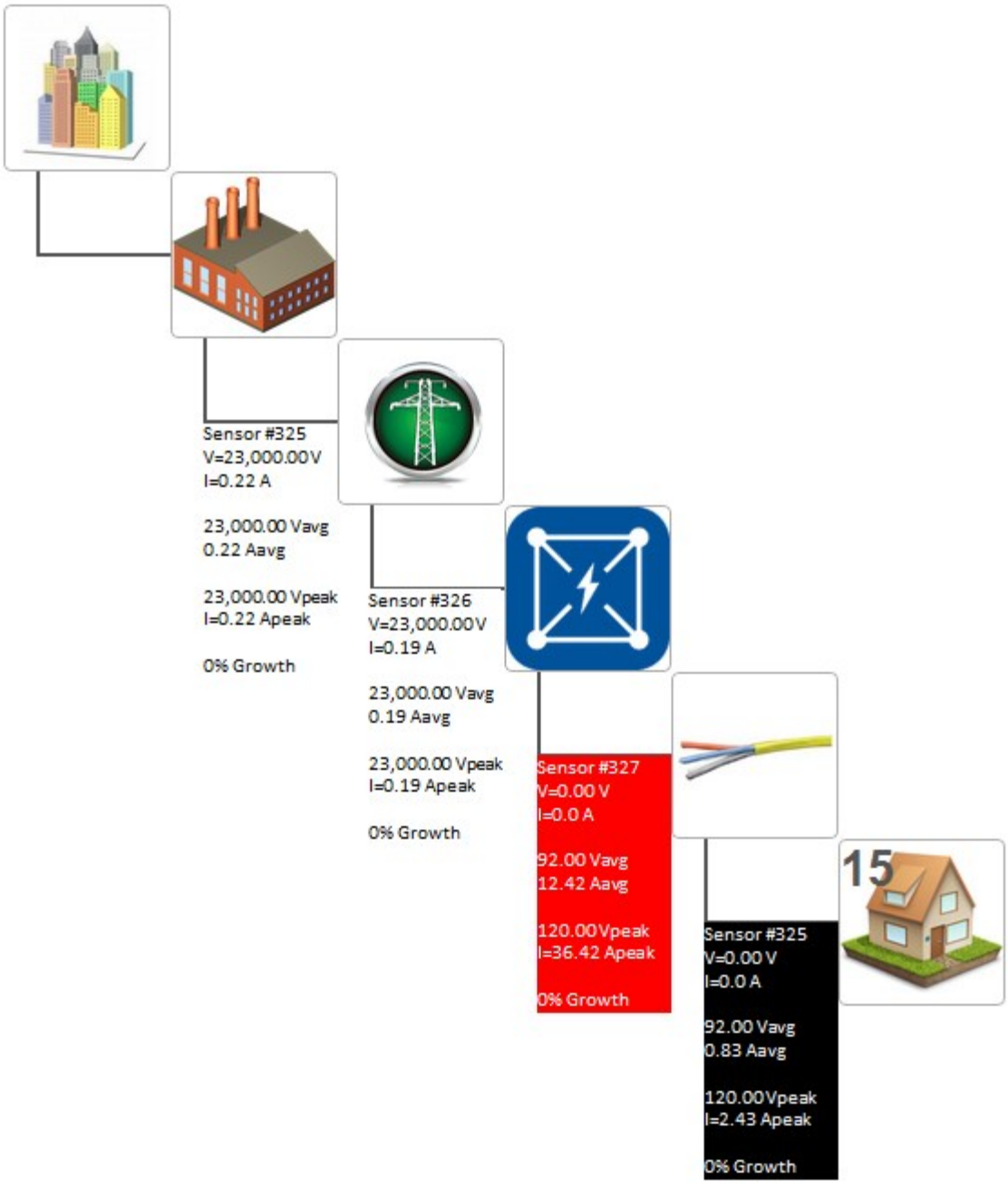

Figure 4.3-1 Blackout nodes and Critical Upgrade node

As you can see in Figure 4.3-1, the peak voltage and current for the power plant and the power wires are above the threshold and need immediate attention. Moreover, the power grid has been deemed faulty as the output of the node is significantly less than the input. These two cases have been pointed out by the system for immediate action and deployment of physical resources to resolve.

4.4 Instance 3 (Urgent Need)

As you can see in Figure 4.4-1, the peak voltage and current for the power plant and the power wires are above the equipment's rated threshold (50 A peak) and needs immediate attention. This cases has been pointed out by the system for immediate action and deployment of physical resources to resolve.

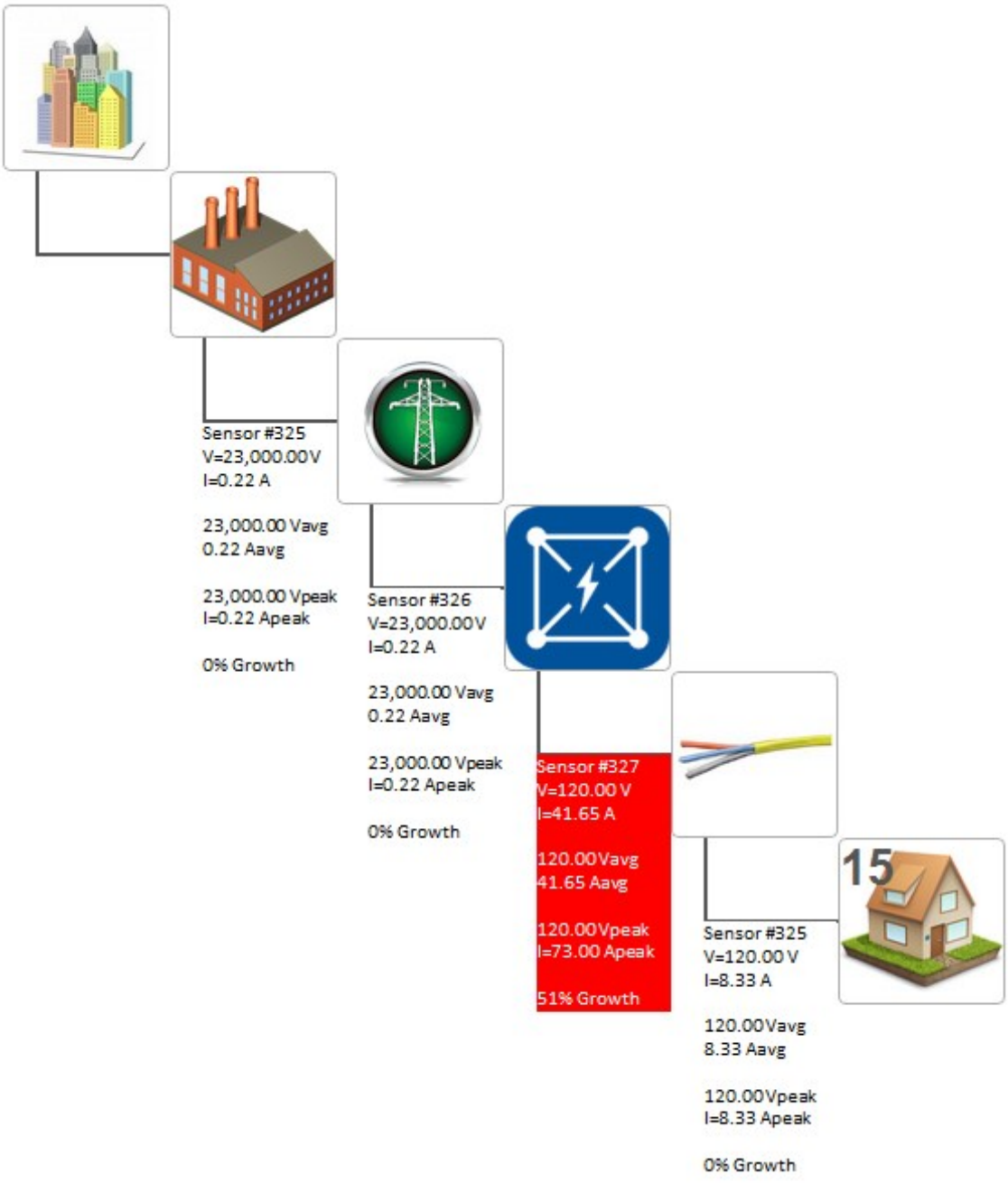

Figure 4.4-1 Alert to upgrade node on urgent need basis

4.5 Instance 4 (Days to upgrade)

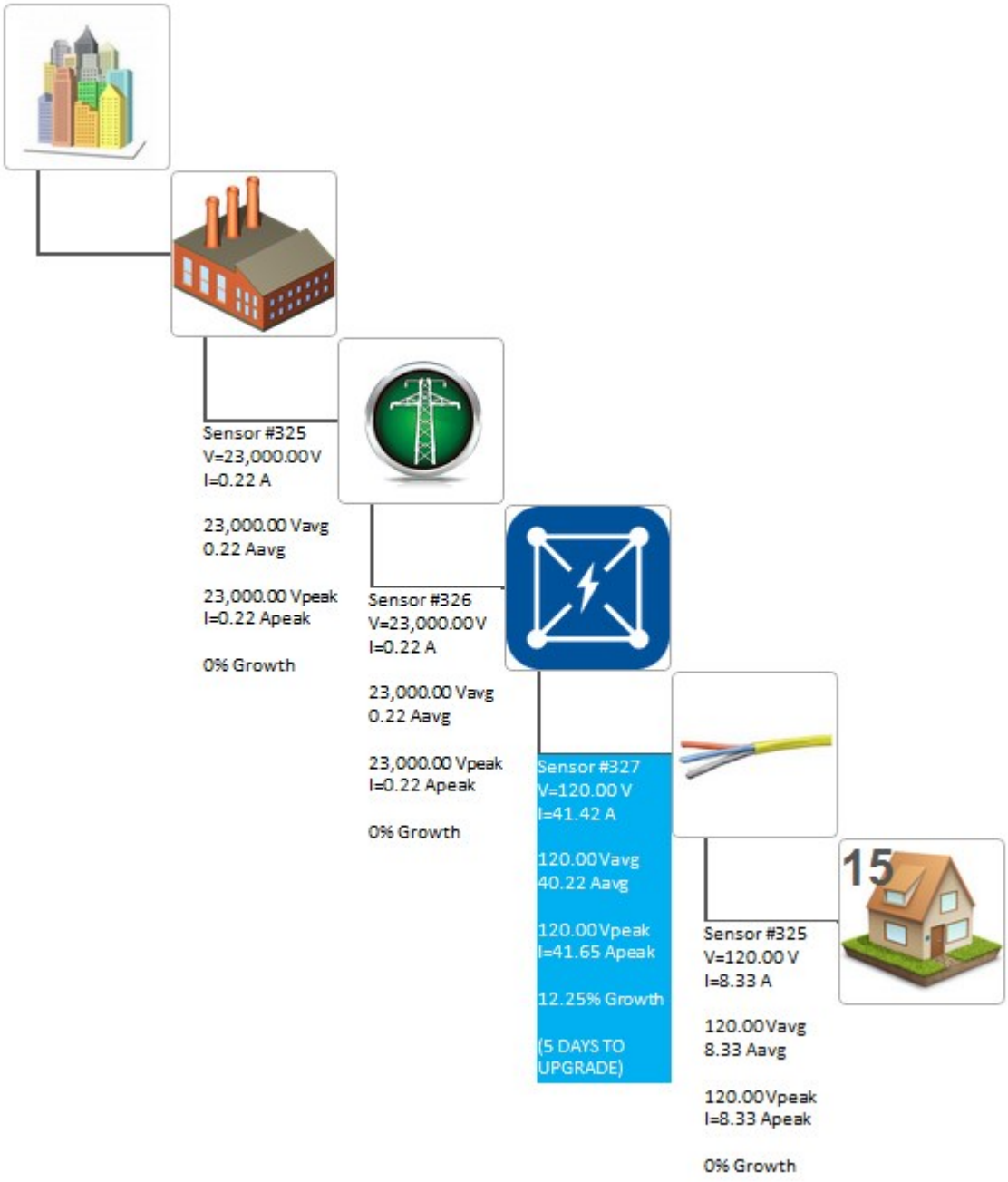

Figure 4.5-1 Showing days to upgrade nodes

Judging by the power growth rate of a node, the system can also suggest how many days to upgrade the node. As you can see in Figure 4.5-1, the power wires have a growth rate of 12.25% and are expected to be upgraded in 5 days. This value has been calculated using the equation in (3.3-16).

4.6 Summary

The following table summarizes the different tests that were performed on the system and what the system displayed to the administrator. Node 0 refers to Node 1's parent. Node 1 refers to the power grid, Node 2 refers to the wires and Node 3 refers to the houses in the system shown in the happy path, above. "%E" refers to the output divided by the input of the respective node.

#	Inefficiency Threshold	Faulty Threshold	Node 0 Output Power (W)	Node 1 Output Power (W)	Node 2 Output Power (W)	Node 3 Output Power (W)
1	90%	50%	5060	4807	3846	3846
			5060	4807	2403.5	2160
2	90%	80%	5060	4554	4553	3640
			5060	4500	3150	3150
3	80%	80%	5060	5040	5035	5000
			5060	4100	3200	2560

4	70%	90%	5060	4550	4500	4400
			5060	5000	3400	3400

Table 4.6-1 Results summarizing different test scenarios for the decision system

In Table 4.6-1 the values highlighted in red are deemed to be upgraded urgently, whereas the nodes highlighted with yellow are deemed to be inefficient nodes.

4.7 Remarks

We have developed a system to help monitor different nodes in the city, and using the data collected we can judge when to upgrade a specific element of the node to avoid blackouts in the city.

One of the main hindrance a power generation organization faces when making upgrade decisions is where and when to upgrade various parts of the grid. Using this system, a company can make incremental upgrades to the power grid, in areas where it is most needed; which makes it more feasible for the organization.

The system above is versatile in the sense that additional nodes that need to be monitored in the city can be added with minimal work. All that needs to be done is a new table has to be constructed in the database that associates that node, and sensors can be deployed to that node.

5 CONCLUSIONS

A practical, decision making/forecasting system was developed and a system architecture for storing/processing the compiled data, was proposed. The central server is able to show real-time statistics and real-time power consumptions on various nodes in the power grid system. The sensors employed here are simulating values but can easily be replaced with actual sensors. In essence, we have developed a complete central server solution for an actual applied purpose.

In order to effectively use this system into action, all that needs to be done is to deploy the sensors and have them configured in the database. Once that is done, the system will be able to read the data off the sensor nodes and be able to show and help in forecasting any future problems.

A graphical user interface was developed for the control centre. This allows a user to easily monitor. In addition, received data that is an indication that a fault has occurred is highlighted so it can be easily identified by the user. The developed prototype operates successfully and its functionality was fully documented.

As a conclusion, the proposed system can be utilized by power distribution companies to effectively monitor the grid and identify problem components in the grid to accommodate EV charging needs. The proposed system can provide significant operating cost savings for utility companies.

5.1 Future Work

Given that this project completes a first prototype, there are many more features that would be helpful in making the system more flexible and efficient. Next is a list of tests and ideas for system features that would be a good start for furthering the abilities of the decision and forecasting system. This list is not intended to be in any particular order of importance.

- See if battery packs as charging stations is a viable solution
- We would be able to see if there are time slots for charging electric vehicles and if that would help mitigate the solution for the time being
- We would be able to see where charging stations would be able to be implemented, and see if pricing (high electricity powers in house whereas cheaper prices in charging station) would help concentrate the power consumption intensity in areas where it can be sustained.
- Security/Encryption can be added to the communication network to help prevent foul play.
- Predict patterns which will help automotive companies to predict market acceptance.
- Fine tune the read data and be able to make more fine decisions to stabilize the system for future penetration of Electric Vehicles.
- Take statistics from one area and apply them to another area where the system wants to be employed.
- Since a communication system would already be in place it can be used to support additional applications. A wider range of sensors can be used to detect overheating, collapse, conductor deterioration or icing, or even weather patterns.
- It is well known that the amount of load on a transmission line varies for many different reasons. It may be desired by the energy providers to be able to remotely change the threshold levels within the detecting devices to indicate a fault.

6 REFERENCES

[1] P. Kelley-Detwiler, "Electric Cars and the Power Grid: How are they coming together?," 28 January 2013. [Online]. Available: http://www.forbes.com/sites/peterdetwiler/2013/01/28/electric-cars-and-the-power-grid-how-are-they-coming-together/.

[2] D. Wright, "As gas prices rise, businesses explore opportunities of electric vehicles," 18 April 2011. [Online]. Available: http://www.theglobeandmail.com/technology/science/as-gas-prices-rise-businesses-explore-opportunities-of-electric-vehicles/article577743?page=all.

[3] M. Bellis, "Inventors," [Online]. Available: http://inventors.about.com/od/estartinventions/a/History-Of-Electric-Vehicles.htm.

[4] J. Chaves, "The Gas Engine vs The Electric Engine," The Breeze, 10 December 2012. [Online]. Available: http://www.sbrhsbreeze.org/opinion/2012/12/10/the-gas-engine-vs-the-electric-engine/.

[5] . R. Dejan, R. Venkatramana and G. David, "Embedded Web Server for Wireless Sensor Networks," in *IEEE*, 2009.

[6] H. Baek, J. Lim and S. Oh, "Beacon-Based Slotted ALOHA for Wireless Networks with Large Propagation Delay," *IEEE COMMUNICATIONS LETTERS,* vol. 17, no. 11, November 2013.

[7] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing and T. Bangemann, "Towards an architecture for service-oriented process monitoring and control," *IEEE,* 2010.

[8] D. Savio, L. Karlik and S. Karnouskos, "Predicting Energy Measurements of

service-enabled devices in the future smartgrid," *Internation Conference on Computer Modelling and Simulation,* no. 12th, 2010.

[9] J. R. Agüero, "Applying Self-Healing Schemes to Modern Power Distribution Systems," *IEEE,* 2012.

[10] S. Karnouskos and T. N. d. Holanda, "Simulation of a Smart Grid City with software agents," *Third UKSim European Symposium on Computer Modeling and Simulation,* 2009.

[11] S. Karnouskos, "Cyber-Physical Systems in the SmartGrid".

[12] L. M., Interviewee, *Cisco: Smart grid will eclipse size of the Internet http://news.cnet.com/8301-11128_3-10241102-54.html.* [Interview]. May 2009.

[13] M. V., "The next billion SAP users will be smart meters," July 2009. [Online]. Available: http://dealarchitect.typepad.com/deal_architect/2009/07/the-next-billion-sap-users-will-be-smart-meters.html.

[14] National Institute of Standards and Technology (NIST), "NiST guidelines for smart grid cyber security," September 2010. [Online]. Available: http://csrc.nist.gov/publications/drafts/nistir-7628-r1/draft_nistir_7628_r1_vol2.pdf.

[15] Laeeq, "Top 5 PHP frameworks 2012," phpzag.com, 30 May 2012. [Online]. Available: http://www.phpzag.com/top-5-php-frameworks-2012/.

[16] L. L. Katha and S. Katha, "Analyzing PHP Frameworks for Use in a Project-Based Software Engineering Course," in *IEEE*, Southern Connecticut State University, 501 Crescent Street, New Haven, CT 06515 USA, 2013.

[17] Netcraft News, "April 2013 Web Server Survey," 2 April 2013. [Online]. Available: http://news.netcraft.com/archives/2013/04/02/april-2013-web-server-survey.html.

[18] M. Davis, "JBoss vs. Tomcat: Choosing A Java Application Server," 29 July 2013. [Online]. Available: http://www.futurehosting.com/blog/jboss-vs-tomcat-choosing-a-java-application-server/.

[19] K. Chan, "Choose the right PHP framework," Creative Bloq, 27 December 2012. [Online]. Available: http://www.creativebloq.com/design/choose-right-php-framework-12122774.

[20] . V. Minh-Thanh, N. Minh-Triet and N. Tuan-Duc, "Towards Residential Smart Grid: A Practical Design of Wireless Sensor Network and Mini-Web Server Based Low Cost Home Energy Monitoring System," in *International Conference on Advanced Technologies for Communications (ATC'13)*, 2013.

7 APPENDIX A

7.1 Program Codes

The following are the current codes of the system:

7.1.1 Sensors

7.1.1.1 Linker Class

```
class linker {
    public function __construct () {
        global $config;
        $this->serverPage = $config['serverPage'];
    }

    public function sendPacket ($a) {
        $p = array();
        foreach ($a as $k => $v) {
            $p[] = urlencode($k)."=".urlencode($v);
        }

        $params = implode ("&", $p);

        $response = file_get_contents ($this->serverPage."&".$params);

        return $response;
    }
}
```

7.1.1.2 Sensor Class

```
<?php
  class Sensor {
      public function __construct ($id=-1) {
          $this->linker = new linker();
          $this->voltage = -1;
```

```
        $this->current = -1;
        $this->id = $id;
    }

    public function sendValue() {
        $parameters = array(
            "v" => $this->voltage,
            "i" => $this->current,
            "id" => $this->id,
            "time" => date("Y-m-d H:i:s")
        );

        $response = $this->linker->sendPacket($parameters);
        print 'Sent value from Sensor Id:'.$this->id."\n";
    }

    public function readValue() {
        $this->voltage = rand(100, 120);
        $this->current = rand(10, 50);
    }

    public function run() {
        $this->readValue();
        $this->sendValue();
    }
  }
?>
```

7.1.2 Central Server

7.1.2.1 Models

7.1.2.1.1 City

```
<?php

/**
```

```
 * This is the model class for table "city".
 *
 * The followings are the available columns in table 'city':
 * @property string $id
 * @property string $name
 *
 * The followings are the available model relations:
 * @property Powerplant $id0
 * @property Powerplant[] $powerplants
 */
class City extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'city';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('name', 'length', 'max'=>45),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, name', 'safe', 'on'=>'search'),
        );
    }
```

```
/**
 * @return array relational rules.
 */
public function relations()
{
    // NOTE: you may need to adjust the relation name and the related
    // class name for the relations automatically generated below.
    return array(
        'powerplants' => array(self::HAS_MANY, 'Powerplant', 'cityId'),
    );
}

/**
 * @return array customized attribute labels (name=>label)
 */
public function attributeLabels()
{
    return array(
        'id' => 'ID',
        'name' => 'Name',
    );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
 */
```

```
    public function search()
    {
        // @todo Please modify the following code to remove attributes that should not be searched.

        $criteria=new CDbCriteria;

        $criteria->compare('id',$this->id,true);
        $criteria->compare('name',$this->name,true);

        return new CActiveDataProvider($this, array(
            'criteria'=>$criteria,
        ));
    }

    /**
     * Returns the static model of the specified AR class.
     * Please note that you should have this exact method in all your CActiveRecord descendants!
     * @param string $className active record class name.
     * @return City the static model class
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }

    public function behaviors()
    {
        return array(
            'activerecord-relation'=>array(
                'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
            ),
        );
    }
}
```

7.1.2.1.2 PowerPlant

```
<?php

/**
 * This is the model class for table "powerPlant".
 *
 * The followings are the available columns in table 'powerPlant':
 * @property string $id
 * @property string $cityId
 */
class PowerPlant extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'powerPlant';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('cityId', 'length', 'max'=>10),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, cityId', 'safe', 'on'=>'search'),
        );
    }
```

```
/**
 * @return array relational rules.
 */
public function relations()
{
    // NOTE: you may need to adjust the relation name and the related
    // class name for the relations automatically generated below.
    return array(
        'distributionwires' => array(self::HAS_MANY, 'DistributionWire', 'powerPlantId'),
        'city' => array(self::BELONGS_TO, 'City', 'cityId'),
    );
}

/**
 * @return array customized attribute labels (name=>label)
 */
public function attributeLabels()
{
    return array(
        'id' => 'ID',
        'cityId' => 'City',
    );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
```

```
 */
public function search()
{
    // @todo Please modify the following code to remove attributes that should not be searched.

    $criteria=new CDbCriteria;

    $criteria->compare('id',$this->id,true);
    $criteria->compare('cityId',$this->cityId,true);

    return new CActiveDataProvider($this, array(
        'criteria'=>$criteria,
    ));
}

/**
 * Returns the static model of the specified AR class.
 * Please note that you should have this exact method in all your CActiveRecord descendants!
 * @param string $className active record class name.
 * @return PowerPlant the static model class
 */
public static function model($className=__CLASS__)
{
    return parent::model($className);
}


public function behaviors()
{
    return array(
        'activerecord-relation'=>array(
            'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
        ),
    );
}
```

```
}
```

7.1.2.1.3 DistributionWire

```
<?php

/**
 * This is the model class for table "distributionWire".
 *
 * The followings are the available columns in table 'distributionWire':
 * @property string $id
 * @property string $powerPlantId
 */
class DistributionWire extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'distributionWire';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('powerPlantId', 'length', 'max'=>10),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, powerPlantId', 'safe', 'on'=>'search'),
        );
```

```
}

/**
 * @return array relational rules.
 */
public function relations()
{
    // NOTE: you may need to adjust the relation name and the related
    // class name for the relations automatically generated below.
    return array(
        'powergrids' => array(self::HAS_MANY, 'PowerGrid', 'distributionWireId'),
        'powerplant' => array(self::BELONGS_TO, 'PowerPlant', 'powerPlantId'),
    );
}

/**
 * @return array customized attribute labels (name=>label)
 */
public function attributeLabels()
{
    return array(
        'id' => 'ID',
        'powerPlantId' => 'Power Plant',
    );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
```

```
     * @return CActiveDataProvider the data provider that can return the models
     * based on the search/filter conditions.
     */
    public function search()
    {
        // @todo Please modify the following code to remove attributes that should not be searched.

        $criteria=new CDbCriteria;

        $criteria->compare('id',$this->id,true);
        $criteria->compare('powerPlantId',$this->powerPlantId,true);

        return new CActiveDataProvider($this, array(
            'criteria'=>$criteria,
        ));
    }

    /**
     * Returns the static model of the specified AR class.
     * Please note that you should have this exact method in all your CActiveRecord descendants!
     * @param string $className active record class name.
     * @return DistributionWire the static model class
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }

    public function behaviors()
    {
        return array(
            'activerecord-relation'=>array(
                'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
            ),
        );
```

```
    }
}
```

7.1.2.1.4 PowerGrid

```
<?php

/**
 * This is the model class for table "powerGrid".
 *
 * The followings are the available columns in table 'powerGrid':
 * @property string $id
 * @property string $distributionWireId
 */
class PowerGrid extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'powerGrid';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('distributionWireId', 'length', 'max'=>10),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, distributionWireId', 'safe', 'on'=>'search'),
```

```
        );
    }

    /**
     * @return array relational rules.
     */
    public function relations()
    {
        // NOTE: you may need to adjust the relation name and the related
        // class name for the relations automatically generated below.
        return array(
            'wires' => array(self::HAS_MANY, 'Wire', 'powerGridId'),
            'distributionwire' => array(self::BELONGS_TO, 'DistributionWire', 'distributionWireId'),
        );
    }

    /**
     * @return array customized attribute labels (name=>label)
     */
    public function attributeLabels()
    {
        return array(
            'id' => 'ID',
            'distributionWireId' => 'Distribution Wire',
        );
    }

    /**
     * Retrieves a list of models based on the current search/filter conditions.
     *
     * Typical usecase:
     * - Initialize the model fields with values from filter form.
     * - Execute this method to get CActiveDataProvider instance which will filter
     * models according to data in model fields.
     * - Pass data provider to CGridView, CListView or any similar widget.
```

```
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
 */
public function search()
{
    // @todo Please modify the following code to remove attributes that should not be searched.

    $criteria=new CDbCriteria;

    $criteria->compare('id',$this->id,true);
    $criteria->compare('distributionWireId',$this->distributionWireId,true);

    return new CActiveDataProvider($this, array(
        'criteria'=>$criteria,
    ));
}

/**
 * Returns the static model of the specified AR class.
 * Please note that you should have this exact method in all your CActiveRecord descendants!
 * @param string $className active record class name.
 * @return PowerGrid the static model class
 */
public static function model($className=__CLASS__)
{
    return parent::model($className);
}


public function behaviors()
{
    return array(
        'activerecord-relation'=>array(
            'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
```

```
            ),
        );
    }
}
```

7.1.2.1.5 Wire

```
<?php

/**
 * This is the model class for table "wire".
 *
 * The followings are the available columns in table 'wire':
 * @property string $id
 * @property string $powerGridId
 *
 * The followings are the available model relations:
 * @property Chargingstation[] $chargingstations
 * @property House[] $houses
 * @property Office[] $offices
 * @property Powergrid $powergr
 * @property Powergrid $powerGrid
 * @property Office $id0
 */
class Wire extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'wire';
    }

    /**
     * @return array validation rules for model attributes.
```

```
 */
public function rules()
{
    // NOTE: you should only define rules for those attributes that
    // will receive user inputs.
    return array(
        array('powerGridId', 'length', 'max'=>10),
        // The following rule is used by search().
        // @todo Please remove those attributes that should not be searched.
        array('id, powerGridId', 'safe', 'on'=>'search'),
    );
}

/**
 * @return array relational rules.
 */
public function relations()
{
    // NOTE: you may need to adjust the relation name and the related
    // class name for the relations automatically generated below.
    return array(
        'chargingstations' => array(self::HAS_MANY, 'Chargingstation', 'wireId'),
        'houses' => array(self::HAS_MANY, 'House', 'wireId'),
        'offices' => array(self::HAS_MANY, 'Office', 'wireId'),
        'powerGrid' => array(self::BELONGS_TO, 'Powergrid', 'powerGridId'),
    );
}

/**
 * @return array customized attribute labels (name=>label)
 */
public function attributeLabels()
{
    return array(
        'id' => 'ID',
```

```
            'powerGridId' => 'Power Grid',
        );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
 */
public function search()
{
    // @todo Please modify the following code to remove attributes that should not be searched.

    $criteria=new CDbCriteria;

    $criteria->compare('id',$this->id,true);
    $criteria->compare('powerGridId',$this->powerGridId,true);

    return new CActiveDataProvider($this, array(
        'criteria'=>$criteria,
    ));
}

/**
 * Returns the static model of the specified AR class.
 * Please note that you should have this exact method in all your CActiveRecord descendants!
 * @param string $className active record class name.
 * @return Wire the static model class
```

```
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }


    public function behaviors()
    {
        return array(
            'activerecord-relation'=>array(
                'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
            ),
        );
    }
}
```

7.1.2.1.6 House

```
<?php

/**
 * This is the model class for table "house".
 *
 * The followings are the available columns in table 'house':
 * @property string $id
 * @property string $wireId
 *
 * The followings are the available model relations:
 * @property Wire $wire
 */
class House extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
```

```
public function tableName()
{
    return 'house';
}

/**
 * @return array validation rules for model attributes.
 */
public function rules()
{
    // NOTE: you should only define rules for those attributes that
    // will receive user inputs.
    return array(
        array('wireId', 'length', 'max'=>10),
        // The following rule is used by search().
        // @todo Please remove those attributes that should not be searched.
        array('id, wireId', 'safe', 'on'=>'search'),
    );
}

/**
 * @return array relational rules.
 */
public function relations()
{
    // NOTE: you may need to adjust the relation name and the related
    // class name for the relations automatically generated below.
    return array(
        'wire' => array(self::BELONGS_TO, 'Wire', 'wireId'),
    );
}

/**
 * @return array customized attribute labels (name=>label)
 */
```

```
public function attributeLabels()
{
    return array(
        'id' => 'ID',
        'wireId' => 'Wire',
    );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
 */
public function search()
{
    // @todo Please modify the following code to remove attributes that should not be searched.

    $criteria=new CDbCriteria;

    $criteria->compare('id',$this->id,true);
    $criteria->compare('wireId',$this->wireId,true);

    return new CActiveDataProvider($this, array(
        'criteria'=>$criteria,
    ));
}

/**
```

```
     * Returns the static model of the specified AR class.
     * Please note that you should have this exact method in all your CActiveRecord descendants!
     * @param string $className active record class name.
     * @return House the static model class
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }


    public function behaviors()
    {
        return array(
            'activerecord-relation'=>array(
                'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
            ),
        );
    }
}
```

7.1.2.1.7 Office

```
<?php

/**
 * This is the model class for table "office".
 *
 * The followings are the available columns in table 'office':
 * @property string $id
 * @property string $wireId
 *
 * The followings are the available model relations:
 * @property Wire $wire
 */
class Office extends CActiveRecord
```

```
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'office';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('wireId', 'length', 'max'=>10),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, wireId', 'safe', 'on'=>'search'),
        );
    }

    /**
     * @return array relational rules.
     */
    public function relations()
    {
        // NOTE: you may need to adjust the relation name and the related
        // class name for the relations automatically generated below.
        return array(
            'wire' => array(self::BELONGS_TO, 'Wire', 'wireId'),
        );
    }
```

```
/**
 * @return array customized attribute labels (name=>label)
 */
public function attributeLabels()
{
    return array(
        'id' => 'ID',
        'wireId' => 'Wire',
    );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
 */
public function search()
{
    // @todo Please modify the following code to remove attributes that should not be searched.

    $criteria=new CDbCriteria;

    $criteria->compare('id',$this->id,true);
    $criteria->compare('wireId',$this->wireId,true);

    return new CActiveDataProvider($this, array(
        'criteria'=>$criteria,
```

```
        ));
    }

    /**
     * Returns the static model of the specified AR class.
     * Please note that you should have this exact method in all your CActiveRecord descendants!
     * @param string $className active record class name.
     * @return Office the static model class
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }

    public function behaviors()
    {
        return array(
            'activerecord-relation'=>array(
                'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
            ),
        );
    }
}
```

7.1.2.1.8 ChargingStation

```
<?php

/**
 * This is the model class for table "chargingstation".
 *
 * The followings are the available columns in table 'chargingstation':
 * @property string $id
 * @property string $wireId
 *
 * The followings are the available model relations:
```

```
 * @property Wire $wire
 */
class ChargingStation extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'chargingstation';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('wireId', 'length', 'max'=>10),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, wireId', 'safe', 'on'=>'search'),
        );
    }

    /**
     * @return array relational rules.
     */
    public function relations()
    {
        // NOTE: you may need to adjust the relation name and the related
        // class name for the relations automatically generated below.
        return array(
```

```
            'wire' => array(self::BELONGS_TO, 'Wire', 'wireId'),
        );
    }

    /**
     * @return array customized attribute labels (name=>label)
     */
    public function attributeLabels()
    {
        return array(
            'id' => 'ID',
            'wireId' => 'Wire',
        );
    }

    /**
     * Retrieves a list of models based on the current search/filter conditions.
     *
     * Typical usecase:
     * - Initialize the model fields with values from filter form.
     * - Execute this method to get CActiveDataProvider instance which will filter
     * models according to data in model fields.
     * - Pass data provider to CGridView, CListView or any similar widget.
     *
     * @return CActiveDataProvider the data provider that can return the models
     * based on the search/filter conditions.
     */
    public function search()
    {
        // @todo Please modify the following code to remove attributes that should not be searched.

        $criteria=new CDbCriteria;

        $criteria->compare('id',$this->id,true);
        $criteria->compare('wireId',$this->wireId,true);
```

```
        return new CActiveDataProvider($this, array(
            'criteria'=>$criteria,
        ));
    }

    /**
     * Returns the static model of the specified AR class.
     * Please note that you should have this exact method in all your CActiveRecord descendants!
     * @param string $className active record class name.
     * @return ChargingStation the static model class
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }


    public function behaviors()
    {
        return array(
            'activerecord-relation'=>array(
                'class'=>'ext.yiiext.behaviors.activerecord-relation.EActiveRecordRelationBehavior',
            ),
        );
    }
}
```

7.1.2.1.9 Sensor

```
<?php

/**
 * This is the model class for table "sensor".
 *
 * The followings are the available columns in table 'sensor':
```

```
 * @property string $id
 * @property string $unit
 * @property string $unitId
 *
 * The followings are the available model relations:
 * @property Sensorvalue[] $sensorvalues
 */
class Sensor extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'sensor';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('unit, unitId', 'required'),
            array('unit', 'length', 'max'=>45),
            array('unitId', 'length', 'max'=>10),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, unit, unitId', 'safe', 'on'=>'search'),
        );
    }

    /**
```

```
 * @return array relational rules.
 */
public function relations()
{
    // NOTE: you may need to adjust the relation name and the related
    // class name for the relations automatically generated below.
    return array(
        'sensorvalues' => array(self::HAS_MANY, 'Sensorvalue', 'sensorId'),
    );
}

/**
 * @return array customized attribute labels (name=>label)
 */
public function attributeLabels()
{
    return array(
        'id' => 'ID',
        'unit' => 'Unit',
        'unitId' => 'Unit',
    );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
 */
```

```
    public function search()
    {
        // @todo Please modify the following code to remove attributes that should not be searched.

        $criteria=new CDbCriteria;

        $criteria->compare('id',$this->id,true);
        $criteria->compare('unit',$this->unit,true);
        $criteria->compare('unitId',$this->unitId,true);

        return new CActiveDataProvider($this, array(
            'criteria'=>$criteria,
        ));
    }

    /**
     * Returns the static model of the specified AR class.
     * Please note that you should have this exact method in all your CActiveRecord descendants!
     * @param string $className active record class name.
     * @return Sensor the static model class
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }
}
```

7.1.2.1.10 SensorValue

```
<?php

/**
 * This is the model class for table "sensorValue".
 *
 * The followings are the available columns in table 'sensorValue':
 * @property string $id
```

```
 * @property string $sensorId
 * @property double $voltage
 * @property double $current
 * @property string $time
 */
class SensorValue extends CActiveRecord
{
    /**
     * @return string the associated database table name
     */
    public function tableName()
    {
        return 'sensorValue';
    }

    /**
     * @return array validation rules for model attributes.
     */
    public function rules()
    {
        // NOTE: you should only define rules for those attributes that
        // will receive user inputs.
        return array(
            array('voltage, current', 'numerical'),
            array('sensorId', 'length', 'max'=>10),
            array('time', 'safe'),
            // The following rule is used by search().
            // @todo Please remove those attributes that should not be searched.
            array('id, sensorId, voltage, current, time', 'safe', 'on'=>'search'),
        );
    }

    /**
     * @return array relational rules.
     */
```

```
public function relations()
{
    // NOTE: you may need to adjust the relation name and the related
    // class name for the relations automatically generated below.
    return array(
    );
}

/**
 * @return array customized attribute labels (name=>label)
 */
public function attributeLabels()
{
    return array(
        'id' => 'ID',
        'sensorId' => 'Sensor',
        'voltage' => 'Voltage',
        'current' => 'Current',
        'time' => 'Time',
    );
}

/**
 * Retrieves a list of models based on the current search/filter conditions.
 *
 * Typical usecase:
 * - Initialize the model fields with values from filter form.
 * - Execute this method to get CActiveDataProvider instance which will filter
 * models according to data in model fields.
 * - Pass data provider to CGridView, CListView or any similar widget.
 *
 * @return CActiveDataProvider the data provider that can return the models
 * based on the search/filter conditions.
 */
public function search()
```

```
    {
        // @todo Please modify the following code to remove attributes that should not be searched.

        $criteria=new CDbCriteria;

        $criteria->compare('id',$this->id,true);
        $criteria->compare('sensorId',$this->sensorId,true);
        $criteria->compare('voltage',$this->voltage);
        $criteria->compare('current',$this->current);
        $criteria->compare('time',$this->time,true);

        return new CActiveDataProvider($this, array(
            'criteria'=>$criteria,
        ));
    }

    /**
     * Returns the static model of the specified AR class.
     * Please note that you should have this exact method in all your CActiveRecord descendants!
     * @param string $className active record class name.
     * @return SensorValue the static model class
     */
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }
}
```

7.1.2.2 Views

7.1.2.2.1 Creator

```
<?php
/* @var $this SiteController */

$this->pageTitle=Yii::app()->name;
?>
```

```
<h1>Creator</h1>

<p>This part of the application is used to randomly generate a city for demonstration purposes.</p>
<p>The following will be created in this process:</p>
<ul>
    <li>City</li>
    <li>Power Stations</li>
    <li>Power Wires (bigWires)</li>
    <li>Power Grid Stations</li>
    <li>Distribution Wires (wires)</li>
    <li>Houses</li>
    <li>Offices</li>
    <li>Charging Stations</li>
</ul>
<p>Moreover, sensors will be deployed on various end points randomly to read data from</p>
<p>If you understand the above, you can click the button below to start the generation</p>
<div align='left'>
    <input type='submit' value='Generate City' onClick='window.location.href="<?=$this->createUrl('/site/creatorLoading');?>";' />
</div>
```

7.1.2.2.2 Creator Loading

```
<?php
/* @var $this SiteController */

$this->pageTitle=Yii::app()->name;
?>

<h1>Creating City... Please wait</h1>
<div align=center id="loading">
    <img src="<?=Yii::app()->baseUrl.'/images/loading.gif';?>" />
</div>

<script type="text/javascript">
```

```
    jQuery('document').ready(function($) {
        $('#loading').load('<?=$this->createUrl('/site/doCreator');?>');
    });
</script>
```

7.1.2.2.3 Do Creator

```
<?=CHtml::link('Click here to view your newly created city', array('/site/doView', 'cityId' => $cityId));?>
```

7.1.2.2.4 Do Creator Page

```
<?=CHtml::link('Click here to view your newly created city', array('/site/doView', 'cityId' => $cityId));?>
```

7.1.2.2.5 Do View

```
<style type='text/css'>
    .icon {
        width:100px;
        height:100px;
        border-style:solid;
        border-width:1px;
        border-color:#aaa;
        border-radius:5px;
        background: no-repeat center center;
        background-size:contain;
        font-weight:bold;
        font-size:30px;
    }

    .city {background-image: url('<?=Yii::app()->baseUrl.'/images/city.jpg';?>');}
    .powerPlant {background-image: url('<?=Yii::app()->baseUrl.'/images/powerPlant.png';?>');}
    .distributionWire {background-image: url('<?=Yii::app()->baseUrl.'/images/distributionWire.gif';?>');}
    .powerGrid {background-image: url('<?=Yii::app()->baseUrl.'/images/powerGrid.gif';?>');}
    .wire {background-image: url('<?=Yii::app()->baseUrl.'/images/wire.png';?>');}
    .chargingStation {background-image: url('<?=Yii::app()->baseUrl.'/images/chargingStation.png';?>');}
    .office {background-image: url('<?=Yii::app()->baseUrl.'/images/office.png';?>');}
    .house {background-image: url('<?=Yii::app()->baseUrl.'/images/house.jpg';?>');}
```

```
    .diagram {
        width:auto;
    }

    .diagram tr td {
        padding:0px;
        width:100px;
        height:100px;
        vertical-align: top;
    }

    .diagram tr td .lineDiv1, .diagram tr td .lineDiv2{
        width:50px;
        height:50px;
        border-width:3px;
        float:right;
    }

    .diagram tr .corner .lineDiv1{
        border-left-style:solid;
        border-bottom-style:solid;
    }

    .diagram tr .corner .lineDiv2{
        border-left-style:solid;
    }

</style>

<?php
    if (!(isset($sensored) && $sensored)) {
        echo CHtml::link('Deploy sensors', array('/site/deploySensors', 'cityId' => $city->id));
    } else {
        echo 'Sensors have been deployed successfully, '.CHtml::link('Click here to view this city realtime', array('/site/realtime', 'cityId' => $city->id));;
```

```
    }
?>

<table cellspacing=0 cellpadding=0 class="diagram">
    <tr>
        <td><div class="icon city"></div></td><td></td><td></td><td></td><td></td><td></td><td></td>
    </tr>
    <?php foreach ($city->powerplants as $p) { ?>
        <tr>
            <td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td><div class="icon powerPlant"></div></td><td></td><td></td><td></td><td></td>
        </tr>
            <?php foreach ($p->distributionwires as $d) { ?>
                <tr>
                    <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td><div class="icon distributionWire"></div></td><td></td><td></td><td></td>
                </tr>
                    <?php foreach ($d->powergrids as $g) { ?>
                        <tr>
                            <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td><div class="icon powerGrid"></div></td><td></td><td></td>
                        </tr>
                            <?php foreach ($g->wires as $w) { ?>
                                <tr>
                                    <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;"
```

/> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td> </td><td></td>

</tr>

<tr>

<td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td> <?=count($w->houses);?> </td>

</tr>

<tr>

<td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td> <?=count($w->offices);?> </td>

</tr>

<tr>

<td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /> <br style="clear:both;" /></td><td class="corner" align=right> <br style="clear:both;" /></td><td> <?=count($w->chargingstations);?> </td>

```
                                </tr>
                            <?php } ?>
                        <?php } ?>
                    <?php } ?>
                <?php } ?>
</table>
```

7.1.2.2.6 Error

```
<?php
/* @var $this SiteController */
/* @var $error array */

$this->pageTitle=Yii::app()->name . ' - Error';
$this->breadcrumbs=array(
    'Error',
);
?>

<h2>Error <?php echo $code; ?></h2>

<div class="error">
<?php echo CHtml::encode($message); ?>
</div>
```

7.1.2.2.7 Get Sensor Values

```
<?php
    echo "Sensor #".$sensor->id."<br />V=".
    number_format((float)$sensorValue->voltage, 2, '.', '')." V<br />I=".
    number_format((float)$sensorValue->current, 2, '.', '')." A<br />".

    number_format((float)$model->avgVoltage, 2, '.', '')." Vavg<br />I=".
    number_format((float)$model->avgCurrent, 2, '.', '')." Aavg<br />".

    number_format((float)$model->maxVoltage, 2, '.', '')." Vpeak<br />I=".
```

```
    number_format((float)$model->maxCurrent, 2, '.', '')." Apeak<br />";
?>
```

7.1.2.2.8 Index

```
<?php
/* @var $this SiteController */

$this->pageTitle=Yii::app()->name;
?>

<h1>Welcome to <i><?php echo CHtml::encode(Yii::app()->name); ?></i></h1>

<h2>Abstract</h2>
<p>With the increasing prices of gasoline, more energy companies along with car manufacturers are looking at employing electric energy for modern vehicles, i.e. Electric Vehicles (EVs). In turn this forward leap has some unwanted side effects on the current power grid system we have in place, i.e. wires not able to sustain such high currents for prolonged periods of time, transformers not able to sustain a stable voltage, etc. Thus, we have to come up with an incremental upgrade approach to the current power grid system. To solve this issue we need to: </p>
<ul>
    <li>Monitor voltage and current output of various nodes in the electrical distribution system.</li>
    <li>Be able to watch for patterns and estimate when a wire/transformer or power grid should be upgraded.</li>
    <li>Have data from the nodes come to a central server on a real-time basis for future speculations.</li>
</ul>

<p>Please select one of the options from the top menu bar to continue</p>
```

7.1.2.2.9 Realtime

```
<style type='text/css'>
    .icon {
        width:100px;
        height:100px;
        border-style:solid;
        border-width:1px;
        border-color:#aaa;
```

```
    border-radius:5px;
    background: no-repeat center center;
    background-size:contain;
    font-weight:bold;
    font-size:30px;
}

.city {background-image: url('<?=Yii::app()->baseUrl.'/images/city.jpg';?>');}
.powerPlant {background-image: url('<?=Yii::app()->baseUrl.'/images/powerPlant.png';?>');}
.distributionWire {background-image: url('<?=Yii::app()->baseUrl.'/images/distributionWire.gif';?>');}
.powerGrid {background-image: url('<?=Yii::app()->baseUrl.'/images/powerGrid.gif';?>');}
.wire {background-image: url('<?=Yii::app()->baseUrl.'/images/wire.png';?>');}
.chargingStation {background-image: url('<?=Yii::app()->baseUrl.'/images/chargingStation.png';?>');}
.office {background-image: url('<?=Yii::app()->baseUrl.'/images/office.png';?>');}
.house {background-image: url('<?=Yii::app()->baseUrl.'/images/house.jpg';?>');}

.diagram {
    width:auto;
}

.diagram tr td {
    padding:0px;
    width:100px;
    height:100px;
    vertical-align: top;
}

.diagram tr td .lineDiv1, .diagram tr td .lineDiv2{
    width:80px;
    height:50px;
    border-width:3px;
    float:right;
}

.diagram tr .corner .lineDiv1{
```

```
        border-left-style:solid;
        border-bottom-style:solid;
    }

    .diagram tr .corner .lineDiv2{
        border-left-style:solid;
    }

</style>

<script type="text/javascript">
    jQuery('document').ready(function($) {
        function updateVals() {
            $('.sensor').each(function() {
                var id = this.id;
                var a = id.split('-');
                var unit = a[0];
                var unitId = a[1];
                $('#'+id).load('<?=$this->createUrl('/site/getSensorValues');?>'+'&unit='+unit+'&unitId='+unitId);
            });
        }

        setInterval(function(){updateVals();}, 1000);

        updateVals();
    });
</script>

<h1>Realtime City viewer</h1>
<p>This page will update every minute</p>
<br />
<table cellspacing=0 cellpadding=0 class="diagram">
    <tr>
        <td><div class="icon city"></div></td><td></td><td></td><td></td><td></td><td></td><td></td>
    </tr>
```

```
<?php foreach ($city->powerplants as $p) { ?>
    <tr>
        <td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div id="powerPlant-<?=$p->id;?>" class="lineDiv2 sensor"></div></td><td><div class="icon powerPlant"></div></td><td></td><td></td><td></td><td></td>
    </tr>
        <?php foreach ($p->distributionwires as $d) { ?>
            <tr>
                <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div id="distributionWire-<?=$d->id;?>" class="lineDiv2 sensor"></div><br style="clear:both;" /></td><td><div class="icon distributionWire"></div></td><td></td><td></td><td></td>
            </tr>
                <?php foreach ($d->powergrids as $g) { ?>
                    <tr>
                        <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div id="powerGrid-<?=$g->id;?>" class="lineDiv2 sensor"></div></td><td><div class="icon powerGrid"></div></td><td></td><td></td>
                    </tr>
                        <?php foreach ($g->wires as $w) { ?>
                            <tr>
                                <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div id="wire-<?=$w->id;?>" class="lineDiv2 sensor"></div></td><td><div class="icon wire"></div></td><td></td>
                            </tr>
                            <?php foreach ($w->houses as $h) { ?>
                                <tr>
                                    <td class="corner" align=right><div class="lineDiv2"></div><br
```

```
style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div id="house-<?=$h->id;?>" class="lineDiv2 sensor"></div><br style="clear:both;" /></td><td><div class="icon house"><?=count($w->houses);?></div></td>
                                    </tr>
                                <?php } ?>
                                <?php foreach ($w->offices as $o) { ?>
                                    <tr>
                                        <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div id="office-<?=$o->id;?>" class="lineDiv2 sensor"></div><br style="clear:both;" /></td><td><div class="icon office"><?=count($w->offices);?></div></td>
                                    </tr>
                                <?php } ?>
                                <?php foreach ($w->chargingstations as $x) { ?>
                                    <tr>
                                    <td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv2"></div><br style="clear:both;" /><div class="lineDiv2"></div><br style="clear:both;" /></td><td class="corner" align=right><div class="lineDiv1"></div><br style="clear:both;" /><div id="chargingStation-<?=$x->id;?>" class="lineDiv2 sensor"></div></td><td><div class="icon chargingStation"><?=count($w->chargingstations);?></div></td>
                                    </tr>
```

```
                                <?php } ?>
                            <?php } ?>
                        <?php } ?>
                <?php } ?>
    <?php } ?>
</table>
```

7.1.2.2.10 rViewer

```
<?php
/* @var $this SiteController */

$this->pageTitle=Yii::app()->name;
?>

<h1>Please choose a city from below:</i></h1>
<ul>
    <?php
        foreach ($cities as $city) {
            echo "<li>".CHtml::link($city->id .". ".$city->name, array('/site/realtime', 'cityId' => $city->id))."</li>";
        }
    ?>
</ul>
```

7.1.2.2.11 Viewer

```
<?php
/* @var $this SiteController */

$this->pageTitle=Yii::app()->name;
?>

<h1>Please choose a city from below:</i></h1>
<ul>
    <?php
        foreach ($cities as $city) {
```

```
            echo "<li>".CHtml::link($city->id .". ".$city->name, array('/site/doView', 'cityId' => $city->id))."</li>";
        }
    ?>
</ul>
```

7.1.2.3 Controller

7.1.2.3.1 Site Controller

```
<?php

class SiteController extends Controller
{
    /**
     * Declares class-based actions.
     */
    public function actions()
    {
        return array(
            // captcha action renders the CAPTCHA image displayed on the contact page
            'captcha'=>array(
                'class'=>'CCaptchaAction',
                'backColor'=>0xFFFFFF,
            ),
            // page action renders "static" pages stored under 'protected/views/site/pages'
            // They can be accessed via: index.php?r=site/page&view=FileName
            'page'=>array(
                'class'=>'CViewAction',
            ),
        );
    }

    /**
     * This is the default 'index' action that is invoked
     * when an action is not explicitly requested by users.
     */
    public function actionIndex()
```

```
{
    // renders the view file 'protected/views/site/index.php'
    // using the default layout 'protected/views/layouts/main.php'
    $this->render('index');
}

/**
 * This is the action to handle external exceptions.
 */
public function actionError()
{
    if($error=Yii::app()->errorHandler->error)
    {
        if(Yii::app()->request->isAjaxRequest)
            echo $error['message'];
        else
            $this->render('error', $error);
    }
}

public function actionCreator() {
    $this->render('creator');
}

public function actionDoCreator() {
    $houses = array();
    $offices = array();
    $chargingStations = array();
    $wires = array();

    //lets generate a random population first
    $p = rand(1, 500);

    //avergae people in a household
    $h = $p/5;
```

```
//create all the houses
for ($i=0; $i<$h; $i++) {
    $hm = new House();
    $hm->save();
    $houses[] = $hm->id;
}

//average offices occupancy assuming 2 people from each house work
$o = ($p/2)/1000;

//create all the offices
for ($i=0; $i<$o; $i++) {
    $om = new Office();
    $om->save();
    $offices[] = $om->id;
}

//random number of charging stations
$c = rand(1,5);

//create all the charging stations
for ($i=0; $i<$c; $i++) {
    $cm = new ChargingStation();
    $cm->save();
    $chargingStations[] = $cm->id;
}

//generate random number of wires
$w = rand(1,($h/20));

//create all wires
$hl=0;
$ol=0;
$cl=0;
```

```
for ($i=0; $i<=$w; $i++) {
    $hma = array();
    $oma = array();
    $cma = array();

    if ($i==$w) {
        $hma = array_slice ($houses, $hl);
        $oma = array_slice ($offices, $ol);
        $cma = array_slice ($chargingStations, $cl);
    } else {
        $ra = rand(0, count($houses)-$hl);
        if ($hl < count($houses)) $hma = array_slice ($houses, $hl, $ra);
        $hl = count($hma);
        $ra = rand(0, count($offices)-$ol);
        if ($ol < count($offices)) $oma = array_slice ($offices, $ol, $ra);
        $ol = count($oma);
        $ra = rand(0, count($chargingStations)-$cl);
        if ($cl < count($chargingStations)) $cma = array_slice ($chargingStations, $cl, $ra);
        $cl = count($cma);
    }

    if (($hl+$cl+$ol) <= (count($houses)+count($offices)+count($chargingStations))) {
        $wm = new Wire();
        $wm->save();

        $wires[] = $wm->id;

        //now assign houses and offices to this... first create a chunk
        $wm->houses = $hma;
        $wm->offices = $oma;
        $wm->chargingstations = $cma;
        $wm->save();
    } else {
        $w = $i;
        continue;
```

```
    }
}

//now unset everything
unset ($houses);
unset ($offices);
unset ($chargingStations);
unset ($hma);
unset ($oma);
unset ($cma);

//generate random number of powerGrids
$g = rand(1,ceil($w/20));
//create all powerGrids
$wl=0;
for ($i=0; $i<=$g; $i++) {
    $wma = array();

    if ($i==$g) {
        $wma = array_slice ($wires, $wl);
    } else {
        if ($wl <= count($wires)) $wma = array_slice ($wires, $wl, rand(1, count($wires)-$wl));
        $wl += count($wma);
    }

    if ($wl <= count($wires)) {
        $gm = new PowerGrid();
        $gm->save();

        $powerGrids[] = $gm->id;

        //now assign houses and offices to this... first create a chunk
        $gm->wires = $wma;
        $gm->save();
    } else {
```

```
            $g = $i;
            continue;
        }
    }

    //now unset everything
    unset ($wires);
    unset ($wma);


    //generate random number of powerGrids
    $s = rand(1,($g/20));

    //create all powerGrids
    $gl=0;
    for ($i=0; $i<=$s; $i++) {
        $gma = array();

        if ($i==$s) {
            $gma = array_slice ($powerGrids, $gl);
        } else {
            if ($gl < count($powerGrids)) $gma = array_slice ($powerGrids, $gl, rand(0, count($powerGrids)-
$gl));
            $gl += count($gma);
        }

        if ($gl <= count($powerGrids)) {
            $dm = new DistributionWire();
            $dm->save();

            $distributionWires[] = $dm->id;

            //now assign houses and offices to this... first create a chunk
            $dm->powergrids = $gma;
```

```
            $dm->save();
        } else {
            $s = $i;
            continue;
        }
    }

    //now unset everything
    unset ($powerGrids);
    unset ($gma);

    //generate random number of powerGrids
    $x = rand(1,($s/20));

    //create all powerGrids
    $sl=0;
    for ($i=0; $i<=$x; $i++) {
        $sma = array();

        if ($i==$x) {
            $sma = array_slice ($distributionWires, $sl);
        } else {
            if ($sl <= count($distributionWires)) $sma = array_slice ($distributionWires, $sl, rand(0, count($distributionWires)-$sl));
            $sl += count($sma);
        }

        if ($sl <= count($distributionWires)) {
            $xm = new PowerPlant();
            $xm->save();

            $powerPlants[] = $xm->id;

            //now assign houses and offices to this... first create a chunk
            $xm->distributionwires = $sma;
```

```
                $xm->save();
            } else {
                $x = $i;
                continue;
            }
        }

        //now unset everything
        unset ($distributionWires);
        unset ($sma);

        $tm = new City();
        $tm->save();

        $tm->powerplants = $powerPlants;

        $tm->save();

        unset ($powerPlants);

        $data = array();
        $data['cityId'] = $tm->id;
        unset ($tm);

        if(Yii::app()->request->isAjaxRequest) $this->renderpartial('doCreator', $data);
        else $this->render('doCreatorPage', $data);
    }

    public function actionCreatorLoading() {
        $this->render('creatorLoading');
    }

    public function actionViewer() {
        $data['cities'] = City::model()->findAll();
```

```
        $this->render('viewer', $data);
    }

    public function actionRViewer() {
        $data['cities'] = City::model()->findAll();
        $this->render('rviewer', $data);
    }

    public function actionDoView() {
        $data['city'] = City::model()->findByPk($_GET['cityId']);
        $this->render('doView', $data);
    }

    public function actionRealtime() {
        $data['city'] = City::model()->findByPk($_GET['cityId']);
        $this->render('realtime', $data);
    }

    public function actionCalculateValues() {

    }

    public function actionGetSensorValues() {
        if (isset($_GET['sensorId'])) $data['sensor'] = Sensor::model()->findByPk($_GET['sensorId']);
        elseif (isset($_GET['unit']) && isset($_GET['unitId'])) $data['sensor'] = Sensor::model()->find('unit=:unit and
unitId=:unitId', array(':unit' => $_GET['unit'], ':unitId' => $_GET['unitId']));

        $sql='SELECT * FROM sensorvalue where sensorId=:sensorId ORDER BY id DESC LIMIT 1';
        $params=array(':sensorId'=>$data['sensor']->id);
        $s=SensorValue::model()->findAllBySql($sql,$params);
        $data['sensorValue'] = $s[0];

        $modelName = ucfirst($data['sensor']->unit);
        $data['model'] = $modelName::model()->findByPk($data['sensor']->unitId);
```

```
    if(Yii::app()->request->isAjaxRequest) $this->renderpartial('getSensorValues', $data);
    else $this->render('getSensorValues', $data);

}

public function actionSensorUpdate() {
    if (isset($_GET['v']) && isset($_GET['i']) && isset($_GET['id'])) {
        $s = new SensorValue();
        $s->current = $_GET['i'];
        $s->voltage = $_GET['v'];
        $s->time = $_GET['time'];
        $s->sensorId = $_GET['id'];
        $s->save();
    }

    $sensor = Sensor::model()->findByPk($_GET['id']);

    $modelName = ucfirst($sensor->unit);
    $model = $modelName::model()->findByPk($sensor->unitId);

    if ($sensor->current > $model->maxCurrent) $model->maxCurrent = $sensor->current;
    if ($sensor->voltage > $model->maxVoltage) $model->maxVoltage = $sensor->voltage;

    $model->avgVoltage = (double)($sensor->voltage+$model->avgVoltage)/2.0;
    $model->avgCurrent = (double)($sensor->current+$model->avgCurrent)/2.0;

    echo "Successfully updated sensor";
    return;
}

public function actionDeploySensors() {
    $data['city'] = City::model()->findByPk($_GET['cityId']);

    foreach ($data['city']->powerplants as $p) {
        $sensor = new Sensor();
```

```
$sensor->unit = 'powerPlant';
$sensor->unitId = $p->id;
try {
    $sensor->save();
} catch (Exception $e) {
}

foreach ($p->distributionwires as $d) {
    $sensor = new Sensor();
    $sensor->unit = 'distributionWire';
    $sensor->unitId = $d->id;
    try {
        $sensor->save();
    } catch (Exception $e) {
    }

    foreach ($d->powergrids as $g) {
        $sensor = new Sensor();
        $sensor->unit = 'powerGrid';
        $sensor->unitId = $g->id;
        try {
            $sensor->save();
        } catch (Exception $e) {
        }

        foreach ($g->wires as $w) {
            $sensor = new Sensor();
            $sensor->unit = 'wire';
            $sensor->unitId = $w->id;
            try {
                $sensor->save();
            } catch (Exception $e) {
            }

            foreach ($w->houses as $h) {
```

```
                    $sensor = new Sensor();
                    $sensor->unit = 'house';
                    $sensor->unitId = $h->id;
                    try {
                        $sensor->save();
                    } catch (Exception $e) {
                    }
                }

                foreach ($w->offices as $o) {
                    $sensor = new Sensor();
                    $sensor->unit = 'office';
                    $sensor->unitId = $o->id;
                    try {
                        $sensor->save();
                    } catch (Exception $e) {
                    }
                }

                foreach ($w->chargingstations as $c) {
                    $sensor = new Sensor();
                    $sensor->unit = 'chargingStation';
                    $sensor->unitId = $c->id;
                    try {
                        $sensor->save();
                    } catch (Exception $e) {
                    }
                }
            }
        }
    }
}

$data['sensored'] = true;
$this->render('doView', $data);
```

```
    }

    /**
     * Displays the login page
     */
    public function actionLogin()
    {
        $model=new LoginForm;
        // if it is ajax validation request
        if(isset($_POST['ajax']) && $_POST['ajax']==='login-form')
        {
            echo CActiveForm::validate($model);
            Yii::app()->end();
        }
        // collect user input data
        if(isset($_POST['LoginForm']))
        {
            $model->attributes=$_POST['LoginForm'];
            // validate user input and redirect to the previous page if valid
            if($model->validate() && $model->login())
                $this->redirect(Yii::app()->user->returnUrl);
        }
        // display the login form
        $this->render('login',array('model'=>$model));
    }

    /**
     * Logs out the current user and redirect to homepage.
     */
    public function actionLogout()
    {
        Yii::app()->user->logout();
        $this->redirect(Yii::app()->homeUrl);
    }

}
```

8 VITA AUCTORIS

Syed Sami was born in 1989 in Newcastle Upon Tyne, England. He graduated from Islamia College Peshawar, Peshawar, Pakistan in 2007. From there he went on to the University of Windsor where he obtained a B.Sc. in Electrical Engineering in 2011. He is currently a candidate for the Master's degree in Electrical Engineering at the University of Windsor and hopes to graduate in Spring 2014. His research interest includes Wireless Communication Protocols, Simulator Designs, Artificial Intelligence and Image Recognition Systems.