CEMA: Comfort Control and Energy Management Algorithms for use in Residential Spaces through Wireless Sensor Networks

by

Rami F.Z. Henry

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements

for the M.A.Sc. Degree in

Electrical and Computer Engineering

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

Abstract

In recent years, many strides have been achieved in the area of Wireless Sensor Networks (WSNs), which is leading to constant innovations in the types of applications that WSNs can support. Much advancement has also been achieved in the area of smart homes, enabling its occupants to manually and easily control their utility expenses.

In this thesis, both areas of research will be colluded for a simple, yet critical application: efficient and economical comfort control in smart residential spaces. The goal is to design a central, modular energy consumption control system for residential spaces, which manages energy consumption in all aspects of a typical residence. This thesis is concerned with two facets of energy consumption in residences. The first facet is concerned with controlling when the heating, ventilating, and air conditioning unit (HVAC) operates for each room separately. This is in contrast to a typical HVAC system where comfort is provided across the floor as a whole. The second facet is concerned with controlling the lighting in each room so as to not exceed a certain input value. The communication network that supports the realization of these coveted goals is based on Zigbee interconnected sensor nodes which pour data unto a smart thermostat which does all the required calculations and activates the modules required for comfort control and energy management, if needed.

A Java-based discrete event simulator is then written up to simulate a floor of a typical Canadian single-family dwelling. The simulation assumes errorless communication and proceeds to record certain room variables and the ongoing cost of operation periodically. These results from the simulator are compared to the results of the well known simulator, created by DesignBuilder, which describes typical home conditions. The conclusion from this analysis is that the Comfort Control and Energy Management Algorithms (CEMA) are feasible, and that their implementation incurs significant monetary savings.

Acknowledgments

I would like to sincerely thank Dr. Hussein Mouftah for the extensive care, guidance, and especially patience the professor has given throughout my M.A.Sc. studies. It was a privilege to have worked and researched under the guidance of this world-class scientist and professor, in addition to being given a chance at a new life in this country. His personal attention, especially during the summer of 2008, is also duly remembered.

Acknowledgements also go out to two of Professor Mouftah's post-doctoral fellows, Dr. Eshoul and Dr. Erol-Kantarci, for their invaluable guidance. Dr. Eshoul's section of the code that forms the basis of the queue where discrete events are placed to be parsed and analyzed is duly noted and I thus state that this part of the code is Dr. Eshoul's. Dr. Erol-Kantarci's .dll file that describes average random usage of four common household appliances is also duly noted and I thus state that this particular .dll file is Dr. Erol-Kantarci's, even though it was not used in this thesis.

To my dear colleagues in the SITE 4001 lab, I thank you for the opportunity of getting to know such broad-minded, wise, and fun researchers and for your assistance throughout my studies.

Last but not least, I would like to thank my parents for the tiring monetary and psychological sacrifices they had to entail so that I could realize the dream of further education and start a new life here in Canada.

I would like to dedicate this Master Thesis to my late mother, who succumbed to late-stage breast cancer in August 2008. Her words of advice for my new life in Canada, in addition to her sacrifices for me during her long fight with cancer, will never be forgotten.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AHU	Air Handling Unit
ASHRAE	American Society of Heating, Refrigerating, and Air-Conditioning Engineers
BACnet	Building Automation and Control network
BER	Bit Error Rate
BTU	British Thermal Units
CAV	Constant Air Volume type of HVAC
CCTV	Closed-Circuit Television
CEMA	Comfort and Energy Management Algorithms
CFM	cubic feet per minute
CoP	Coefficient of Performance
CSV	Comma Separated Values
DHC	District Heating and Cooling
GUI	Graphical User Interface
HCI	Host Controller Interface
HVAC	Heating, Ventilating, and Air Conditioning
IDE	Integrated Development Environment
ISM	Industrial, Scientific, and Medical (Frequency Band)

MS/TP	Master/Slave Token Passing Network
PI	Proportional Integral controllers
PMV	Predicted Mean Vote
QoS	Quality of Service
SEER	Seasonal Energy Efficiency Ratio
VAV	Variable Air Volume type of HVAC
WSN	Wireless Sensor Network

Chapter 1: Introduction

1.1 Background

1.1.1 Wireless Sensor Networks

A typical WSN architecture includes intelligent end-nodes with built-in communication modules [**Mah07**], in addition to sink nodes in each transmission range area if the WSN is especially large, and a server to collect and/or parse the information given. The exact components of a WSN depend on the protocol used, but the above-stated components are elements in almost all WSNs. Examples of WSNs deserve a thorough literature review in and of themselves, but for the sake of this discussion examples may be given: [I] Lighting Control, [**Vip05**], [II] Pet Management Systems [**Kim07**], [III] Greenhouse and general agricultural monitoring, [**Goh07**] [**Lee08**] [**Yim07**], [IV] Intelligent Closed-Circuit Television (CCTV) Networks [**Sha07**], [V] Industrial Automation [**Yam08**] [**Zhe06**], [VI] and finally for communication in the battlefield [**Als08**].

WSNs, in their simplest form, have many inherit challenges and issues that need to be dealt with. For example, the following references deal with advances in routing protocols and Quality Of Service (QoS) [**Man01**] [**Zha07**] [**Kou07**] [**Chu07**] [**Uma**] [**Mac08**] [**Krc07**] [**Tah09**] [**Pan**]. The references are devoted to the creation and advancing of WSN routing protocols that take into account the inherent properties of WSNs. In addition, other literature has been devoted to the area of QoS in WSN; that is, allowing the WSN to support delay-critical applications, among many other examples. Another example is security in WSNs. The following references are examples of papers discussing this issue: [**Wan08**] [**Zha08**] [**Zia06**]. The nature of the WSN in

and of itself allows for huge gaps in security. These gaps include the ease of physically tampering with the end and/or sink nodes, tampering with the communication protocol and/or cracking the encryption code typically used, among other examples of security breaches. The referenced papers, among many others, discuss methods and additional protocols used to safeguard against these potential security breaches.

1.1.2 Smart Homes

The second background topic to be discussed is in the area of smart homes. In traditional homes, the occupant(s) controls the day-to-day activities of the residence; the amount and expiry date of food in the fridge, the lighting, the heating, cooling, and the operation of the Heating, Cooling, and Air-Conditioning system (HVAC), etc. Of course, this traditional method has its advantages and disadvantages. The advantages are obvious and self-explanatory; the disadvantages, however, warrant a discussion. Humans are not computers, and thus it is impossible for humans to accurately and efficiently achieve the optimum energy efficiency regarding their use of energy-intensive appliances. In addition, keeping track of all aspects of the residence together– food, security, cleanliness, energy consumption, etc, is very time-consuming and burdensome. Also, in case of emergencies, occupants in a residence will be much slower to implement emergency response protocols than, for example, a central residence automated system.

Thus, the need arose for residences with a brain, i.e. smart homes. The term is very loose and incorporates many types. This is a new and ongoing research topic and incorporates many different facets, such as smart homes with RFID tags [**Lee06**] [**Dar08**], smart homes that employ information fusion [**Lit07**], smart homes that employ an underlying IP Network [**Gil07**]. Generally though, a smart home is a residence where a central computer offloads many mundane day-to-day tasks from the occupants, in addition to providing critical care in case of emergencies [**Tam06**] [**Nat09**]. The difference between Smart Homes is the tasks offloaded and the type of critical care provided. There is extensive literature on the network that forms the basis of these smart homes [**Che08**] [**Mor93**] [**Kas05**] [**Gil07**] [**Beg09**] [**Pap09**], in addition to integrating these Smart Homes with the Web [**Lid07**].

Examples of smart homes abound but three types of applications will be discussed here. The first example of a smart home is one which assists the elderly in living a normal life. As is well known, elderly people sometimes pose a serious risk to themselves when living all alone in a single-family dwelling. This is because they can fall off the stairs, they often tend to forget, and they find difficulty in buying groceries for themselves, as examples of the difficulties they face. Smart Homes can especially alleviate these difficulties by performing these tasks for these occupants. Studies and research papers delved into many subtopics, and all were essential for the area of elderly care. These papers include ones about how to allow the system to learn the occupants' habits [**Cam06**], using motion sensors to track the occupants in the house including the elderly [**Oud06**], using fiber optics [**Foo06**], and using audio sensors [**McH06**]. In addition, smart homes have been tailor-made for dealing specifically with emergencies for the elderly [**Tam06**] and for dealing with quadriplegia patients [**Uzu07**].

The second example of a smart home is a home rigged to remotely monitor the general health vitals of its occupants, also known as Tele-Health. Remote health care is generally defined to be health-care professionals monitoring the vital signs of certain types of patients remotely, and acting accordingly, including in emergencies. Many different techniques have been employed to achieve this goal, from sensors and Smart-Phones [**Gay06**], detection of drifts and outliers [**Jai06**], to using a novel system called Data Management System, [**ODo06**]. In addition, architectural-based frameworks and frameworks for urban environments have been discussed [**Val06**] [**Mor07**].

The third and final example of a smart home is one where energy consumption is monitored. For instance, the province of Ontario is implementing smart meters that tracks occupants' electricity usage throughout the day and places different prices throughout the day and during holidays, so as to nudge people manually to move their heavy electrical usage during off-peak periods, both to trim costs and to ease the pressure off the electrical grid [**Sma09**]. Smart meters are not technically part of a smart home, but nonetheless is an important aspect of it. Other examples include controlling the lighting of a smart structure [**Vip05**], and a next-generation standalone HVAC system [**Tak09**].

1.2 Thesis Motivations and Objectives

Energy management while achieving comfort control is of a top priority if a country is to try and control its energy consumption. As will be seen in the next chapter, a lot of literature has been written about the topic of energy management in homes, and about comfort control in homes, separately. As of the writing of this thesis, very little literature exists on the combination of the two concepts; devising a central energy management control system which is responsible for optimizing energy use for *existing* residential spaces without changing any of its underlying systems.

Simply put, the motivation of this thesis is the development of a simple, cheap, easy-to-install, and efficient central energy-management system for residential spaces. This system controls everything and anything that consumed energy in a residential space; from the HVAC system to the lighting system, and so on.

Understandably, this is an enormous undertaking that will require years of research and implementations. Thus, this thesis is to concentrate on only two aspects of this energy management system, controlling the energy consumption of the HVAC and the lighting systems. It has been shown that these two systems are the biggest consumers of energy in any structure, including the residential types [**Tak09**].

Thus, the objectives of this thesis are two-fold: the first objective is to prove qualitatively that a comfort and energy management control system based on WSNs is feasible and realistic; that is, it actually achieves the required comfort control. For this thesis, the concept of comfort control is restricted to the areas of HVAC and lighting control. The second objective is to prove quantitatively, in the form of monetary savings, that this comfort control system outperforms a traditional residential dwelling with no energy-saving measures.

1.3 Thesis Contributions

The first and main contribution of this thesis is the group of CEMA algorithms that allow, when executed, a residential space to economically achieve comfort control. The other contribution is a Java-based, modular, and easily expandable discrete-event simulator to test the different CEMA algorithm modules. These modules have certain inputs and bring forth outputs that satisfy the main goal of this project, which is economical and feasible comfort control. The modules represent different energy-consuming aspects of a typical residential area. We have also designed two modules to be added to our Java-based discrete-event simulator; a module that describe the CEMA algorithm for controlling a typical HVAC system *for each heating/cooling area* (unlike typical HVACs which control whole floors), and a module that describes the CEMA algorithm for controlling the indoor light intensities so as to be maintained at a constant level.

1.4 Thesis Outline

The rest of this thesis is written as follows. Chapter 2 discusses previous literature on the subject of overall comfort control in residential areas, in addition to a discussion on the network protocols that support wireless comfort control schemes. Chapter 3 discusses the assumptions, equations, and mechanical engineering principals CEMA is based on, then presents the algorithms themselves. Chapter 3 also discusses the method to test the CEMA algorithms; i.e.: the detailed workings of the standalone simulator, the JAVA language it was written by, the classes, the main event queue, and the Graphical User Interface (GUI). Chapter 4 presents the ensuing results of the program, given the course of action outlined in the abstract of this thesis. This discussion includes extensive screen snapshots of the program, any entailing intermediate output files, and graphs that show and prove that the two above-stated objectives were met. Chapter 5 concludes the thesis and presents the final remarks on the issue at hand, and provides areas of further interest and further research.

Chapter 2: Comfort Control and Energy Management in Smart Homes – State of the Art

2.1 Introduction

The aim of this literature review is to present the relevant literature pertaining to the main objective of this thesis; i.e.: to present literature describing past attempts at residential comfort control. In addition, literature that describes the underlying network that supports a smart home is also included. Each paper being reviewed is critically analyzed and its strengths and weaknesses, as well relevance to this thesis, highlighted.

An important note is that very little literature that combines energy management and comfort control for residential spaces, exist. Literature exists for each of the two issues at hand though, as will be shown in this chapter.

2.2 Comfort Control and Energy Management

The first comfort control technology surveyed here is a next-generation HVAC system without the need for WSNs [**Tak09**]. The paper applies this HVAC system to office buildings, but nonetheless this system can also be downsized and used for residential spaces. This paper delves into the inner workings of the traditional HVAC system and modifies them to be 40% more efficient in terms of CO_2 emissions. This modification is numerically proven by 3D graphs that

show that comfort has been reached (comfort being mathematically defined in the same paper [**Tak09**]).

The authors of [**Tak09**] then define the traditional HVAC system as being one of two types: the Constant Air Volume (CAV) and the Variable Air Volume (VAV) types. Both deal with the heat load of the room in different ways. CAV decreases the heat load via a constant air volume per unit time. This is in accordance to the well-known psychometric equations.VAV decreases the heat load of a room by varying the incoming air volume while keeping temperature constant. The authors of [**Tak09**] propose to use a modified VAV system; one which has the basics of VAV but which also varies air temperature. The proposed system is based on the optimization of already existing documented comfort equations given in their fourth source [**Tak09**]. Their contribution is the simplification of these extreme complex non-linear energy consumption equations, which is needed in order to test whether their optimization is viable or not. In addition, the authors of [**Tak09**] use, as an index to whether their system is achieving comfort, the result of the Fanger equations of [**ASH81**]. This result is called the Predicted Mean Vote (PMV), and is defined mathematically in [**ASH81**], [**Tak09**]. In addition, the authors show, from [**ASH81**] the area in the psychometric chart (whose purpose is explained in section 3.1.5) where comfort is defined [**Tak09**]. The authors then attempt to describe, mathematically, the relationship between realistic energy consumption (with respect to chilled water, electricity and HVAC pump usage), PMV, and heat balances. These complex groups of equations (which are impossible to solve using brute-force methods) are shown in [**Tak09**]**.** Figure 2.1 shows the authors' first major approximation: the division of office spaces into separate heating areas which are independent of each other with respect to heating loads [**Tak09**]:

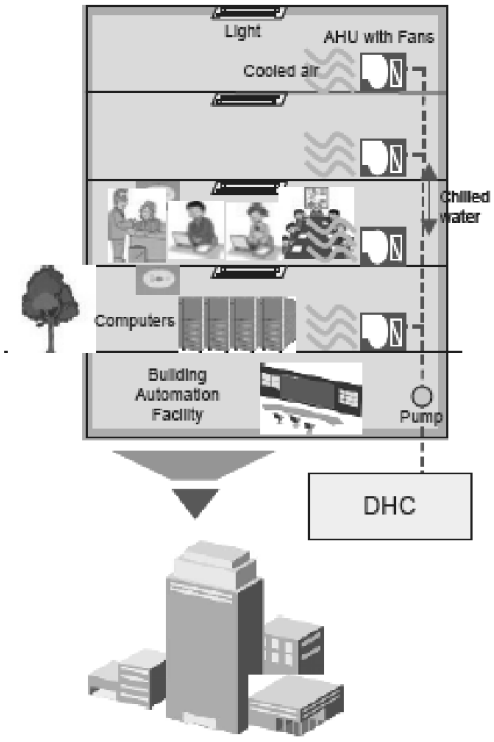


Figure 2.1: DHC architecture in an office building

As can be seen here in this example, each floor is independent of the other in terms of the heating and cooling required. Each zone is called a district heating and cooling District Heating and Cooling (DHC). As can also be seen, the key to the optimization of this system are the Air Handling Units (AHU) units whose fans allow the air to pass by the chilled water in order to lower its temperature (or pass by the warm water to increase its temperature in winter months). The DHCs are independent of one another because the walls are insulated enough to render them independent of one another [**Tak09**].

The authors attempt to simplify the above-stated complex equations to something that is more solvable. The authors, without elaborating, have simplified the main function using iterative driving functions called Spline-Functions [**Tak09**]. The authors then present the results of their simulation using a set of inputs defined in their paper and present the results in the form of informative 3D graphs which link outdoor air temperature, outdoor relative humidity, ambient temperature, and PMV. These graphs prove that the most economical way of achieving comfort is taking outdoor air and cooling it via the AHU, instead of re-circulating all or part of the air.

Finally, using the derived iterative functions, the approximate energy consumed for both the traditional and new HVAC systems is presented side-by-side. Figure 2.2 shows that the new HVAC system consumes less energy:

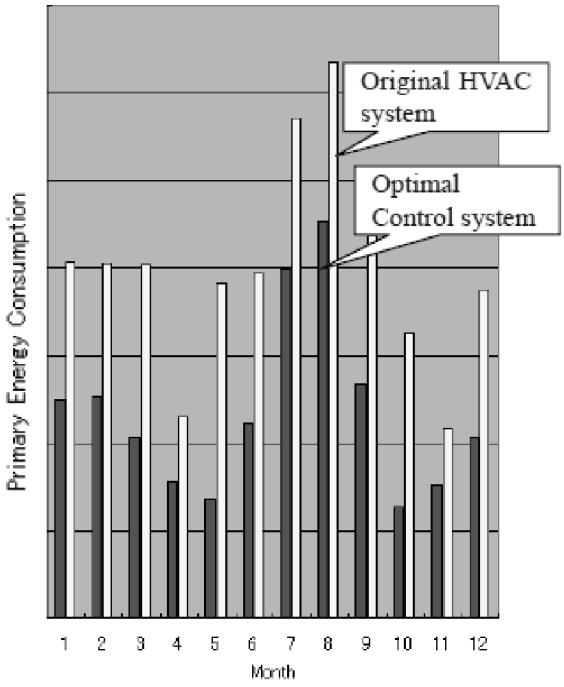


Figure 2.2: Energy consumption for both HVAC systems

This paper is of importance to our work, as the utilization of independent heating/cooling zones is adopted in this thesis, in addition to the concept of measuring comfort quantitatively [**Tak09**]. However, there are also major differences between reference [**Tak09**] and this thesis. In [**Tak09**], the authors delve deep into the inner workings of the HVAC system, and alter it significantly. However, trying to implement these changes to existing HVAC systems in homes would be prohibitively expensive. Instead, the goal of this thesis is to simulate a plug and play situation; a wireless sensor network that will be deployed over existing residential HVAC systems. The intent of this thesis is to prove that the expected system to be deployed improves the energy consumption of an existing, operational HVAC system. In [**Tak09**], the authors do not mention where the traditional HVAC energy consumption values in Figure 2.2 come from, in addition to that no mention of the working of the simulation is given, as some examples of perceived missing links in this paper. It is important to note as of the writing of this thesis, most

papers talking about improving HVAC performances in general (whether for office or residential spaces) speak about changing the inside workings of the HVAC controller, by adding Proportional Integral (PI) controllers, for example, or making the AHU more efficient [**Kom09**] [**And08**], instead of using existing HVAC systems as a black box and making its operation more efficient. Finally, the authors of [**Tak09**] do mention the inclusion of lighting control into their new smart buildings, but do not elaborate.

The next paper does not speak about comfort control as in the past paper, but talks about conserving energy through efficient lighting [**Men**]. It is important to state that most other lighting controls utilize some form of network for their control [**Vip05**] [**Par07**]; obviously lighting cannot control itself and it is prohibitively expensive to reconfigure the wiring inside a functioning structure.

There are some similarities between the main idea of reference [**Men**] and the light control algorithm presented by this thesis. This paper starts with an all-important diagram to show how the authors prepare to tackle the lighting issue in an open space office building floor:

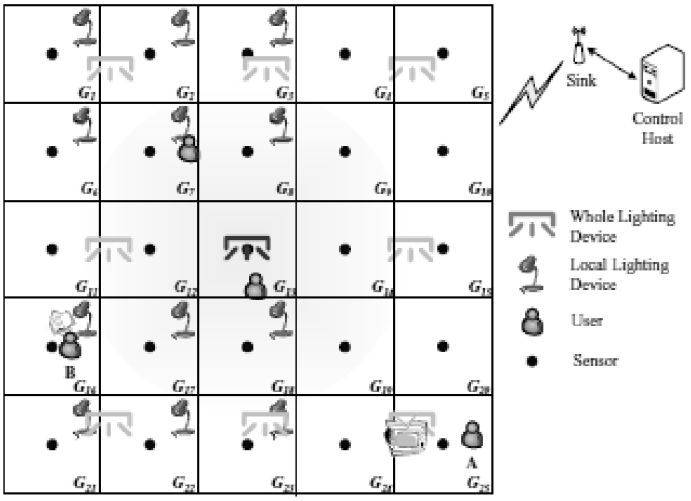


Figure 2.3: Lighting plan in the proposed office space

As can be seen here, the lighting plan of the open-area office space is divided as follows: the area is divided into a grid with evenly spaced squares, with a light sensor square in the middle of each square. There are two types of lighting: major overhead lighting, which when activated lights up its square and the squares around it as seen in Figure 2.3, and minor lighting, which lights up

only the square that lighting is in. This obviously models major overhead lighting and desk lamps in a typical office setting [**Men**].The sensors send their data to a sink which is connected to the control host; i.e.: the brain of the program

The authors of [**Men**] designed an innovative lighting control and device control algorithms that form the backbone of this lighting control system. To understand these two algorithms, one must understand the general architecture of the system as a whole, and the flowchart defining the outputs needed from these two algorithms. Figures 2.4 and 2.5 respectively show that.

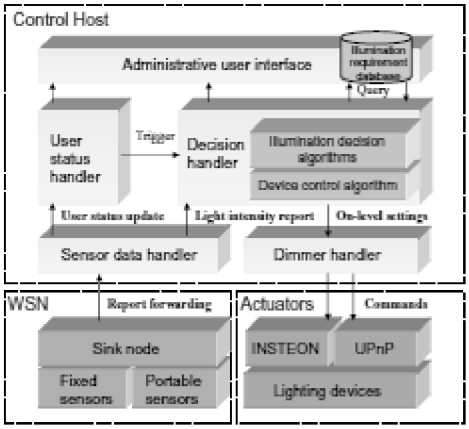


Figure 2.4: System components of the proposed lighting control

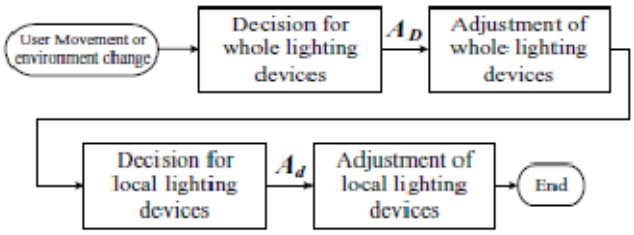


Figure 2.5: System architecture

From Figure 2.4, we can see how the system works in general; there are portable nodes attached to the users (or attached in a way that senses the movements of the user), and there are fixed, light-intensity measuring nodes in the middle of each square in the grid shown in Figure 2.3. This system is based upon two types of criteria: Users' different lighting requests and trying to match between them the best way possible, and the conservation of energy. This is the reason behind the decision handler module in Figure 2.5. The rest of the modules are thus easy to

explain: The sensor data handler sends the appropriate parsed sensor data to either the user status handler or the light intensity handler modules. The User status handler handles how the users' movements affect the lighting (i.e.: user moved into a new area which was dark and now must light up for that user), and stores the movements of the user in a database. At the same time, if and when these movements require lighting changes, a trigger is sent to the Decision Handler Module. This module takes in the triggers, if any, and the light intensity sensor data as input, then adjusts the major and minor lights according to both the users' requests and the principle of energy conservation. Figure 2.4 shows how the Decision Handler Module proceeds to do the light intensity adjustment. An algorithm is run to decide on the light intensity of the major lighting devices, then the module proceeds to issue the commands to the actuators in Figure 2.4 to dim the needed lights, if any. The same two procedures then occur for the minor lights. This is an innovative trick as the major lighting covers many squares in the grid, then, if needed, minor lights can be shut off or dimmed. This procedure is mainly used to conserve energy [**Men**].

The rest of this paper is a complex mathematical analysis of the two light-intensity changing algorithms, of which the basics will be shown and explained here. First, the authors of [**Men**] state the following preliminary data and assumptions. This system has k grids, n mobile users with sensors attached to them, m major lighting devices, and m' minor lighting devices. The n mobile users inform the control host (in Figure 2.4) of their current actions (such as working, eating, etc.) and via an undefined (in [**Men**]) localization algorithm, the control host knows in which square each user is. Each square in Figure 2.4 is denoted by G_i, $1 \leq i \geq k$. In each square, there is a fixed light intensity sensor f_i, $1 \leq i \geq k$, and each portable user has a sensor p_j, $1 \leq j \geq n$. The light sources are denoted D_i to D_m and d_i to $d_{m'}$ to denote major and minor lighting sources respectively. The sensor f_i nearest to D_i is denoted by $f_{c(Di)}$. Since all users are mobile, to denote the nearest portable sensor to the user a bound function $bound(u_j)$ is required. This function bounds at most one minor lighting device to one user. That is, when one user moves in the vicinity of at most one minor lighting device, this minor device may or may not be triggered. This is independent from the triggering of the major lighting devices. The light intensities captured by f_i and p_j are denoted by s(f_i) and s(p_j) respectively. The authors state, accurately, that there are many sources for the light intensities captured by these sensors. Thus, the authors of [**Men**] denote the contributions of the major lighting in square I and/or the minor lighting in s(f_i) and s(p_j) as l(D_i) and l(d_i) respectively. The reference [**Men**]draws from its fourteenth source the

fact that light intensities are additive, and according to Figure 2.6 below, the normalized degradation of light intensity is constant whatever the intensity of the light. Note that these readings are $l(f_{c(Di)})$ versus the horizontal distance from D_i

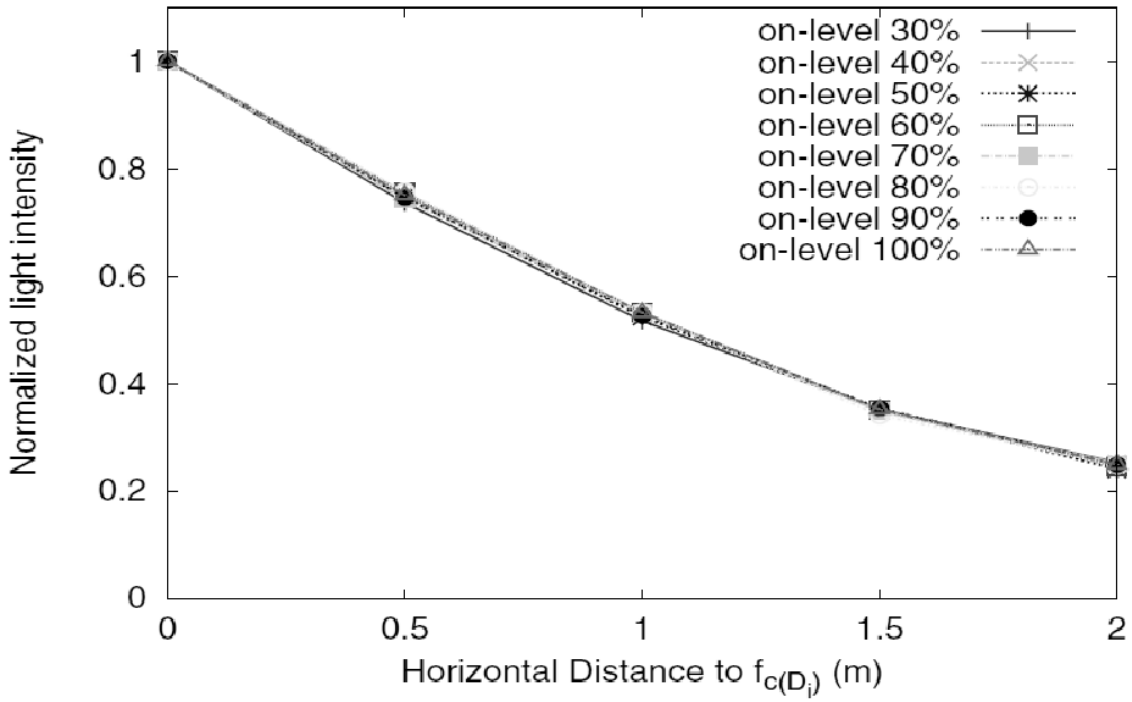


Figure 2.6 : Normalized light intensity degradation

The lighting from the minor lights is assumed to not affect the readings of the major sensors. Also, it is assumed that there are no obstacles between the major lights D_i and the major sensors f_i. Finally, it is assumed that one minor lighting device is sufficient for its user. As stated before in this chapter, light intensities are additive. Thus, the authors of [**Men**] state that the effect of D_i on $l(f_{c(Di)})$ is 1 and the rest can be expressed as weighted factors w_j^i, where $0 \leq w_j^i \leq 1$, and all of w across i and j are expressed in the following matrix:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w_1^1} & \cdots & \mathbf{w_1^m} \\ \vdots & \ddots & \vdots \\ \mathbf{w_k^1} & \cdots & \mathbf{w_k^m} \end{bmatrix} \quad (\mathbf{2.1})$$

Taking into account that light intensities are additive, the authors assume that light intensity in general will come from three sources; sunlight, D_i, and the other lighting devices in the vicinity [**Men**]. Thus, the total light intensity measured in all of the fixed sensors can be expressed as follows:

$$S_f = W.L_D + S_{SUN} \quad (2.2)$$

Where S_f, S_{SUN}, and L_D are *k*1* vectors representing the total light intensity measured, the effect of sunlight measured on each sensor and the light intensity contributed by each lighting device. As was stated at the beginning of discussing this paper, the goal of this part of the mathematical analysis is to try and find an expression for L_D and L_d, so that the decision handler algorithms can proceed correctly. The above stated can also be expressed as follows [**Men**]:

$$L_D = (S_f - S_{SUN}) . \widehat{W^{-1}} \quad (2.3)$$

S_f can be measured at runtime, S_{SUN} can be measured during the deployment phase, and W can be measured during the deployment phase with the blinds covering any effect of sunlight. Thus, there is a mathematical method to compute L_D. Similarly, the calculation of L_d is straightforward [**Men**]. It can be expressed as [**Men**]:

$$L_d = S(p_k) - S(f_j) \quad (2.4)$$

Where the three terms are the light intensity contributed by the minor lighting devices, the total light intensity measured from the portable sensor, and the total light intensity measured respectively. The above equation basically means that the light intensity contributed by the minor light device in a certain square of the grid is the difference between the total light intensity measured from the portable sensor and the light intensity measured by the fixed light sensor.

This is of course assuming a user exists within the square and is bound to the minor lighting device.

After discussing the above points and receiving a mathematical formula for L_D and L_d, that is, the light intensities contributed by the major and minor lighting devices respectively, the authors of [**Men**] then discuss a method for obtaining the changes that should be made by their system to each major and minor lighting device. These light intensity changes are denoted by A_D and A_d, respectively. These values are also m*1 and $\acute{m}$*1 vectors, respectively; the first variable describes light intensity changes required for major lighting devices

$$A_D = [a(D_1), \quad a(D_2), \quad \dots, \quad a(D_m)]^T \tag{2.5}$$

The second variable describes light intensity changes required for minor lighting devices

$$A_d = [a(d_1), \quad a(d_2), \quad \dots, \quad a(d_{m'})]^T \tag{2.6}$$

In order to achieve that, the authors of [**Men**] follow the following steps:

- Each user of the system u_i has certain requirements as to the major and minor lighting needed, if any. These requirements include the lighting range requirements for the major and minor lighting devices in the grid the user is occupying. This is assuming that the minor lighting device always compensates for the difference in light intensity between that needed by user u_i and that offered by the major light source the user is in. These ranges for major and minor lighting device illumination requirements can be described as follows [**Men**]:

$$\left[B_D^l(u_i), B_D^u(u_i)\right] \tag{2.7}$$

$$\left[B_d^l(u_i), B_d^u(u_i)\right] \tag{2.8}$$

The terms l and u denote the lower and upper bounds, respectively.

- The lighting effect of the major light source of one grid square on the other adjacent squares must also be taken into account. This, in addition to the W matrix discussed before, will help accurately satisfy the requirements of the user. This new matrix is called R_i, which is a k*1 matrix. $r_i(G_j)$ is 1 if square G_j is expected to receive lighting within the major lighting requirement bounds of user u_i, as discussed in the preceding point. For illustrative purposes for their paper, they define X_m to be a m*1 vertical vector whose values are all 1, and $\overline{R_\iota}$ to be the following:

$$\overline{R_\iota} = \begin{bmatrix} r_i(G_1) & 0 & 0 \\ 0 & r_i(G_2) & 0 \\ 0 & 0 & r_i(G_k) \end{bmatrix} \tag{2.9}$$

The requirements are then defined (satisfying user requirements mathematically in the most economic way possible) mathematically. Equation (2.10) describes the first expression to be solved forA_D. Equation (2.11) describes the second expression to be solved forA_d. Equations (2.12) and (2.13) describe the two constraints used in solving forA_D, and Equation (2.14) describes the same forA_d.

$$X_m(A_D + L_D) \tag{2.10}$$

$$X_{m'}(A_d + L_d) \tag{2.11}$$

$$B_D^l R_i \leq \overline{R_\iota}\left(S_f + WA_D\right) \leq B_D^u(u_i)R_i, \qquad \forall i \in [1,n] \tag{2.12}$$

$$0 \leq A_D + L_D \leq L_D^{max} \tag{2.13}$$

$$B_d^l(u_i) \leq a(d_j) + s(p_i) \leq B_d^u(u_i), if\ bound(u_i) = j, \forall i \in [1, n] \tag{2.14}$$

Equations (2.10) and (2.11) describe the concept of conservation of energy; that is, the changes to the illuminations made by the major and minor lighting devices respectively should be the minimum changes possible to conserve electrical energy. Equations (2.12) and (2.14) are similar in that both describe that any changes made to major and minor light devices, respectively, to be within the user-set requirements. The difference is that in Equation (2.12), the changes in lighting devices are subject to the matrix W to include the effect of degrading light intensity away from its source, and subject to $\overline{R_t}$ to include the effect of that a major lighting device also illuminates adjacent squares. Equation (2.14) does not have such restrictions as it is assumed each minor lighting device is independent of the other. Equation (2.13) simply states that any change made to the major lighting device is to be within the maximum limit of the device itself.

The above-stated equations by themselves are very complex to solve due to the nature of the equations. Thus, the authors of [**Men**] propose a simplification; since minor lighting device changes are not included in major lighting device calculations (vice versa is not true however) A_D can be solved for first (Equations (2.10), (2.12) and (2.13)), and *then* A_d is independently obtained (Equations (2.11), and (2.14)), taking into account any and/or all changes made by A_D. A_D is solvable by using the Simplex method [**Men**], and A_d is simply the difference between the lower bound user requirement and the mobile sensor reading, after A_d is computed and executed. However, a last problem must be tackled; what happens if user requirements are in conflict with one another? The authors of [**Men**] prove it is impossible to solve this issue as it stands, so they propose that the user requirements be relaxed gradually until a feasible solution for A_D can be calculated. This relaxation is done in discrete steps of course. The algorithm to solve for A_D and then A_d is as follows:

- For each square G_i, check and see which users are within the vicinity of its major lighting device. For each of these users, check and see if the requirements of the user overlap with

the sensed light intensity values. If not, then technically the user is not within the vicinity of the major light source (because the requirements of the user were not satisfied), and thus R_i is updated with $r_{user}(G_i) = 0$.

- For each square, check and see whether there are common requirements for the subset of users stated in the previous point after the adjustments. If there are no common requirements, find the minimum requirements that satisfy all this subset of users. For each user in this subset, check whether the requirements of this user do not overlap with the minimum requirements; if not, set $r_{user}(G_i) = 0$ for this user.

- Solve Equations (2.10), (2.12), and (2.13) for A_D. If no feasible solution is still found, widen the lighting requirement range for each user in the above stated subset by a pre-defined constant α. Then repeat this step again.

- Finally, after a feasible A_D is found, pass through each minor lighting device. If there is a user bound to it, set this minor lighting device according to Equation (2.4). If not, set it to 0.

Finally, the device control algorithm module is delved into, and it is briefly stated that this module takes A_D and A_d from the Illumination Control Module, and then gives instructions to the actuators unto which lights to change and by what amount. These changes are of course finite and are done in iterations until the new light intensity is within range of the new target light intensity, by a pre-defined amount β. In addition, the authors then delve into the details of the implementation they used to test their algorithms [**Men**], including a Java program to implement the illumination decision and device control modules.

This paper is extremely invaluable to this thesis for many reasons. First of all, it utilizes wireless sensor networks for data collection and parsing, which is the goal of this thesis. Second, it tackles goals which are more complex than the requirements of this thesis with regards to lighting control. For example, in this thesis, the lighting in the rooms is separate and independent of one another, yet there is input from sunlight and has to be taken into account. This means that

most of the mathematical analysis employed in this paper is not needed for the purposes of this thesis. In addition, in this thesis, lighting from minor lighting devices, if any, is ignored. Nevertheless, this paper is important in that it states the devices needed to implement the various modules of the system.

2.4 Wireless Sensor Networks for Smart Structures

The last section of this literature review deals with the issue of the network used to support the CEMA algorithms. The choice of wireless sensor networks has been explained elsewhere in this thesis; but is being reiterated here: the use of WSNs is needed for low costs, so as to be able to incorporate it in existing Canadian homes. This literature review explains the choice for the networking protocol used. Three main types of communication protocols for smart buildings are discussed here: Bluetooth [**Beg09**], a novel protocol called EPIS [**Pen09**], and Zigbee [**Lee08**] [**Pen09**]. A representative paper from each category is provided and a critical analysis is presented.

Reference [**Beg09**], is about implementing two mundane smart structure functions; garage door opener, and an electronic attendance system. It first states the specifications and advantages of using Bluetooth instead of other wireless network protocols. These specifications and advantages are as follows [**Beg09**]:

- Bluetooth employs low power at operation and a low cost.

- Bluetooth transmits around the 2.4 GHz band, which is in the Industrial, Scientific, and Medical (ISM) band and thus requires no license.

- The range of transmission for an acceptable Bit Error Rate (BER) is around 10 meters.

- Bluetooth employs the frequency hopping technique, hopping between frequencies in the 2.402 and 2.480 GHz range with the rate of 1600 hops/sec.

- Bluetooth networks are defined by a minimum of two units in the network; the master, and the slave, in the form of a star network. This unit is called a piconet. The maximum size of the piconet is one master and seven slaves. A Bluetooth unit can be a slave in different piconets or be a slave in one and a master in another. This enables piconets to be interconnected.

- There are two types of Bluetooth connections; synchronous Bluetooth connections suitable for voice applications, and asynchronous Bluetooth connections, for time-insensitive applications.

- Bluetooth protocol utilizes the Host Controller Interface (HCI) protocol to form and disseminate data packets.

The authors of [**Beg09**] tackle details about HCI and Bluetooth that are beyond the scope of this literature review, and are too detailed for this thesis.

The authors then tackle the first application in which they use Bluetooth; the automatic attendance system. Note that BDAADR (see Figure 2.7) is the equivalent of the MAC address in the TCP/IP stack; it is the physical address of the Bluetooth device [**Beg09**]:

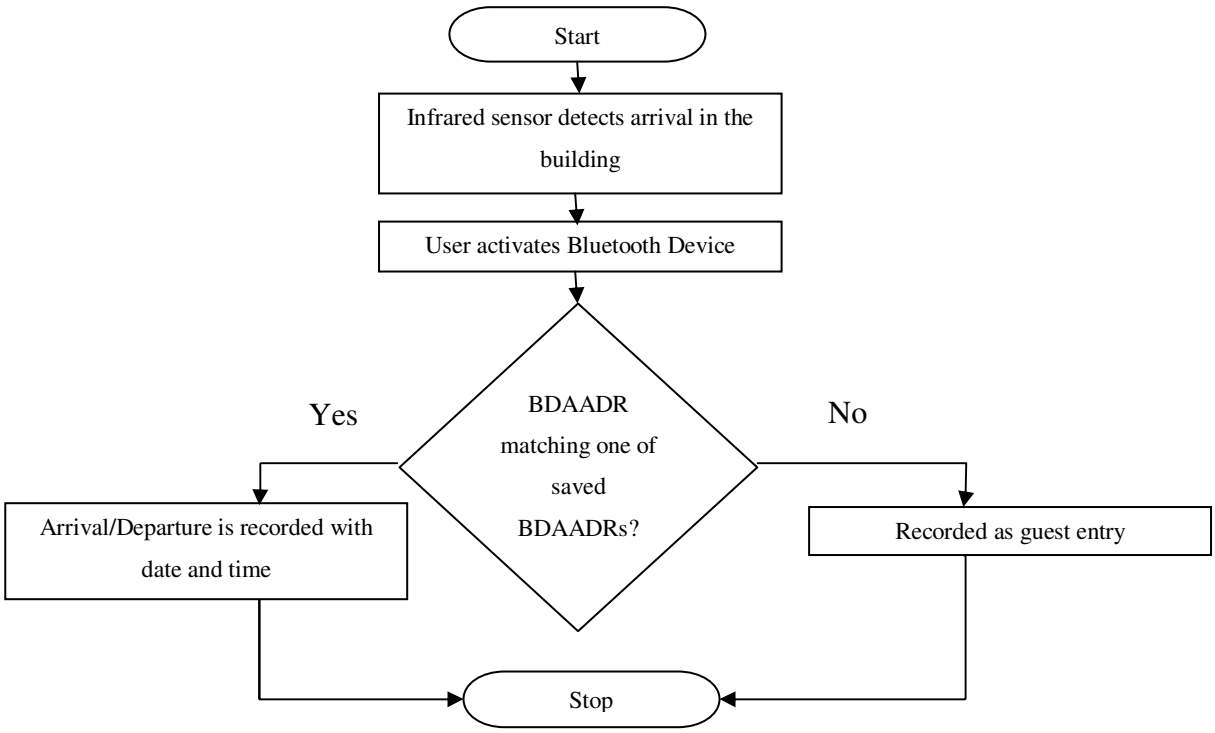


Figure 2.7: The Flowchart for Automatic Attendance System

Basically, if the user is entering or leaving the building, the system records the user name and the BDAADR address along with the date and time. If the BDADDR is not in the system, the user is recorded as a guest. Figure 2.8 shows the flowchart for the automatic garage door opener [**Beg09**]:

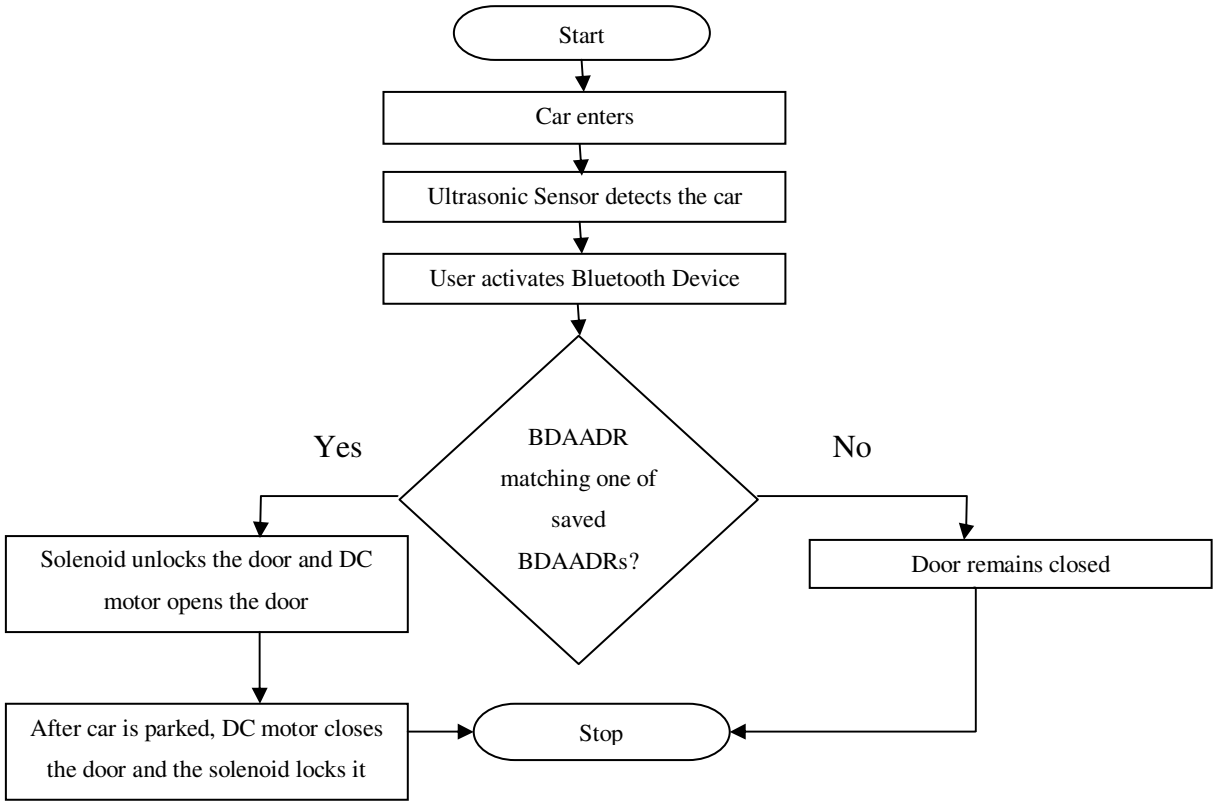


Figure 2.8: Flowchart of the automatic garage door opener

When the ultrasonic sensor detects the car, and Bluetooth is enabled, the system checks the BDAADR address to see if they match what is in the system. If the addresses do match, the system activates actuators that allow the user to park the car. If not, the door remains closed.

The system flowcharts are simple enough as is and they seem to satisfy the two applications shown (no implementation is given). However, the focus of this part of the literature review is the discussion of the protocols used for the WSNs. A comparison between the three past-mentioned protocols is given after the three types of protocols are discussed.

The next two representative papers discuss the concept of using Zigbee in WSNs. Reference [**Lee08**] talks about using Zigbee-based WSNs to control the climate in glasshouses. The authors

of [**Lee08**] speak about using Zigbee for its ultra-low power consumption, its ease of installation and its minimal need for manual setting up. Then, the authors discuss the different methods of routing in Zigbee, and state that there are three types of Zigbee devices; Zigbee end devices (the sensors), Zigbee routers (which act as cluster heads), and Zigbee middleware that connects the two [**Zig06**]. The routing method selected by the authors is the static routing method, since for this particular application, no complications in the WSN routing protocol was necessary. There has been no need to implement such protocols as LEACH, which elects a random cluster head and can thus be inefficient, or SPIN, which causes huge overhead to the system due to the repeated retransmissions due to the flooding nature of the SPIN protocol [**Lee08**]. Thus, the authors of this paper elect a direct one-hop connection between the end devices, sink nodes and the Zigbee router, as is shown in Figure 2.9, where each vertical rectangle is a greenhouse, each dark colored node is a Zigbee end device (a sensor), each light colored node is Zigbee middleware, and the two devices beneath the glasshouse illustrations are the Zigbee routers:

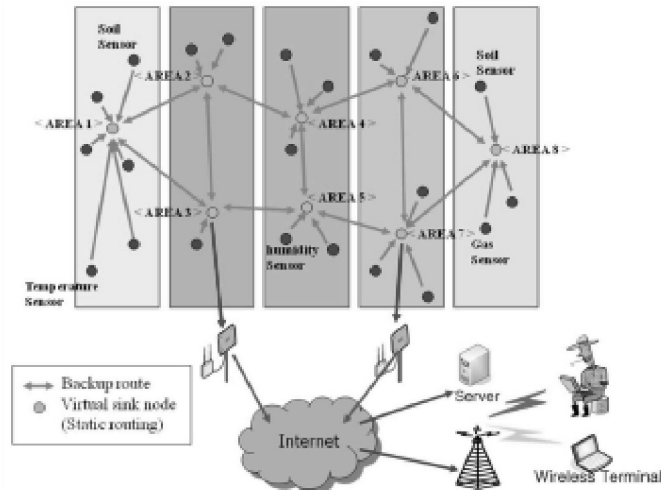


Figure 2.9: WSN architecture for entire glass house complex

As can be seen here, the authors of [**Lee08**] discard the more complex routing protocols and opt for direct static routing. Finally, the authors implement the system in actual glass houses, but no mention of the results are given

This goal of reference [**Pen09**] is to provide energy efficient Wireless Sensor Network for the Smart Home Environment; exactly the kind of communication network required for the CEMA algorithms. Reference [**Pen09**] for analyses four different WSN protocols, including a novel type

the authors are touting, called EPIS. While performing this comparison, the authors of [**Pen09**]state that Bluetooth employs a maximum of 8 units per piconet, which puts it at a significant disadvantage to Zigbee and EPIS. Also, Bluetooth has a high power consumption rate [**Pen09**], while EPIS and Zigbee share the concept of nodes sleeping until they are to transmit and/or receive, resulting in much longer useful lifetimes [**Pen09**] [**Zig06**]. In addition, Bluetooth does not formally define ways of interaction between service users and providers, while Zigbee has pre-defined methods of interaction. EPIS, on the other hand, has lighter interaction protocols that are just as effective as Zigbee [**Pen09**], making them ideal for smart homes. While this is true, it is obvious the EPIS protocol is still lab-work that needs to be refined, while Zigbee is an industry-tested protocol that has proven its worth.

Finally, an important communication protocol standard for building automation and control is called BACnet, short for Building Automation and Control Network [**DeC10**]. This communication protocol (ISO standard number 16484-5) was initiated by ASHARE in 1995 [**Mik00**], and updated rapidly since then [**DeC10**]. It was set to be used for wired communications and not wireless communications, initially. The main advantage of this group of protocols is that multiple vendors with multiple sensors or multiple control systems (fire alarm , lighting, HVAC, etc.) could work together seamlessly under the auspices of one major communication protocol. Thus, the discussion of this protocol is of importance to this thesis.

This protocol encompasses all layers of the OSI stack, yet includes versatility to allow multiple data link and network layers. However, this protocol has its own network protocols as well; ARCNET, the Master-Slave Token Passing Network (MS/TP), BACnetoverIP, and LonTalk. Figure 2.10 shows a typical architecture for a building network based on BACnet [**Mik00**]:

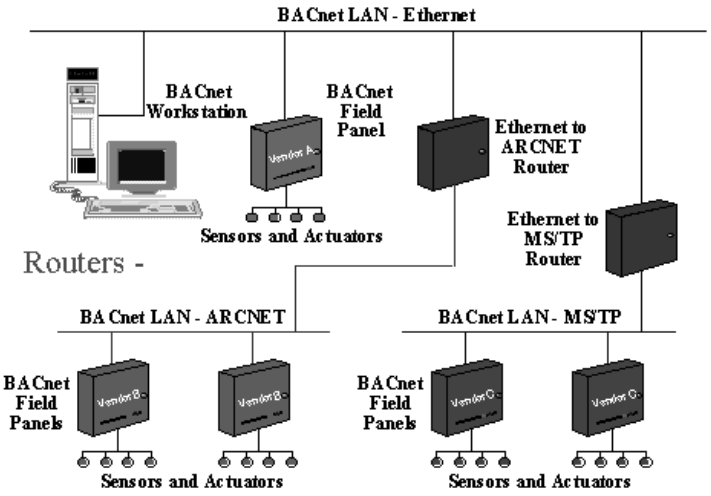


Figure 2.10: Typical BACnet architecture

As can be seen, the network protocol that is used to connect the heterogeneous sub-networks is Ethernet, but this is not a rule and can be any other protocol. What is important is that there must be gateways in place to translate any packets being sent or received between the sub-networks.

Another important standard feature of BACnet is the fact that BACnet devices and routers can communicate with each other directly and remotely over the Internet; either through IP tunneling (the Internet is transparent to the BACnet devices and routers), or through BACnet routers that communicate directly via IP, as is seen from Figure 2.11 [**Mik00**]:

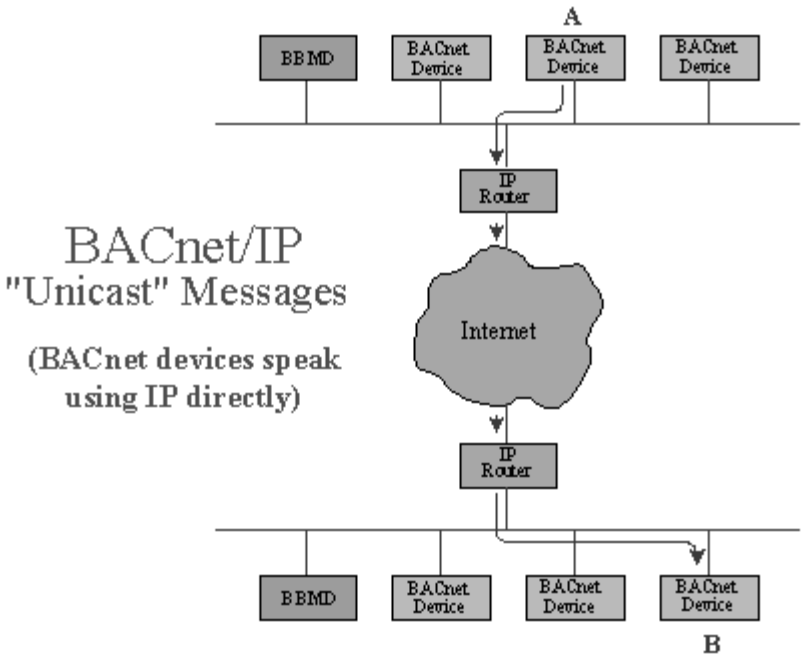


Figure 2.11: BACnet devices communicating directly to each other via the Internet

Lately, there have been many research papers published in the area of integrating Zigbee with BACnet [**Par071**] and integrating BACnet with the Smart Grid initiative [**DeC10**] (the Smart Grid initiative is a push by major developed countries, including Canada, to allow the electrical grid to be controlled by a communication medium for home automation and control). These two important advances, in addition to other advances in integration of BACnet with modern technologies, promise to make BACnet a promising and important smart building communication protocol. However, using BACnet in its current wired form necessitates building the structures from the bottom up to include the wiring, the sensors, and the other BACnet devices and routers. However, this is very improbable and costly, and thus using BACnet from this approach is not feasible. However, given the promising surge in recent attempts in integrating BACnet with wireless MACs, the Internet, technologies such as XML, and the Smart Grid, it remains to be seen whether BACnet will become a major player in the area of smart home networks.

2.5 Conclusion

This literature review has discussed the critical importance of energy-conscious comfort control in structures by analyzing [**Tak09**] and comparing it and analyzing its pros and cons with the objectives of this thesis, in addition to short analyses of other related papers. Then, energy management algorithms, have been explained by analyzing [**Men**], along with short analyses of other related papers. Finally, the topic of the WSNs routing protocols needed to support smart homes has been presented. This has been done by comparing three prevalent communication protocols: Zigbee, Bluetooth, EPIS, and BACnet [**Lee08**] [**Beg09**] [**Pen09**] [**Mik00**] [**Par071**], and proving that Zigbee is the most feasible and attractive of them all. It remains to be seen, however, how much the integration of BACnet with current modern communications protocols and systems is successful, and would thus be an attractive platform on which Zigbee routers and devices can communicate.

Chapter 3: CEMA Algorithms and Simulator

3.1 Preliminaries: CEMA Algorithms

3.1.1 Introduction

In this chapter, an in-depth analysis of two Comfort and Energy Management Algorithms (CEMA) is presented; controlling HVAC operations and controlling the lighting of a typical home. In addition, an option of adding a humidifier to the control system is also presented. Before the algorithms are presented, an overview of the engineering concepts behind the algorithms is discussed, in addition to the design of the WSN used to support the CEMA algorithms. Then, flowcharts for each type of CEMA algorithms are examined.

The simulator written to test these two CEMA algorithms is then presented. The choice of the programming language and development environment are discussed, in addition to presenting the backbone of the simulator, the main components of the simulator, and the simulator inputs and outputs. Finally, the sections of code and the .dll files written by others are presented, explained, and their contributions recognized.

3.1.2 Heat Load and HVAC Wattage Calculations

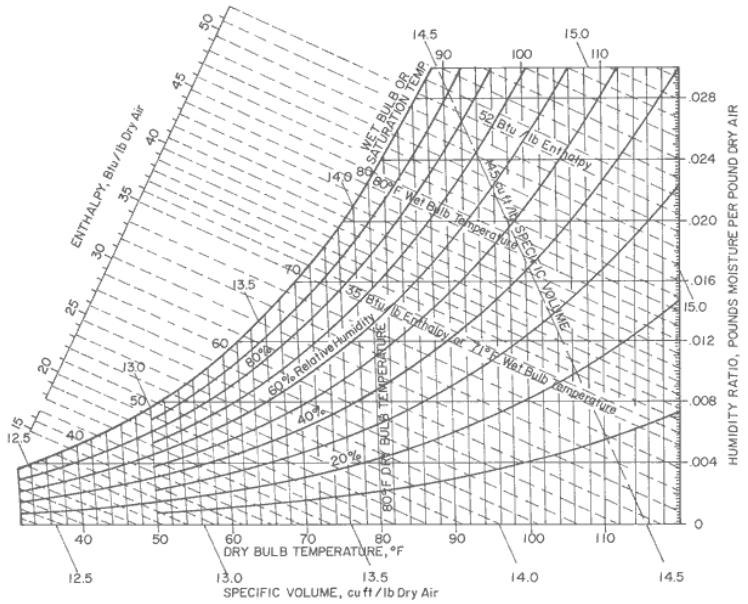


Figure 3.1: The psychometric chart [**Dar**]

One of the basic scientific principles known to man is that energy is neither created nor destroyed. This concept applies to countless applications in daily activities, and comfort control is no exception. The founding principle of the engineering of comfort control is that energy is neither created nor destroyed; it only flows from an area with higher energy to an area with lower energy. Thus, this energy flow must be controlled or artificially reversed in order to maintain a sense of comfort. This energy manifests itself in the temperature inside the residential space; the more the energy inside the enclosed space, the higher the temperature, and vice versa. Energy is transferred between spaces through conduction, convection and radiation. These three types of energy transfer in turn affect relative humidity and other factors according to well-known physics laws. These laws are laid out and explained in multiple reference books employed in this thesis [**Car65**], [**ASH821**], [**ASH82**], [**Jon94**], [**ASH08**], and [**Bel081**]. As a note, these reference books all explain the more complex heating/cooling load and HVAC wattage design that will be discussed later on in this chapter.

Figure 3.1 shows the backbone of all the calculations regarding comfort control; the Psychometric Chart. This chart shows the many thermodynamic properties of wet air (i.e.: air

that has water vapor) at a constant pressure, based on the three methods of heat transfer outlined above. This chart connects dry bulb temperature, wet-bulb temperature, and specific volume of dry air. In addition, the graph connects humidity ratio and enthalpy of dry air to the before-mentioned axes. In order to accurately describe how comfort can be mathematically maintained, a discussion of these axes must be presented [**Car65**] [**Jon94**].

- The Dry Bulb Temperature refers to the famous way of measuring temperature in a conditioned space, which is using a thermometer. The unit is in Fahrenheit.
- The Wet Bulb Temperature measures the saturation of air with water vapor, taking into account the dry-bulb temperature, the relative humidity, and the air pressure of the air in the conditioned space. This measurement is done by measuring temperature using a thermometer whose sensing bulb is covered with a water-soaked material. The unit is in Fahrenheit.
- The Specific Volume of the air is the volume of air given a unit mass. It is used to measure how dense the air in the conditioned space is, relative to the other properties of the air in the conditioned space is. The unit is in ft^3lb^{-1}.
- The Humidity Ratio measures the amount of water vapor relative to the total mass of air in the conditioned space, in grains of water vapor to pound of air (7000 grains = 1 pound).
- Enthalpy measures the energy stored inside an air mass of a conditioned space. This energy includes all types of energy including the energy stored in the water vapor in the air mass of the conditioned space. Its units are British Thermal Units per Pound (BTU lb^{-1}) of dry air.

In a conditioned space, a specific point on the psychometric graph describes the state this conditioned space is in, using the axes described before. In any comfort control application, the idea is to keep the range where the points are (i.e.: the state of the conditioned space) within the comfort range, i.e.: the range of values where a sense of comfort is perceived. This range is well defined in literature, either by illustration [**Tak09**] or in a tabular format [**Jon94**], and this range will be detailed in the next section of this chapter, which talks about the CEMA simulator design.

The concept of heat transfer can be mathematically quantified and studied by dividing actual heat gain in conditioned spaces into two types; sensible heat gain/loss and latent heat gain/loss. This mathematical quantification is needed so as to satisfy the above-stated requirement for the sense of comfort, i.e. keeping the state of the conditioned space within the comfort ranges on a psychometric chart. Sensible heat gain/loss is the gain/loss of thermal energy a person can feel as a rise in temperature; this is the more prevalent form of heat gain/loss. Latent heat gain/loss is the so-called invisible heat gain/loss; one which an occupant of a conditioned space does not directly feel. This latent heat includes heat energy stored in the water vapor in a conditioned space and the water vapor coming from evaporation off a human body. The ratio between sensible heat and latent heat differs according to the type of application. For example, factories and kitchens have both a high sensible heat content and high latent heat content (due to the cookers and other heavy equipment there. However, residences have high sensible heat content and low latent heat content [**ASH82**]. Nevertheless, latent heat is chosen to be included in the calculations detailed below for accuracy when comparing results to real-life simulated scenarios (see chapter 4 for results).

Sensible and Latent Heat gains and losses and intertwined and connected. Both can be affected by the three forms of heat transfer (conduction, convection, and radiation), and both can be affected by each other. In addition, in a normal residential space, many factors come into play; such as the type of walls and windows in use, the weather, or insulation used. These factors also include the activity of the occupants, residential equipment usage, deterioration of the residential structure on the long term, etc. As can be seen, these calculations can get very complicated very quickly if no assumptions or simplifications are made.

There are two well-known texts which tabulate these calculations, which compute the heat load of the conditioned space; that is, heat load is a measure of the transfer of all types of energy inside and outside of the conditioned space. These two texts are [**ASH811**] and [**Car65**], with additional modifications for residential structures in [**Car651**]. Equations **(C.1)** through **(C.16)** in Appendix C tabulate the exact simplified calculations needed for obtaining the instantaneous sensible and latent heat loads of a conditioned space. The calculations are the same for summer months (where the ambient outside temperature is more than the ambient indoor temperature) and winter months (vice versa the summer months), except in that in Equation **(C.5)**, the

difference in temperature term is the actual difference in the winter months, while in the summer months it is the equivalent temperature difference [**Car65**].

The magnitude of electrical power a typical HVAC system consumes is dependent on its design and rated power consumption. For example, if the HVAC system is based on the CAV principle, the air conditioning equipment will operate under the largest heat load of all the conditioned spaces inside the residence. On the other hand, if the HVAC system is based on the VAV principle, the calculations are done separately for each conditioned zone. Keep in mind that for overall evaluation of a certain home, all conditioned zones have to be considered.

The instantaneous sensible and latent heat are used to calculate the dehumidified rate of flow of air from the air conditioning equipment which is needed to maintain a steady set of conditions for a certain conditioned zone. The equation for obtaining this rate of flow of dehumidified air is shown in Appendix C (equation **(C.17)**). However, this rate of flow is based on the mixed air temperature (equation **(C.19)** in Appendix C), and the apparatus dew point temperature (equation **(C.18)** in Appendix C), which is a measure of the amount of water vapor in the conditioned space [**Jon94**]. Substituting equations **(C.18)** and **(C.19)** in **(C.17)** yields the following equation:

$$cfm_{da} = \frac{[cfm_{ventilation}(T_{Room} + 1)] - \frac{ERSH}{1.08}}{[T_{outisde} - 2T_{Room}]} \tag{3.1}$$

The term $cfm_{ventilation}$ denotes the amount of ventilation required for proper survival; this value is standard according to the circumstances of the conditioned zone [**Car65**]. ERSH denotes the effective room sensible heat (equation **(C.14)** in Appendix C). Equation (3.1) is valid for both summer and winter months, since the three equations it depends on (which are all from [**Car651**]) are valid year-round.

After the term $cfm_{da}$is calculated, there are two ways of calculating the minimum wattage of the equipment required for air conditioning. If the HVAC system is based on the VAV, the minimum wattage required is the sum of the latent and sensible heat loads [**Car651**], divided by the SEER. If the system is a CAV-based system, then the air flow required is the biggest out of

all the zones, and is used in the following equation to calculate the minimum HVAC wattage needed for comfort:

$$Minimum\ Wattage = \frac{1}{SEER} * cfm_{da} * \Delta h * density_{air} \quad (3.2)$$

The last three terms constitute the total heat load in need for compensation by the HVAC. This is not the rated wattage of the HVAC system. To illustrate this point, a short discussion of HVAC efficiency will now ensue. HVAC efficiency can be described in two interconnected terms, the Coefficient Of Performance (CoP) and the Seasonal Energy Efficiency Ratio (SEER). CoP is a dimensionless value that is not yet widely adopted, but basically the more the COP the more efficient the HVAC system. The other term, the SEER, is the more widely adopted HVAC efficiency metric, and it measures how much cooling load (in BTU hr^{-1})1 watt of input electrical power produces. The current minimum acceptable SEER rating in Canada and the US is 13 BTU $hr^{-1}W^{-1}$. In the CEMA simulator, the wattage taken from the residential electrical supply, the rated wattage of the HVAC, is actually total zone heat load (the last three terms in Equation (3.2)) divided by the SEER of the HVAC device. The resultant is the value used in calculating the contribution of the HVAC system to the total operating cost of the home.

3.2 Designs of CEMA Controller and Algorithms

3.2.1 WSN Backbone and CEMA controller Design

Figure 3.2: Illustration of the WSN used for this thesis

Figure 3.2 shows a typical floor plan for a single family dwelling or a townhouse – the majority of residence types in Canada today. As has been explained and analyzed in Chapter 2, the best

way for accurate, room-by-room comfort control, regardless of the type of HVAC used, is through the use of a WSN throughout each home. As was also explained, Zigbee is the current best option as a network protocol for this WSN. This is mainly due to its frequency hopping feature, its usage of the free ISM frequency band, and its ultra-low energy consumption (the Zigbee nodes only wake up to send or receive packets, thus conserving energy) [**Zig06**]. Figure 3.2 also shows the proposed WSN network. The rectangle is the CEMA controller, which controls all aspects of economical comfort and energy management control in this floor. It has been proven that only one controller is needed for such types of relatively small structures; i.e.: no hierarchy between the sensor nodes and the controller is needed [**Lee08**]. This lack of hierarchy provides an additional benefit; much less cost and much less complexity.

Figure 3.2 also shows the sensor nodes distributed along the floor plan (the small ovals with black streaks in them). The main idea of the sensor distribution is dividing the area to be controlled into zones and, for each zone individually, comfort control is set. This is regardless of the type of HVAC used; the only difference is the quality of comfort control given by CAV or VAV systems. It is assumed that the indoor walls and doors are insulated enough so that inter-zone dependencies can be ignored. Thus, each zone is simply each room in the floor plan, and including also the hallway in Figure 3.2. One notices in the design that there are two sensor nodes in each zone. This is for redundancy; if one sensor fails, the reading of the other sensor is used. If both are active, then the average of both readings is computed and used.

For the purposes of this thesis, and after presenting and analyzing the meaning of comfort in both Chapters 2 and 3, three metrics are defined; these metrics are what the CEMA algorithms are currently judged on: [1] Temperature [2] Relative Humidity [3] Light Intensity. Thus, the sensor nodes pictured in Figure 3.2 will measure these three parameters in every set time interval, and relay the information to the smart CEMA controller directly. Figure 3.3 shows the block diagram showing the process flow of the proposed system:

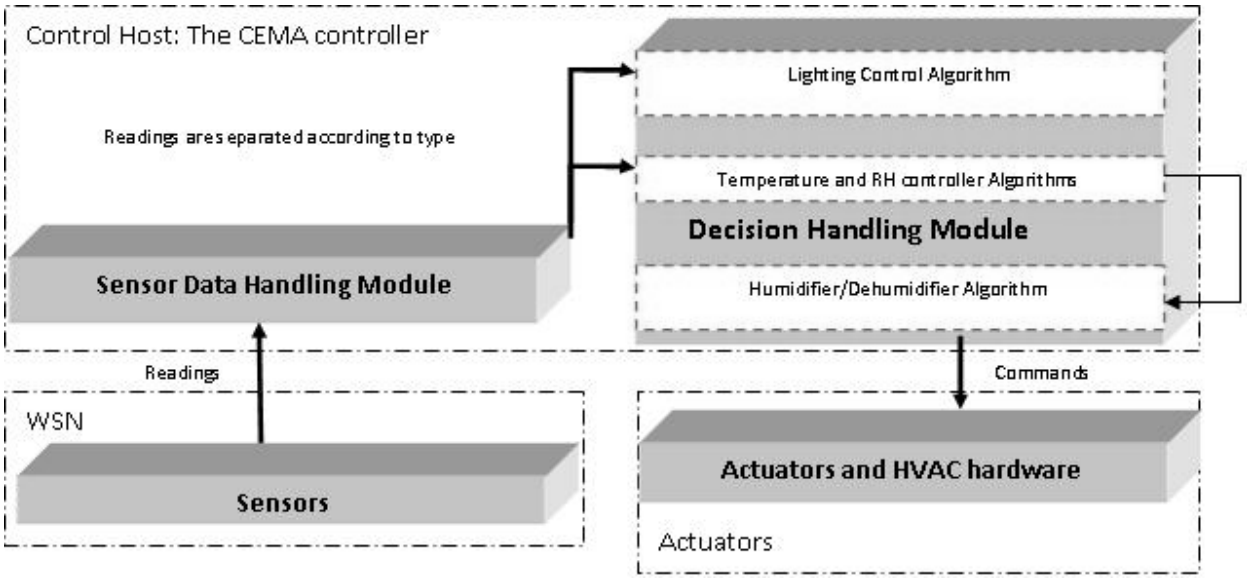


Figure 3.3: Flow control for the proposed CEMA system

The sensors send their data every fixed time interval directly to the CEMA controller, which distributes the readings according to their types; the light intensity readings to the lighting control algorithm and the temperature and relative humidity readings to the temperature and RH controller algorithm.

The key to the usability of this system is its modularity; more algorithms can be added to the decision handling module without theoretically affecting the system. Of course, the number of algorithms implemented in the system is limited by the types of readings an actual sensor node can handle, and by the capabilities of the system in hardware and network-wise terms.

After the algorithms are implemented, the commands to the HVAC hardware and/or the lighting dimmers are given out via cutting off/allowing electricity to the HVAC hardware and/or decreasing the voltage given out to the needed lighting to be dimmed. The area of implementing the CEMA algorithms is out of scope with this thesis, as here we concentrate on proving the quantitative and qualitative feasibility of the CEMA algorithms, and not their implementation.

3.2.2 Individual CEMA Algorithms

As has been stated in the previous section, this thesis concentrates on the three most energy-intensive aspects in comfort control in residential structure: heating control, lighting control, and humidity control. The last control is optional, meaning that not all residences have an extra humidifier/dehumidifier in their air conditioning systems. It is also worthy to note that in controlling the temperature of a conditioned zone, the relative humidity of that zone naturally changes, in line with the well-known laws of thermodynamics [**Jon94**]. However, in some environments, this change in relative humidity is not enough for a sense of comfort to be maintained, and thus a humidifier/dehumidifier can be included. In this CEMA simulator, the option of a humidifier/dehumidifier is an input by the simulator user.

Figure 3.4 below shows the flowchart for the CEMA algorithm concerning temperature and relative humidity control:

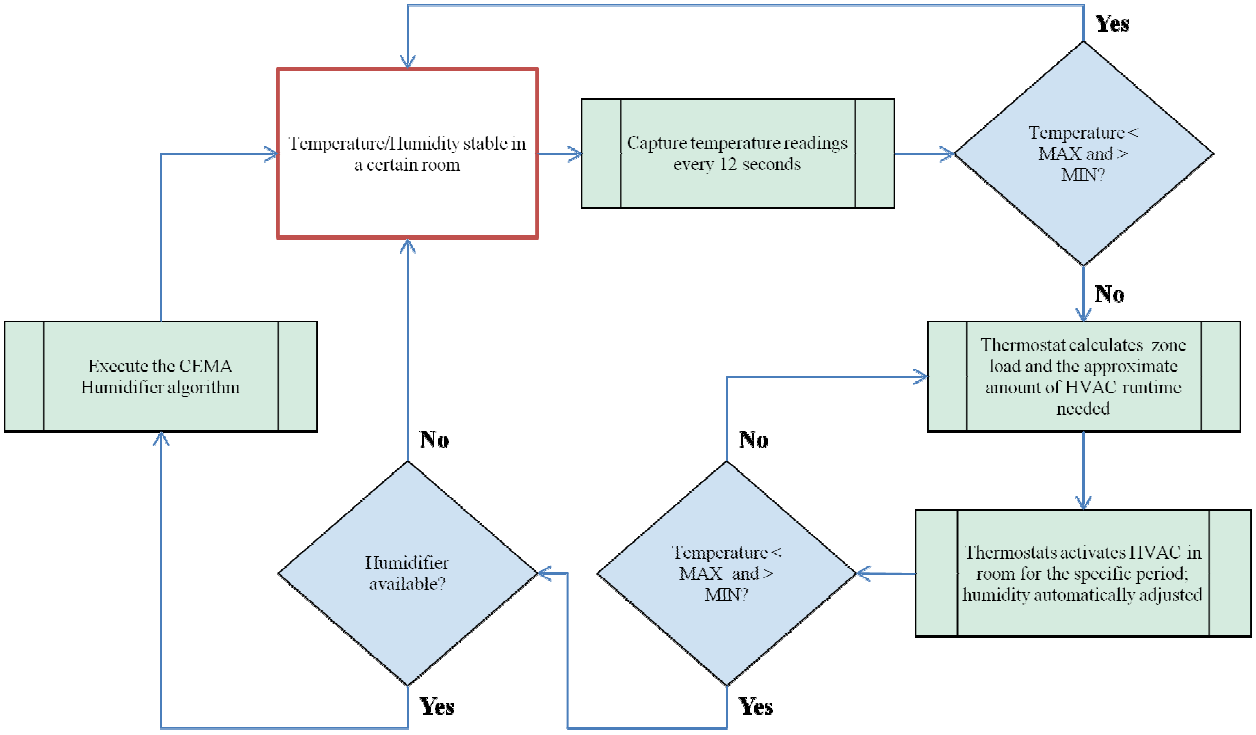


Figure 3.4: CEMA temperature control algorithm

This algorithm is valid for both CAV and VAV-based HVAC systems, and for winter and summer months. In every zone, temperature readings are captured by the sensor nodes every set time interval and sent to the CEMA controller, which takes the average of the values and checks whether the temperature value is between the user-set values. When the recorded temperature is out of the needed temperature range, the CEMA controller calculates the instantaneous heating/cooling load of that particular zone, and on that load calculates the approximate time the A/C has to operate in that zone. The method of the operation of the A/C, in addition to the estimate of time is based on the calculations in the previous section, and the calculations differ between a CAV-based and a VAV-based system. The rate of change of temperature during the operations of an HVAC VAV is faster than a CAV-based system; and both rates of change are designed to be linear. The values of the CAV and VAV rates of temperature change are outlined in Chapter 4.

After the A/C operation time has elapsed, the temperature is checked in that zone again. If the temperature is more than one degree Celsius away from the user-inputted range of temperatures, the above process is repeated again; the load and the approximate A/C working time are calculated and used. Then the temperature is checked again twelve seconds before the end of the A/C operational time. Once the temperature is between the required ranges, the system then goes to check whether there is a dehumidifier in the system or not; if yes, then the humidifier/dehumidifier algorithm is implemented (it is the third algorithm discussed in this section), if not the system returns to checking the temperature every set time interval.

Figure 3.5 below shows the CEMA algorithm concerning lighting control:

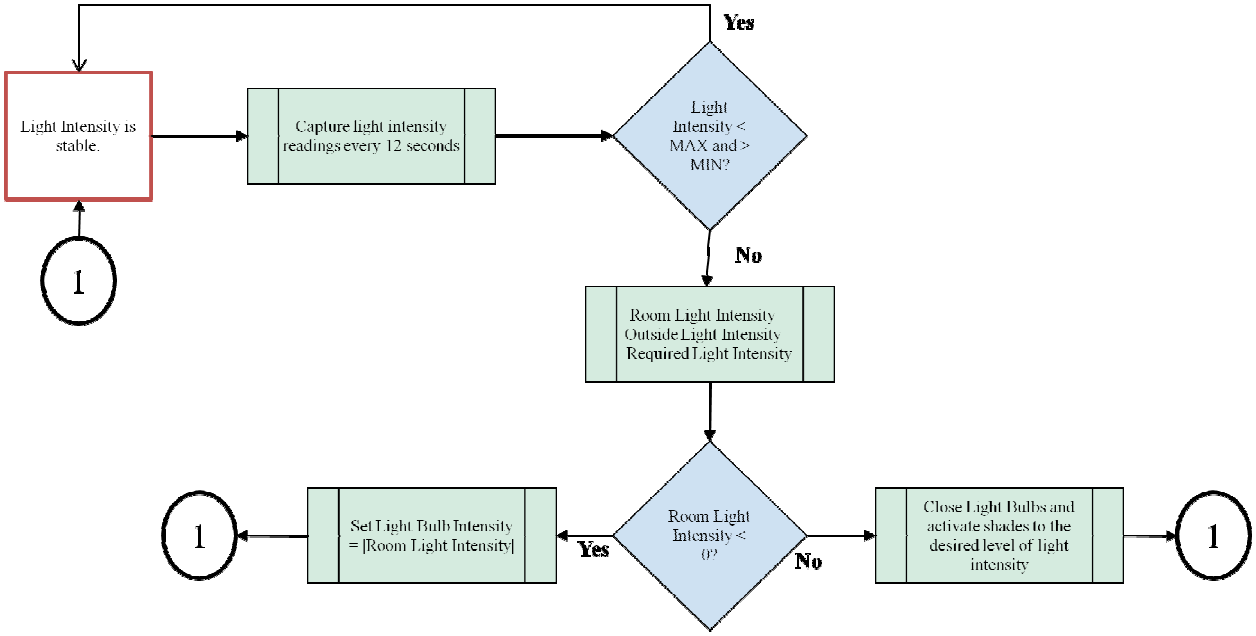


Figure 3.5: CEMA light control algorithm

As with the rest of the CEMA algorithms, the sensor nodes measure light intensity, in lux, of the zone. If the light intensity is outside the user-defined bounds, the required zone light intensity deficit is computed. This deficit can either be a positive number (meaning there is a deficit that needs to be addressed by turning on the lights), or be a negative number (meaning there is excess lighting in the room). The key here is the maintenance of a constant light intensity; either the lights are turned on by a certain percentage or the zone window is shaded by a certain percentage so as to maintain constant light intensity inside the conditioned zone. This naturally incurs significant monetary savings while keeping the same level of comfort, as the lights are not operating under full power.

The CEMA humidifier/dehumidifier algorithm is very similar to the algorithm presented in Figure 3.4 for temperature control, except in that it starts only after the A/C operates for a certain period of time, and that the property in question is the conditioned zone relative humidity not temperature. The rate of relative humidity increase/decrease is set according to Equation (3.3) below. Also, the operation of the humidifier/dehumidifier is independent of the workings of the A/C once initiated as outlined above. Finally, the workings of the humidifier/dehumidifier are out of scope of this thesis.

Finally, Figure 3.6 shows the current CEMA controller finite state machine which summarizes the three above-stated algorithms:

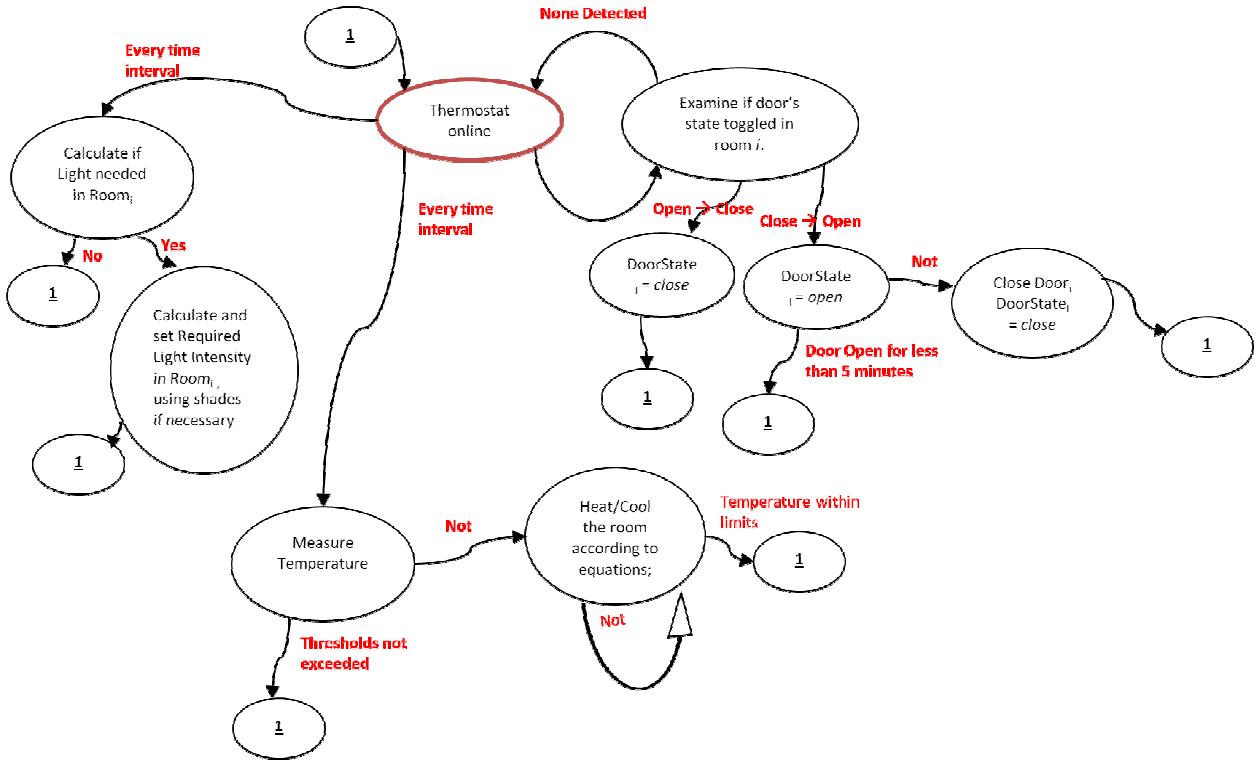


Figure 3.6: CEMA algorithms' FSM

Three important notes have to be made about the above figure. First, this FSM applies to each zone independently. Second, the CEMA controller also intends to control which doors are open and which are closed for each zone. That is, for example, if an outdoor door is open for more than five minutes, it most likely means the occupants forgot to close that door, and thus the CEMA controller closes that open door. Third, regardless of the status of operation of the HVAC or the humidifier/dehumidifier, a zone temperature is always being captured every fixed time interval and utilized according to the algorithm shown in Figure 3.4. Once the humidifier/dehumidifier is active, its operation is independent of any subsequent operation of the A/C.

3.3 CEMA Simulator Design

3.3.1 Introduction

In this section, the details of the mechanism used to test the CEMA controller and algorithms are presented. These details include the programming language used, the mechanism's main characteristics and features, and finally a detailed list of the type of coding used to write up the simulator.

A mechanism must exist to mimic the WSN-based comfort controller, while still maintaining the simplest and least complex structure. There are three main methods of achieving this mimicking; one can build a hardware prototype, or code a simulator or an emulator. Hardware prototypes, when up and running, provide an excellent one-to-one analysis of the actual workings, losses, and efficiencies of the comfort control application. However, there should be some kind of software analysis first to verify the feasibility of the project, to avoid costly rebuilds of the prototype. Thus, either an emulator or a simulator can be designed to test the comfort controller. However, in the case of WSN-based comfort control in residences, designing an emulator would be extremely difficult. This is because an emulator is a software-based analysis of a hardware-based system (such as the proposed comfort controller). This analysis imitates complex wireless communication systems from the ground up and can be excruciating, especially if analysis of all the other OSI layers is included Examples of the resultant complexity includes implementation of the network layer with its buffers and packet drops and bit error rates, and imitation of quality of service for some network protocols. The reader must put in mind that the two main goals of this above-stated mechanism are to prove that the CEMA controller is feasible and that it is an economically attractive option. Under these circumstances, designing a simulator is much more feasible. A simulator differs from an emulator in that a simulator does not include every single process or algorithm in its workings, yet it efficiently imitates the main flowchart and fulfills the main aim for which it has been written.

The programming language used to implement the CEMA simulator is the JAVA© Language and the development environment used is the Eclipse Integrated Development Environment (IDE). The choice for both has been based upon the fact that both are open source and thus are free and continually reviewed. In addition, new modules are being constantly added to both the programming language and the development environment, and here lies the power of the JAVA© language; the ease of integrating different functionalities into the code of a developer. This is unlike its main competitor; the C family of programming languages, which is not free, not of the open source type and integrating different functionalities is much harder. The only potential pitfall in using an open source programming language and development environment is the issue of bugs in either which might complicate the simulator development. However, no bugs in the JAVA© language or the Eclipse IDE have been encountered during the course of coding the CEMA simulator.

3.3.2 Simulator Backbone, I/O, and External Pieces of Code

In any simulator, there are a set of events to be executed, on which beginnings and endings might or might not be dependent on other events. Around these events, there are constants, variables and assumptions which control the flow of such events. There are many different types of simulators, each describing different scenarios. This thesis describes a model that has sources of randomness (the operation of the outdoor doors, as seen in Figure 3.6), and is dynamic (time sensitive) [**Lee09**]. As such, two types of simulators can describe this model; simulators whose events occur at discrete time instances, and simulators whose events occur continuously. The former is called a discrete-event simulator [**Fis01**]. An example of the former is modeling a bank queue, and an example of the latter includes simulating classical mechanical concepts [**Lee09**]. Each type of simulator has its advantages and disadvantages, and each is suited better for different applications. A discrete-event simulator can better model a comfort and energy control residence, though, because at any one instant in any one zone (remember, the controller controls comfort control in multiple zones in parallel), the controller is in one state of the states shown in Figure 3.6.

In this thesis, the discrete event simulator concept is implemented by initiating a queue which is filled with, in the correct chronological order, events which describe the potential movement from one state to another in the FSM.

Before the workings of the simulator are shown, its initial set of constants and assumptions are shown, along with any needed explanation for their existence. In addition, the simulation inputs and outputs are also listed. The set of assumptions are as follows:

- The house is always facing northward.
- The house is assumed airtight in all types of simulators; i.e. no unaccounted for leaks and cracks throughout the home.
- There are no other houses or structures near enough to the tested house to cause any shading.
- Wind Chill Factors are not included in any calculations.
- House Properties are the same throughout the residential structure (same thickness, thermal resistance, color, and type of wall, roof, and windows).
- There are no heat losses or gains other than those tabulated in Appendix C.
- Temperature, relative humidity, and light intensity are measured every 12 seconds. This value was obtained with consultation from experts in the industry.
- Windows are never opened. Any normal infiltration occurring in a residential structure is accounted for in the equations stated in Appendix C which were used in the previous section. This assumption is for ease in programming the CEMA simulator.
- The thermal resistance of an external wall, door, window, or roof are: 0.062, 0.497, 0.568, and 0.044 BTU $hr^{-1}ft^{-2}F^{-1}$ respectively. The choices for these values have been set to be exactly similar to the values placed in the DesignBuilder® simulator, as will be explained later in Chapter 4.
- The lighting of every room in the home is to be defined as a single, hanging, 100W incandescent light bulb that produces 1750 lux. It is assumed to be always on through the duration of the CEMA simulation, albeit not always with full power. (In the DesignBuilder® Simulator, the lights are always assumed to be on, full power).
- There are no devices in any simulator being run for this thesis. This is for simplicity. An exception is the humidifier/dehumidifier, which when chosen to run operates at 500W.

- Each outdoor door, and its corresponding room, is modeled as follows. The incoming rate of people at the door is modeled as a Poisson process with arrival rate λ people per hour, and the service rate μ people per hour of the door (i.e.: the rate at which people leave the room) is modeled as an exponential distribution. No importance is given to the maximum capacity of the room as practically it is impossible for the room to be filled to capacity in a normal residence As such, the door and its room can be modeled as an M/M/1 queue [**Leo04**]. Throughout this thesis, λ = 0.1 arrival/hour and μ = 0.2 departures per hour. The reason for these choices is given in Section 4.2.4.
- The rate of increase or decrease in temperature or relative humidity in the CEMA simulator is determined by running the DesignBuilder® simulator using a VAV-HVAC system, then drawing a line of best fit (utilizing Microsoft® Excel) on the data points representing the operation of the HVAC. From it, it has been estimated that the rate of temperature increase/decrease is linear with the slope ±0.002 °F s^{-1} and the rate of moisture content increase/decrease (either only when a humidifier is present or the dehumidification effect of an HVAC system) is modeled according to the following equation, where y is the new humidity content, t is the time difference between the current simulation time and the time when the humidifier has been activated, and x is the old humidity content:

$$y = x * t^{\pm 0.279/2} \tag{3.3}$$

The inputs of the CEMA simulator are:

- A Comma Separated Values (CSV) file containing the details of the residential structure in question; such as the number of zones, number of outdoor walls in each zone, thermal resistance of each wall, and so on with every other aspect of the residential structure. In addition, this file contains the specific comfort requirements for each zone.
- User Inputs of the required simulation time required, the season the simulator is it to run in, whether the residence in question operates a CAV-based or a VAV-based HVAC system, and whether the residence also operates a humidifier.

The output of the CEMA simulator is:

- A CSV file containing second by second recordings of temperature, relative humidity, and light intensity of each conditioned zone, in addition to the outdoor air properties. This file, after further processing (see Chapter 4), will provide proof of the validity of the CEMA controller and algorithms.

An important note to be made is the units in use throughout this thesis. The reference books employed to determine the equations the CEMA simulator uses have the American system of units ($^{\circ}$F, BTU hr^{-1}, Cubic Feet per Minute (CFM)), while Canada uses the British system of units ($^{\circ}$C, W, m^3 s^{-1}). Thus, all inputs and outputs are based on the British system, while all internal CEMA calculations are based on the American system. An exception to this is the output shown from a certain simulator called DesignBuilder [**Des10**] whose purpose is explained in Chapter 4. To make a fair comparison between CEMA and DesignBuilder, all input parameters use the American system of units, and thus the output of DesignBuilder also uses the American system. This does not complicate the thesis in any way because what is gleaned from the DesignBuilder output, as is seen in Chapter 4, are the trends set by the output values, not individual values themselves.

In addition, a dynamic linked library (dll) file has been included. These files include functionality not written in the original piece of code, for any type of code, so that the written code is more efficient. In other words, the developer, when possible, does not have to reinvent the wheel when programming. For this thesis, one dll file has been included; a dll file mathematically describing the psychometric chart which the equations in Appendix C are based on. In addition a .jar file serves the same function but specifically for Java code, and two .jar files have been used; both served to allow the linkage of the psychometric .dll to CEMA simulator Java code.

3.3.3 CEMA Simulator Packages and their Contents

The Java programming language is based on the concept of **o**bject-**o**riented-**p**rogramming (OOP). This concept basically states that in order to fulfill the functionality of a program, entities called objects have to be created, which contain variables and functionalities. The method and protocol of interaction between these variables and functionalities constitute the overall program functionalities. This is as opposed to writing primitive code that is executed line by line.

In the Java language, OOP is translated into classes and packages. Classes are the entities which contain the variables and functionalities describing one entity in the overall programmer design. Examples include a class for router and another class for switch. Each is entirely separate from the other, and each contains functionalities and variables describing the entity. The protocol of the interaction between them constitutes the overall functionality of the program. Packages are simply a group of classes with similar functionalities; it is for organizational purposes only. An example would be grouping the router and switch classes under one package called Network Devices.

Six packages where designed to simulate a CEMA controller implementing three CEMA algorithms. The following is a list of the packages and their contents in terms of classes, along with a brief description of the class (the significant parts of the commented code are found in Appendix A of this thesis):

- decision_Handling_Module package: This package holds the classes responsible for the decision making in the program. It contains "buildingConstruct.java," the class which has the implementation of all CEMA algorithms, "Door.java," which has the implementation of a door model (used to indicate the type of door, whether indoor or outdoor, and its state, whether open or closed), and "Room.java," which contains the implementation of one conditioned zone.
- input_Output_Manipulation package: This package contains only one class, "CSV_Parser.java," which deals with all input and output CSV files the simulation works with.

- Queue_Element_Types package: This package contains twelve classes describing all the types of events that can be incoming into the queue. More types of events can be added with ease, classes which describe new CEMA algorithms. The list of these classes can be found in the class diagram of the next sub-section.
- simulator_Backbone package: This package has one interface and two classes. An "interface" is an entity which forms a backbone on which classes can extend and use. Interfaces are mainly used to allow classes to share global variables or functionalities without actually allowing "global" variables or functionalities. The interface in this simulator is called "SharedConstants.java." All the Queue_Element_Types classes are implementing the Shared_Constants.java file. The two classes in the simulator_Backbone package are DynamicEvent.java, and EventQueue.java, which describe a generic element in the queue of the discrete-event simulator, and the protocol on how to deal with the queue.
- thesis_Simulator package: This package contains the sole class, Main_Program.java, that acts as the engine of this simulator. This class actually runs the CEMA controller. It contains a graphical user interface, GUI, which allows the user to interact with the program. Appendix A shows the GUI used for this thesis, located at the end of Appendix A, in addition to the important piece of code detailing what happens when a user initiates the simulation.

Appendix D shows the complete class diagram of the CEMA simulator.

3.4 Conclusion

In this chapter, an overview of the thermodynamic principles and equations of comfort control for residential structures has been presented. Then, based on these principles, three CEMA algorithms has been shown and explained; one algorithm for temperature control, one for light intensity control, and the third one for relative humidity control. The WSN backbone supporting

the CEMA controller and algorithms has been also presented, as well as the overall flowchart of the CEMA simulator in the form of an FSM diagram.

Then, in the following section, the simulator itself has been shown and explained in terms of the choice of the coding language and coding environment, the basic calculations and the simulation assumptions and the detailed structure, in the form of a UML-based class diagram, of the overall code.

This chapter forms the basis of this thesis; it contains the main ideas, the algorithms and the code used to verify these algorithms. The next chapter details the results obtained from running the above-stated CEMA simulator.

Chapter 4: Simulator Results and Analysis

4.1 Introduction

In this chapter, analysis of the CEMA algorithms and the CEMA controller, in addition to the final results, are presented. Before the results are shown, an important note must be stated. The aim of the CEMA simulator is, as much as possible, to mimic a real life HVAC in a typical residential structure, putting in mind to adjust the simulator for the new CEMA protocols. However, we have to compare the CEMA-based HVAC system to a reference non-CEMA HVAC system in order to prove that the CEMA-based HVAC systems are more superior in cost saving and in terms of comfort control. As could be deduced from Chapter 3, the model describing typical HVAC systems is extremely complex and cannot be described using one or two simple equations. This is because the model should include changes in outside temperature and relative humidity, number of people in the enclosed space, number of appliances in the enclosed space, etc. Therefore, there is no one equation that can accurately describe temperature and relative humidity changes in an enclosed space. Two options then remain; either physically measure temperatures and relative humidity across a span of time in an actual enclosed space with no shading or other enclosed spaces next to it, or try to find a detailed simulator that does the above-stated functions.

The latter choice has been implemented. Two computer simulation programs were found in the area of HVAC system simulation; the Hot3000® program [**CAN09**] and the DesignBuilder® program [**Des10**]. The Hot3000® program has been designed by National Resources Canada, and is generally used by homeowners who want to improve the efficiency of their homes and track the general day-to-day costs of maintaining this home. It is detailed in terms of the inputs required, covering everything from the thermal conductivity of the building materials to the weather conditions for where one wants to place the simulated structure, to the efficiency of the

HVAC system. However, this simulation program has a complex graphical user interface, and especially with the format of the output given after running the simulation. This output is contrite, simplistic, and in no way near the detail of the outputs of the CEMA simulator.

The DesignBuilder® program [**Des10**] is a detailed, intuitive, and very flexible program designed to make simulating any type of structures relatively easier and more pleasant than most other simulators. The interface starts with an AutoCAD® like feel and allows the user to build the structure of his/her choice, again similar to AutoCAD®. Figure 4.1 shows the structure we have built using DesignBuilder for our purpose:

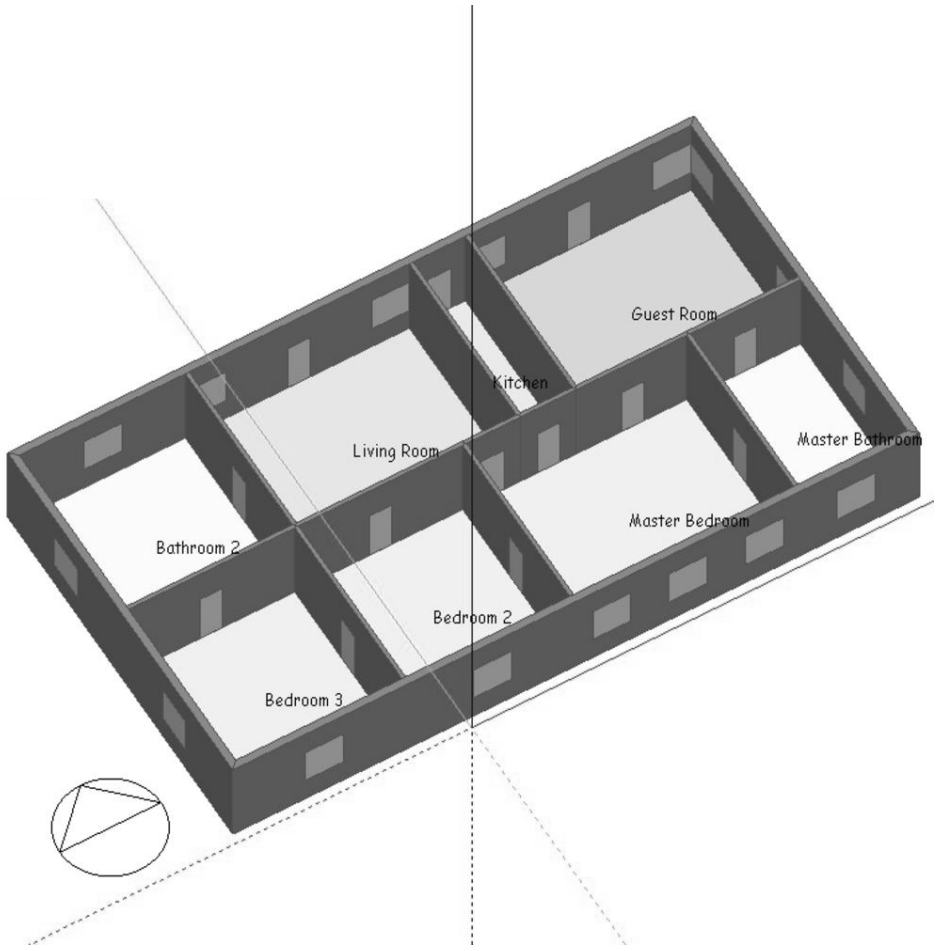


Figure 4.1: Home model used for this thesis

The DesignBuilder simulator allows you to draw, to a hundredth of any unit of measure, the spaces needed. In addition, as can be seen from Figure 4.1, windows, doors, and partitions can also be drawn to the same accuracy. Also, different zones can de described in different ways,

(toilet, living room, etc.). Each of these zones is color coded in the DesignBuilder simulator, but this coding is not evident in Figure 4.1. This in turn allows for different behaviors of people and appliances inside each zone, which allows the DesignBuilder® simulator to be more accurate. Figure 4.2 shows a typical snapshot describing the next step taken after drawing the enclosed spaces in question; setting up the myriad types of variables needed for the simulation:

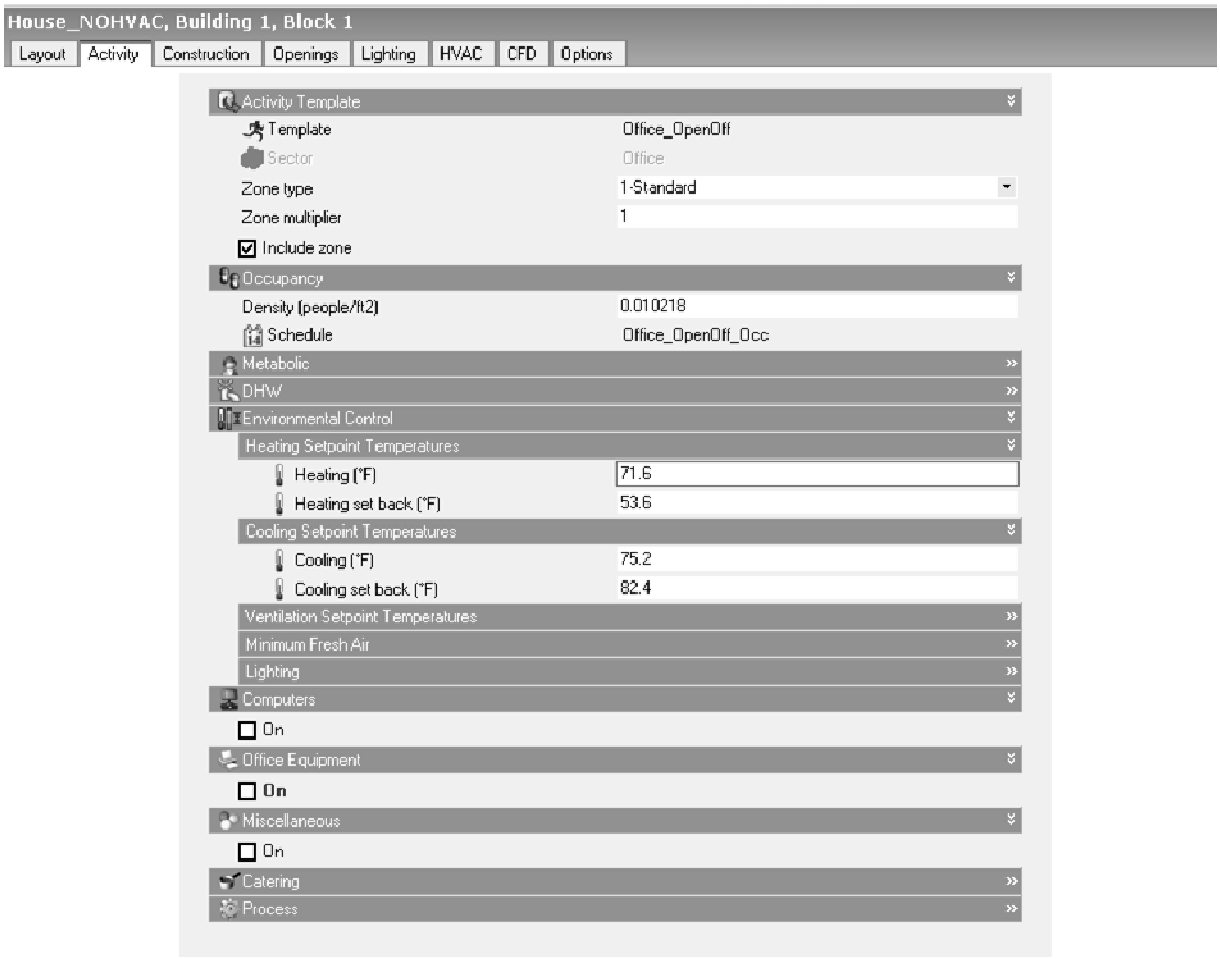


Figure 4.2: Setting up the simulator input parameters

As can be seen from the above figure, there are a lot of variables and constants that can be set up, or cancelled, as is needed, thus adding to the versatility of the simulator. For example, for the structure as a whole, the use of office equipment was cancelled, computers, and set the density of the people to the value seen from the above figure. The same goes with the construction materials properties, the HVAC properties, and the heating and cooling temperature cutoffs. In addition, all of the above variables and constants can be tailored for each zone individually.

Finally, the outputs of the DesignBuilder® simulator are immense and varied, and again can be tailored for the building as a whole (usually the average value of all the zones at a particular instant), or for each zone individually. These outputs include temperature, relative humidity, sensible heat load, latent heat load, electrical consumption, and total fuel consumption, among many others. The resultant plots are drawn to a maximum resolution of one data point every 30 minutes.

The rest of this chapter is organized as follows: First, reference values of outdoor and indoor temperatures for a typical summer week is shown, by running the DesignBuilder® simulator with no HVAC present. The same set of values for winter months is available, but is not shown due to space limitation. The values have been used in the CEMA simulator to illustrate how temperature and relative humidity fluctuate when the HVAC system is turned off. Then, temperature fluctuations inside two of the eight zones shown in Figure 4.1 are displayed and analyzed, for both winter and summer, for four days in each season. These values are calculated from the DesignBuilder simulator using a dual-duct VAV HVAC system (the CAV HVAC system results have not been covered here due to space limitations). No more than four days can be incorporated because Microsoft Excel® does not support more than 23,000 data points per graph, and given that the resolution of the CEMA simulator is 12 seconds, only at most 4 days can be incorporated in one graph. Only two of the eight zones will be covered because of space limitations in this report, and because each zone is assumed independent of the other for simplicity, as has been stated before. The same two zones are simulated using the CEMA simulator and the same two types of graphs are produced. Then, the DesignBuilder simulator is run again for both summer and winter for four days each, and the total kilowatt consumption of the building as a whole is calculated. The same output is then produced from the CEMA simulator. All four sets of outputs are compared and analyzed to determine which type of system beat achieves the tradeoff; namely achieving comfort control in the most economic fashion possible. A flat rate of 2.3 Canadian cents per kilowatt-hour is applied to both systems, again for simplicity.

The next set of outputs is related to the concept of the confidence interval, namely how correct and valid the CEMA system is given the randomness in the system. In this thesis, randomness comes from the rate of entry through external doors (modeled by a Poisson distribution), and the

duration of stay for each person inside each zone (modeled by an exponential distribution). As explained in Section 3.3.2, each external door with its corresponding room is modeled as an M/M/1 queue, and will be treated mathematically as such. The CEMA simulator is run multiple times, both in summer and winter, and the focus will be on one zone only, the guest room. The total kW consumption is calculated while varying the traffic intensity of the M/M/1 queue, and on that the confidence interval is calculated and presented.

The final set of outputs is related to the analysis of the addition of the humidifier/dehumidifier to the comfort control system. This piece of equipment either adds or removes humidity to the enclosed space, and in this thesis its method of control is very similar to the method of temperature control in an enclosed space, except that relative humidity is the controlling factor.

4.2 Results and Analysis

4.2.1 Reference Values

Figures 4.3 shows the outdoor and indoor temperatures, and indoor relative humidity values for the guest room shown in Figure 4.1, respectively (first plot from the top is the Air Temperature.). This room is chosen to be the room whose results are displayed in this thesis. The results of the other rooms will not be shown due to space limitation, except the living room whose output values are used to prove that under a different range of comfort temperatures, the CEMA simulator performs as expected. Also, by visual inspection, we found that all the output values from the other rooms vary slightly from the guest room, so the values of the guest room alone can prove the effectiveness of the CEMA simulator and its algorithms:

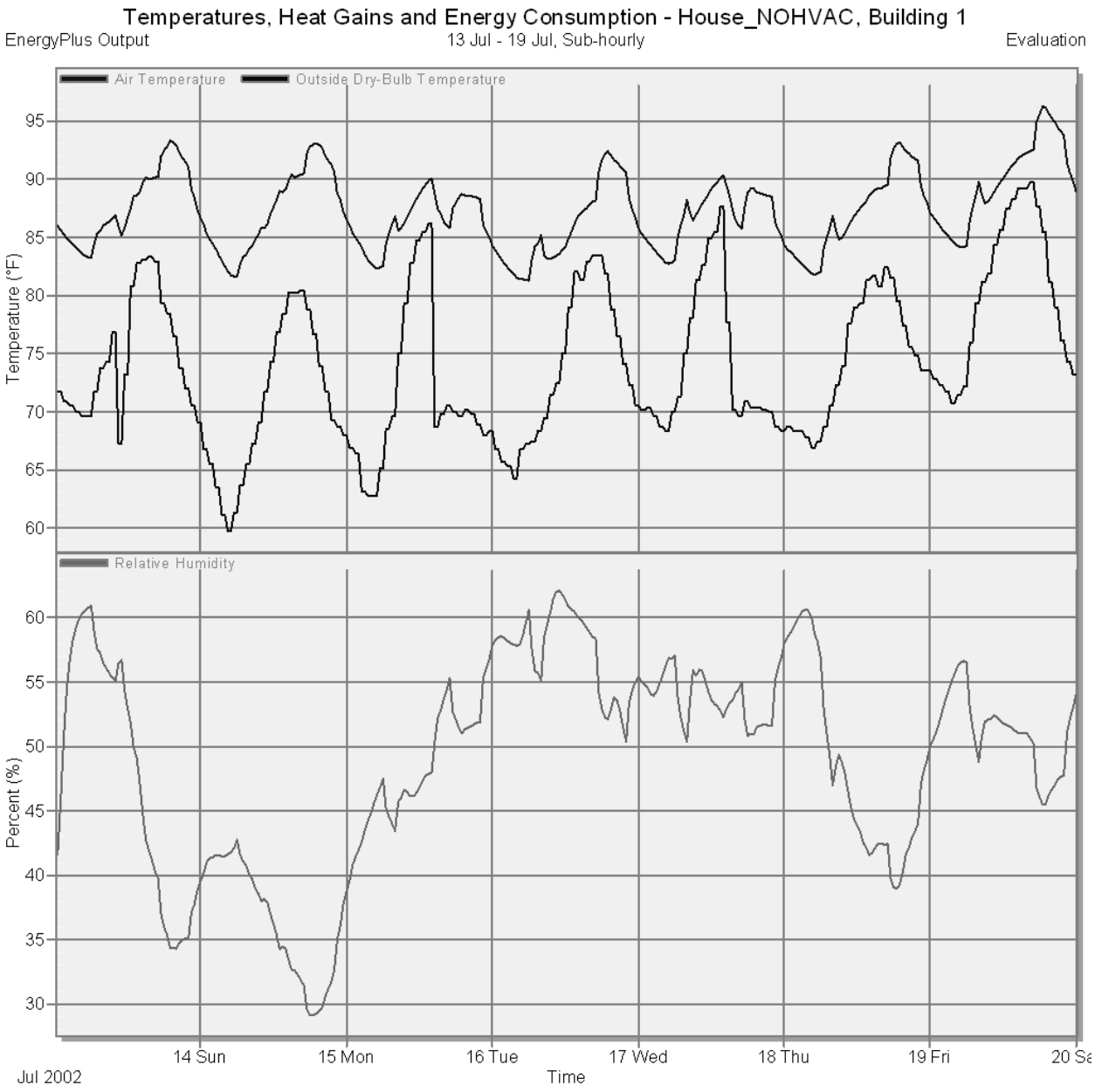


Figure 4.3: Outdoor, indoor, and relative humidity values for the guest room without HVAC

As is seen from the above figure, indoor temperatures are always more than outdoor temperatures. The reason for this phenomenon is that typical modern structures retain heat; that is, modern structures do not have any natural comfort control mechanism [**Bai94**]. An important note to state here is that DesignBuilder® graphs' resolution does not go beyond one data point every 30 minutes. Another important note is that the average of every 30-min interval data point throughout the summer design week displayed above was computed, then the average of every

two of the resultant data points was calculated to obtain 24 one-hour values of the indoor and outdoor temperature, and of the indoor relative humidity values. These values were then used as reference values in the CEMA simulator, taking into account that there are different set of values for either a summer environment or a winter environment. Also, it is to be noted that while temperatures increase to a peak around mid-day then taper off until night, relative humidity readings have no such pattern and are completely random over the course of the week

4.2.2 Comfort Control Analysis Results-Summer and Winter

Figure 4.4 shows the resultant graph after running the CEMA simulator on the guest room, for 4 typical summer days. The x-axis is the time in seconds, and the y-axis is the internal temperature of the guest room, in degrees Celsius. The upper and lower comfort temperature limits are set at 75°F and 70°F (23.89°C and 21.11°C), respectively:

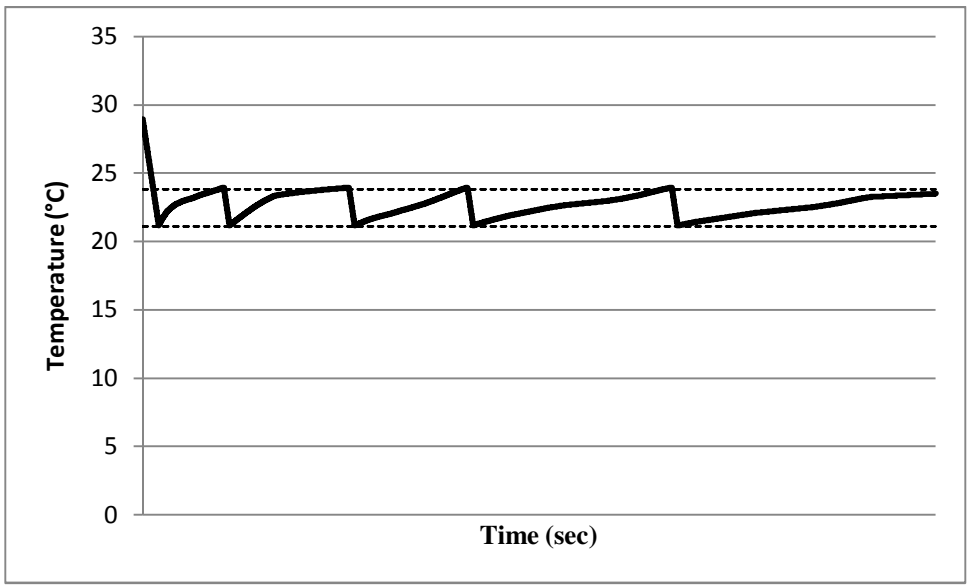


Figure 4.4: Indoor temperature values, guest room, for four typical summer days

As has been stated before in the Introduction chapter, the perception of comfort differs from one person to another, and this concept is relied upon heavily by the CEMA algorithms to achieve economical comfort control. That is, the indoor environment does not need to have a constant set

temperature to achieve comfort. Rather, comfort is achieved by two limits of temperature, within which comfort is felt. In this way, the HVAC heating/cooling elements, which eat up most of the electrical consumption of the HVAC, do not have to operate a lot, thus incurring significant monetary savings. In fact, from Figure 4.4, the only times the cooling elements of an HVAC are operational are when the temperature drops linearly with the rate stated in Chapter 3, which, as can be seen from the above figure, is a small part of the total simulation time of four days. The rest of the above graph is the normal temperature increase to the reference values, achieved by performing small incremental linear steps between the current data point and the next biggest reference temperature data value (i.e.: the nearest reference data point. In a similar fashion, Figure 4.5 shows the guest room for four typical winter days:

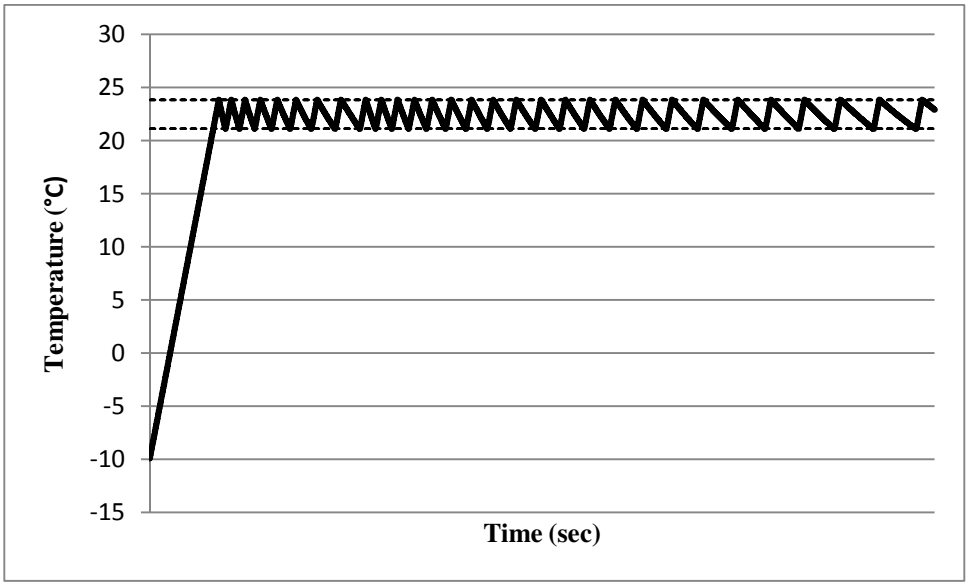


Figure 4.5: Indoor temperature values, guest room, for four typical winter days

The same concept applies for winter months; there is an upper and lower temperature limit, within which the occupants feel comfort. The heating elements only work when the temperature increases linearly; the rest is the normal temperature decrease when no HVAC is present. For reference, Figures 4.6 and 4.7 are similar in concept to Figures 4.4 and 4.5, except that the zone in question is the living room. As can be seen below, even with different upper and lower temperature limits, 70°F and 65°F (21.1°C and 18.33°C) respectively, the system is still feasible and performs according to design:

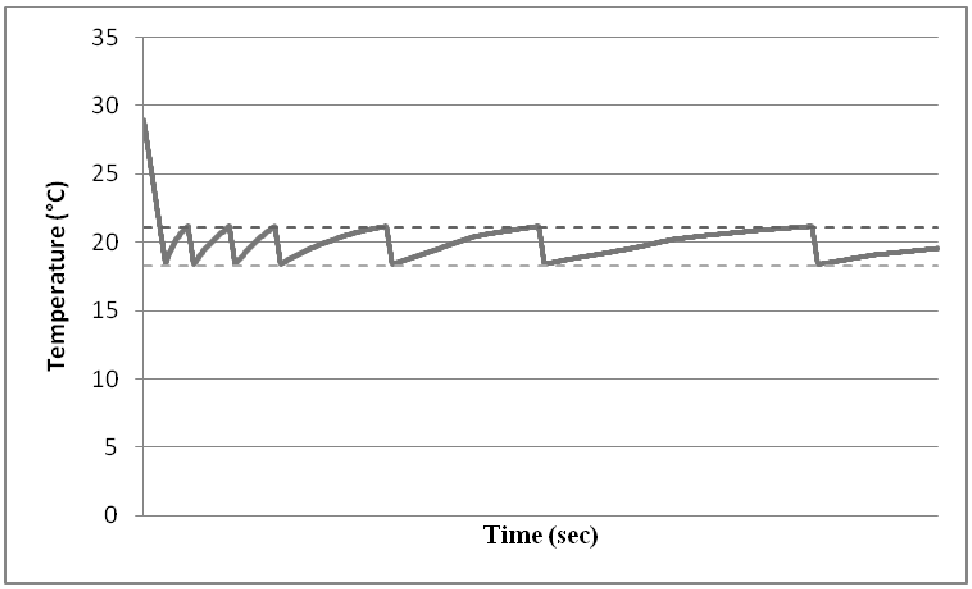


Figure 4.6: Indoor temperature values, living room, for four typical summer days

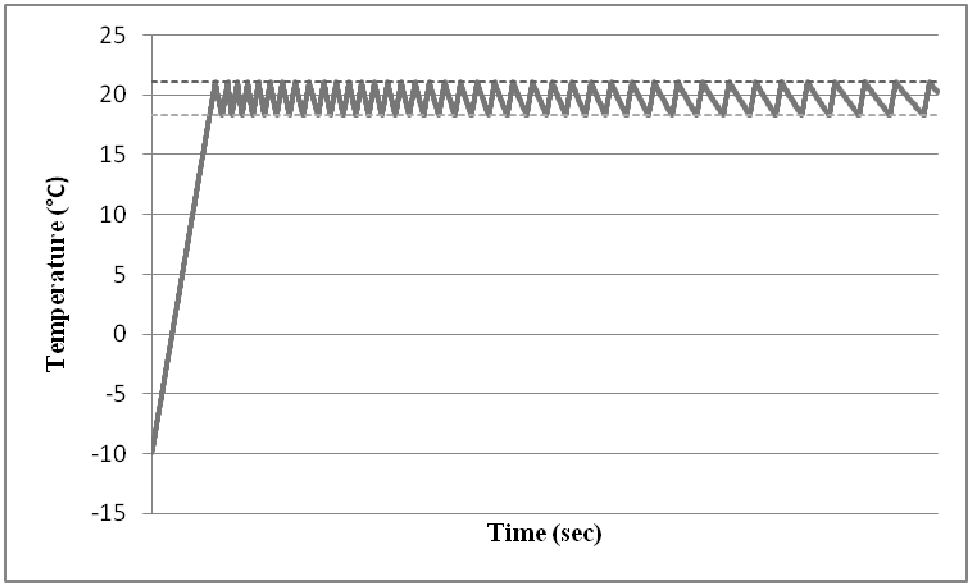


Figure 4.7: Indoor temperature values, living, for four typical winter days

Figures 4.8 and 4.9 show the temperature curves for the guest room for four typical days in summer and winter respectively. Care has been taken to replicate all the guest room properties in order to perform a fair, accurate comparison (Outside Temperature is the step-like plot for Figure 4.8 and the bottom plot in Figure 4.9):

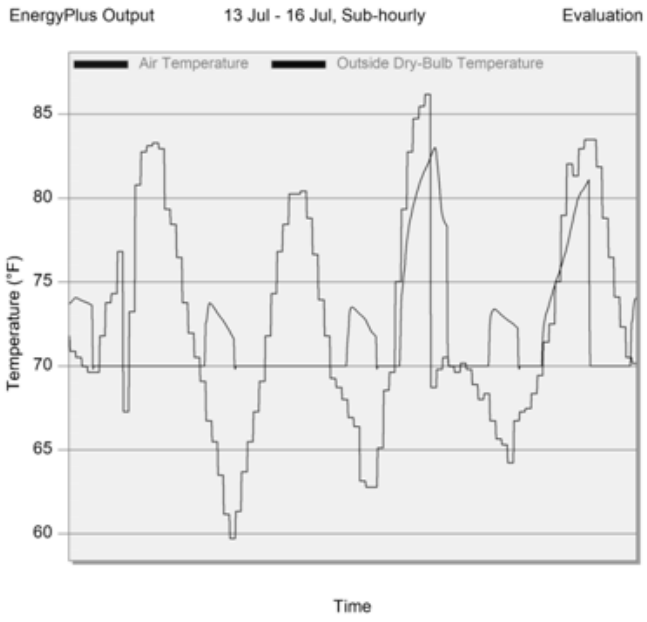


Figure 4.8: DesignBuilder simulator results, guest room, for four typical summer days

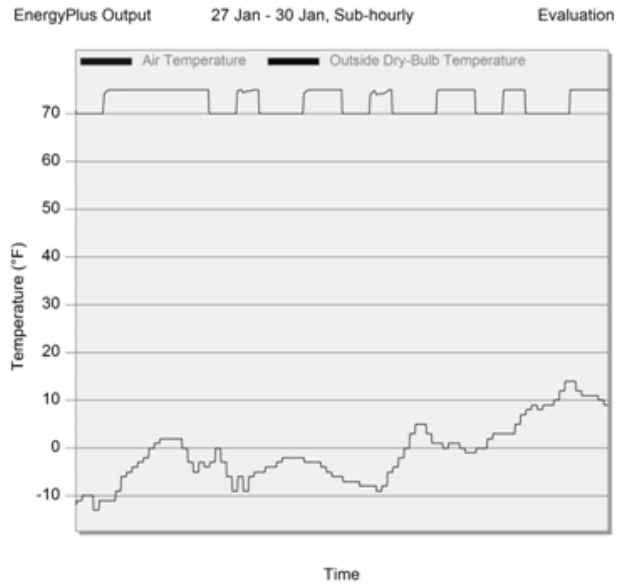


Figure 4.9: DesignBuilder simulator results, guest room, for four typical winter days

It is clear from these two graphs that the comfort concept modeled by the CEMA simulator is different than the comfort concept modeled by DesignBuilder. The CEMA simulator defines comfort to be within a range of temperature, contained within upper and lower limits. The DesignBuilder simulator assumes comfort is attained by maintaining a single temperature value.

While the two concepts achieve their end goal, namely comfort, one system is more costly to maintain than the other. A final observation to state from Figure 4.8: there are periods of time in which the temperature levels that define comfort are breached; that is, comfort is not always attained using the traditional VAV-based HVAC system, whereas from Figure 4.8 it is clear that the concept of comfort is always attained when the CEMA-based HVAC is used.

4.2.3 Financial Gains or Losses Incurred – Summer and Winter

As has been stated before, both concepts of comfort are feasible and attainable, and thus the difference between both comfort philosophies is the cost. Figures 4.10 and 4-11 show the average room temperature and total HVAC energy consumption plots for the whole home, for both summer and winter, respectively (Air Temperature is the Top Plot in the bottom graph):

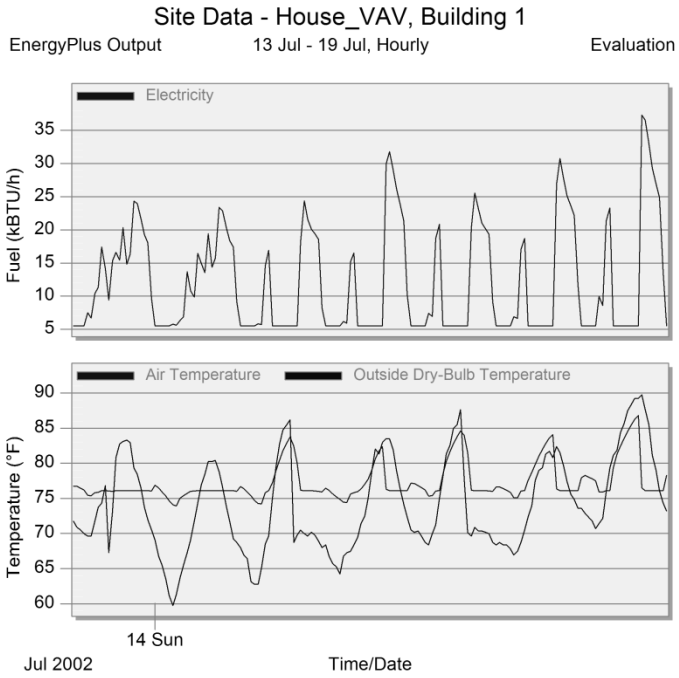


Figure 4.10: Electrical consumption and indoor temperature fluctuations, whole home, summer

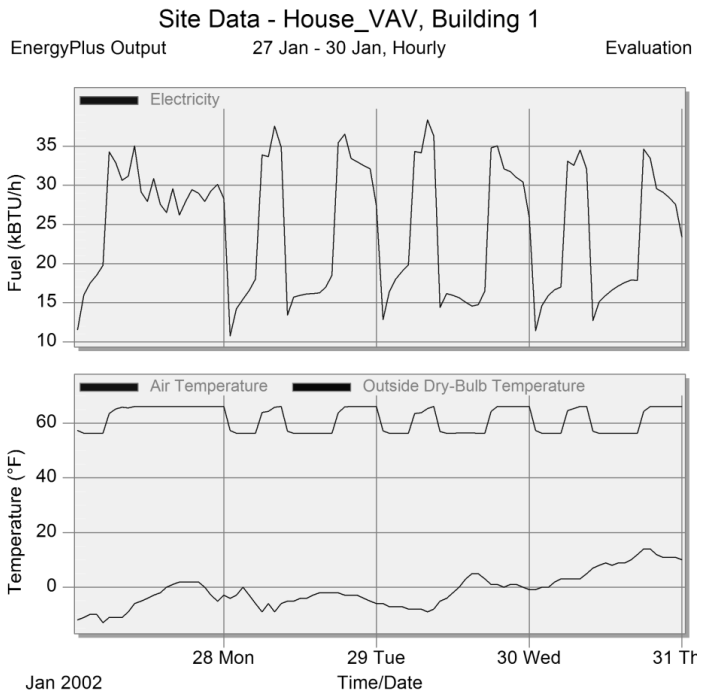


Figure 4.11: Electrical consumption and indoor temperatures fluctuations, whole home, winter

Noting that the sole energy consumer in the DesignBuilder® modeled home is the HVAC system (everything else is turned off), it is clear that the HVAC system is turned on both during the needed temperature increase/decrease (as per the season the simulation is in), and in maintaining the constant comfort temperature. This is evident from the values of electrical consumption when the temperature is steady.

Again noting that the CEMA simulator defines comfort to be within a range of temperatures, the CEMA Simulator has been run for all the rooms, and the total costs of maintaining the home has been directly extracted from the output .csv file. The following equation describes the total electrical costs incurred by the DesignBuilder® home:

$$C = \sum [Energy_{BTU,n}*1000/3413]*2.3/100, \text{ from } n=1 \text{ to } D \quad (4.2)$$

In the above equation, D represents the number of data points in the output graph, the first term represents the value of the data point, in $kBTUhr^{-1}$, the next term changes the units of the first term to kW, and the last term is the price per kilowatt-hour.

For the monetary comparison to be fair, the cost of operating the lighting must be added to the DesignBuilder® simulator, which has been initially excluded for simplicity. Using the lighting assumptions stated in section 3.3.2, the following equation calculates the value added to the total cost of maintaining the DesignBuilder® home.

$$Cost = \sum_{n=1}^{8} W_n * \frac{1}{1000} * T * 2.3 * \frac{1}{100} \quad (4.3)$$

The first two terms from the left denote the wattage of the single lighting device in the zone, in kW. According to the assumptions stated in Chapter 3, the first two terms from the left have a value of 0.1 kW. The third term from the left denotes the time of operation in hours, which from the assumptions stated in Chapter 3, is equal to 4 days*24 hours/day = 96 hours. The last two terms from the left denote the assumed flat price of 2.3 Canadian cents per kilowatt. Thus, the extra cost of maintaining the lighting in the DesignBuilder® modeled home is about 22 Canadian cents per room, or a total of CAD $1.77 for the whole home. Figures 4.12 and 4.13 show the Wattage required, in the CEMA simulator, from the light bulb to reach lighting comfort level for summer and winter respectively in the Hearth Room assuming the comfort level is attained at 80 lux. This simulation run time is 4 days. It has to be noted that the relationship between the output wattage and luminosity for any lighting device is linear, and the lighting control flowchart from Chapter 3:

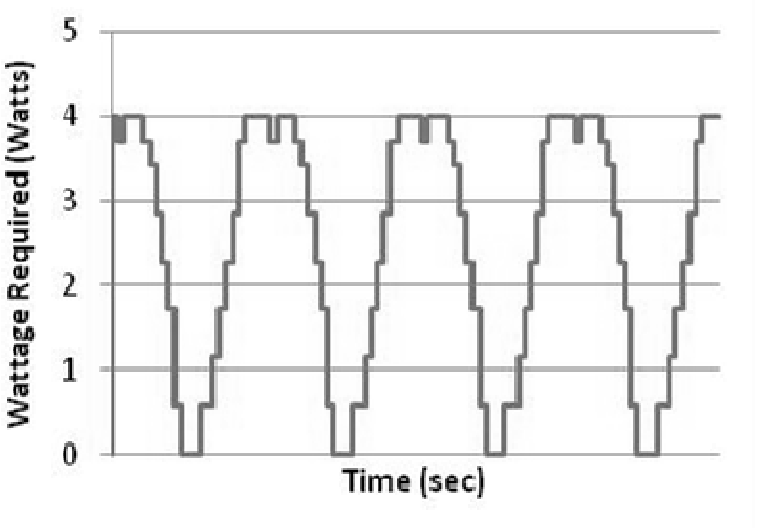


Figure 4.12: Lighting wattage required for comfort, hearth, summer

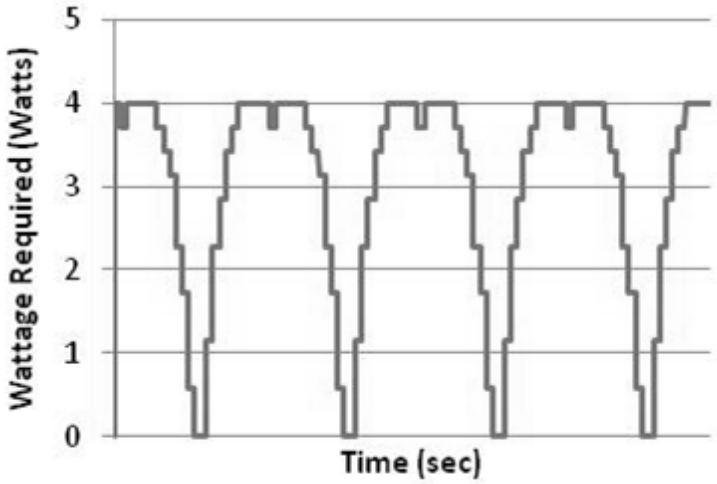


Figure 4.13: Lighting wattage required for comfort, hearth, winter

The fluctuations seen in both graphs reflect the varying amounts of sunlight streaming into the zone, which in turn varies the amount of light wattage required to compensate the light intensity of the sunlight streaming inside the zone, if needed. The light intensity reference values used in the CEMA simulator are variations of the average value of sunlight intensity from [**Kee07**], variations that reflect the differences in light intensity between dawn, noon, dusk, and night, and also between summer and winter. As is seen from both graphs, the maximum value of the needed lighting wattage is a fraction of the total wattage the lighting device is capable of producing, which in turn incurs very significant savings. In fact, using Equation (4.3) and setting W_n= 4

Watts leads to a total cost of only CAD$0.07, which is only 4% of the cost of maintaining the home without lighting control! Even if real life meteorological sunlight intensity values are used in the CEMA simulator, instead of assumed values, the fact remains that when sunlight is used to compensate for light intensity, and when the zone is lit with the needed light intensity comfort value and not with the full power of the lighting device, very significant monetary savings are incurred.

Tables 4-1 and 4-2 show the total operational cost of the simulated home, which is a result of running each simulator 4 times, for four days in the summer and winter seasons respectively:

Cost of Building Operation, Four Summer Days, CAD$		
Iteration	**DesignBuilder®**	**CEMA Simulator**
1	$9.49	$6.42
2	$9.49	$6.27
3	$9.49	$6.31
4	$9.49	$6.27

Table 4.1: Cost of building operation, four summer days, CAD$

Cost of Building Operation, Four Winter Days, CAD$		
Iteration	**DesignBuilder®**	**CEMA Simulator**
1	$17.53	$12.08
2	$17.53	$11.89
3	$17.53	$11.94
4	$17.53	$12.16

Table 4.2: Cost of building operation, four winter days, CAD$

When one looks at the temperature graphs and the two above tables, a conclusion becomes eminent; while both comfort control systems maintain a sense of subjective comfort, the CEMA simulator is always much cheaper to maintain from a financial perspective. It has to be noted that the simulator has been run for four days only; by extrapolation, one would realize that the CEMA simulator can incur even larger monthly savings.

4.2.4 Outdoor Door Modeling and Confidence Interval Results

The correctness of the control system is always a very important issue to deal with in any simulator, and in this thesis this issue is dealt with using the confidence interval concept. The details of this concept and the calculations behind it can be found in Appendix B of this thesis.

As was explained in Section 3.3.2, an outdoor door and its corresponding room is modeled as an M/M/1 queue. For an M/M/1 queue to be stable and not overflow, the incoming rate must be less than the outgoing rate [**Leo04**], and thus $\lambda < \mu$. For M/M/1 queues, the term traffic intensity, denoted by ρ, is defined by the following equation:

$$\rho = \frac{\lambda}{\mu} \tag{4.1}$$

For stability, therefore, ρ must be less than 1. For this reason, the values of λ and μ are given in Section 3.3.2. Further, the average time a person spends in the room is calculated according to the following equation [**Leo04**]:

$$Time\ spent = \left(\frac{\rho}{1-\rho}\right) * \left(\frac{1}{\mu}\right) \tag{4.2}$$

According to the values of μ and λ stated in Section 3.3.2, and according to Equation (4.2), the average time spent in the room comes to 5 hours, which is roughly in line with the amount an average guest stays inside a residence.

To verify the correctness of the CEMA simulator, we vary ρ in increments of 0.2 from 0.2 to 0.8 and measure the total operational cost of the whole home four independent times for each increment, using the CEMA simulator. On these values, the confidence interval is calculated as explained in Appendix B. Figures 4.14 and 4.15 show the resultant interval, for summer and winter respectively, within which we are sure 95% of all independent values of the total operational cost will lie for every ρ:

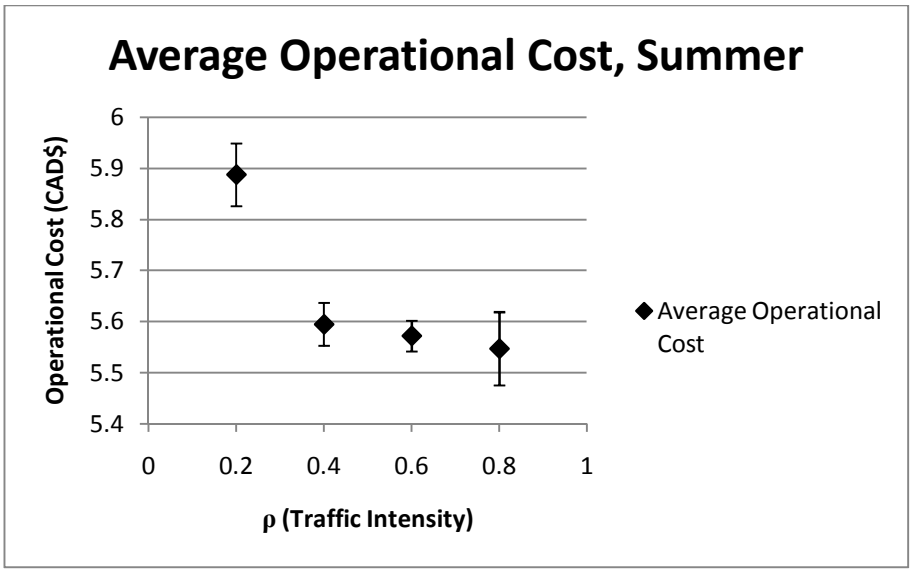


Figure 4.14: Average home operational cost, summer

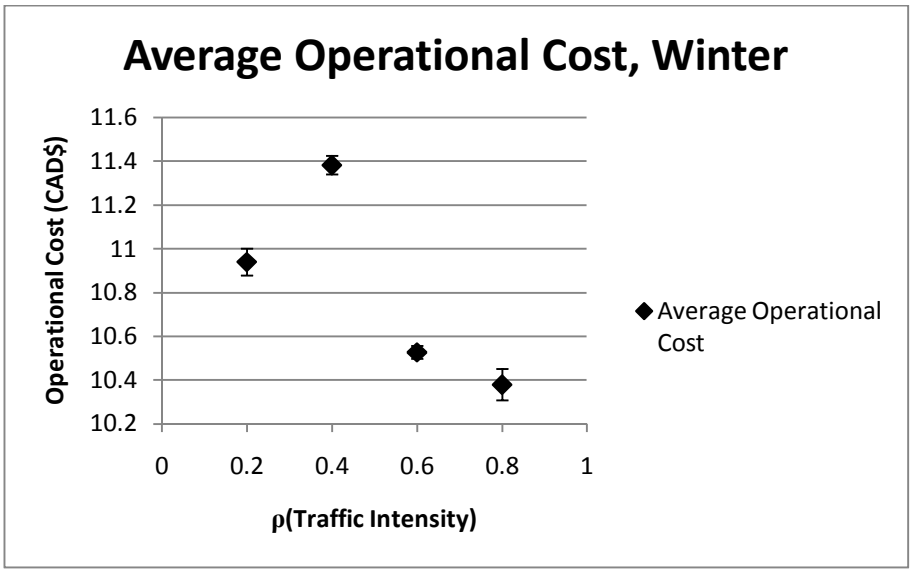


Figure 4.15: Average home operational cost, winter

Numerically, the interval where we are confident that 95% of operational cost readings will lie ranges from 0.18% to 2.33% of the average value of the total operational cost at every ρ. As is knows when performing confidence interval calculations, the smaller the range around an average output value from a simulator, the more correct and the more validated the output of the simulator is [**Wei98**]. As such, the percentage ranges of the confidence intervals of the operational cost of the CEMA simulator do not even come close to the operational cost incurred

by the DesignBuilder simulator. In addition, these percentages prove that the operational cost output of the CEMA simulator is correct and valid, and that the operational cost output of the CEMA simulator is always less than the DesignBuilder operational cost output when given the same set of circumstances.

4.2.5 The Need for a Humidifier/Dehumidifier – Summer and Winter

The last set of outputs to be discussed in this thesis involves the use of an extra piece of equipment for comfort control: humidifiers/dehumidifiers. They provide extra humidification/dehumidification as needed. It has to be noted that it is assumed that humidification and dehumidification can occur from within one device and that this device consumes 500W of power. It is also assumed that the HVAC system by itself, only provides dehumidification. As is seen from Figure 4.3, the value of the relative humidity has no obvious pattern or mathematical equation that can define it, since it is directly dependent on the instantaneous temperature, moisture content, and air pressure. Thus, naturally, inside the room, the relative humidity has the same characteristics. Figures 4.16 and 4.17 show the relative humidity plots when the VAV-HVAC is activated in the CEMA simulator, for summer and winter respectively:

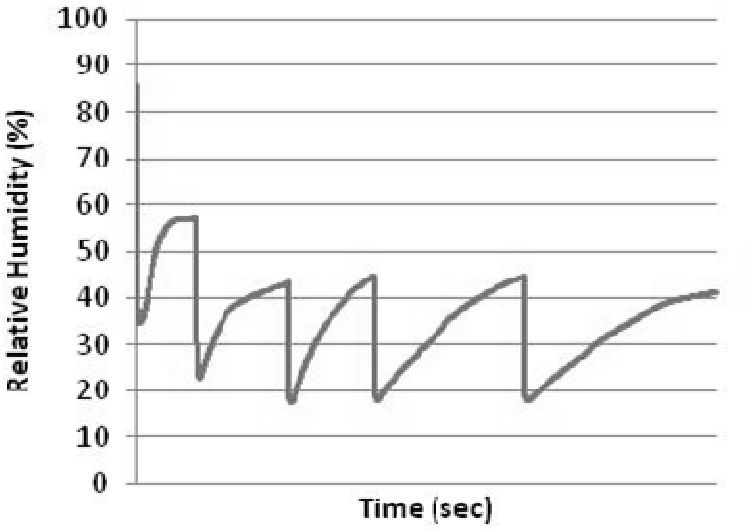


Figure 4.16: Relative Humidity, no extra equipment, guest room, summer

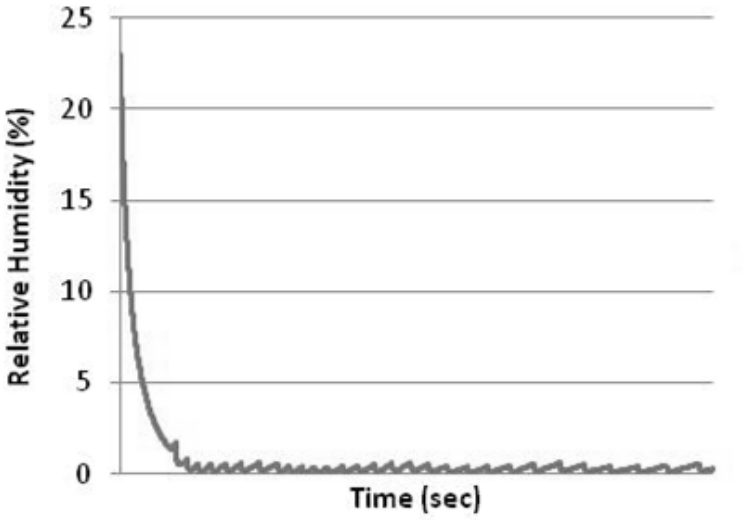


Figure 4.17: Relative Humidity, no extra equipment, guest room, winter

As is seen, due to the inherently high relative humidity of summer, and the inherently low relative humidity of winter, the values for the relative humidity usually fall outside the comfort range as defined in [**ASH82**] and [**Car651**] to be between 30% and 50%. This is especially true in winter. Figures 4.18 and 4.19 show the corresponding plots when the humidifier is used:

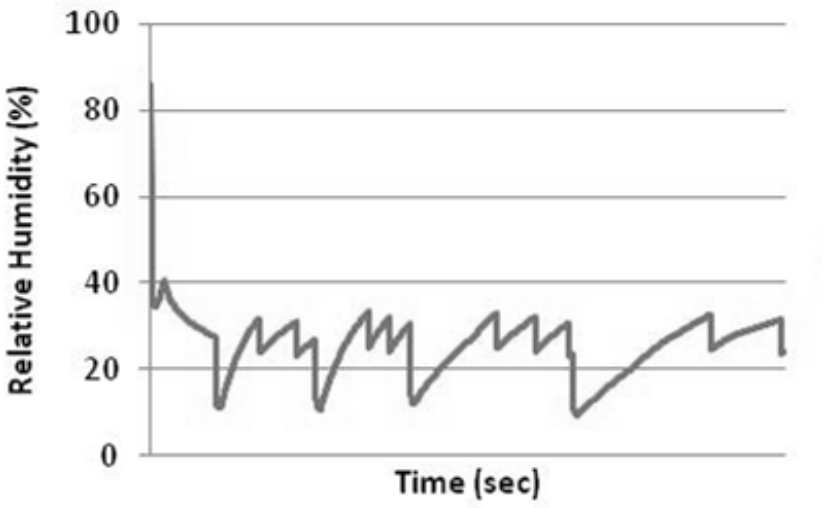


Figure 4.18: Relative Humidity, humidifier/dehumidifier active, guest room, summer

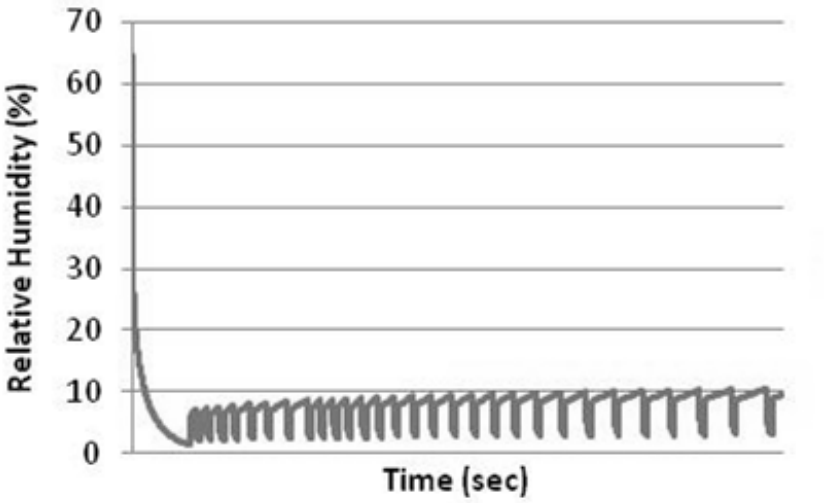


Figure 4.19: Relative Humidity, humidifier/dehumidifier active, guest room, winter

As can be seen, especially in winter, very significant comfort improvements can be achieved, in that the relative humidity starts to slowly increase by time, instead of having near zero levels. While this is not the exact comfort requirements, it still represents an improvement than without a humidifier/dehumidifier. In summer it still represents an improvement however not significant. The reason both plots are out of the comfort range is due to the HVAC always performing dehumidification regardless of the season. As is evident, much more research is warranted into this topic to actually set the terms on which the following tradeoff occurs; how to maintain

comfort in terms of relative humidity while maintaining minimum operational costs (see Chapter 5).

4.3 Conclusion

In this chapter, four types of results have been shown to prove the feasibility and the effectiveness of the CEMA algorithms. The first set is related to highlighting the difference between indoor temperatures with and without a VAV-HVAC system. One room out of eight has been chosen and the simulation always ran for four days only due to Microsoft Excel® limitations. Another room has been chosen to be simulated for the sole purpose of proving that the CEMA simulator functions properly with a different room and a different range of comfort temperatures. These plots showed that the temperature control CEMA algorithm are feasible and do actually keep temperature within comfort levels, although the typical VAV-HVAC system also does so albeit in a different way.

Monetary calculations have then been introduced and tables have been populated to provide estimated operational costs over four days for both types of comfort control systems. As seen in Tables 4-1 and 4-2, The CEMA simulator is clearly much cheaper to operate than the traditional VAV-HVAC system. In addition, integrated lighting control have provided even more savings.

Then, the correctness of the CEMA simulator was explored, by use of the average operational cost for each of the summer and winter cases. For both cases, 95% of the operational cost values fell well below the operational cost of a typical VAV-HVAC system, with the difference between the upper and lower bounds not exceeding 4% in any case. This means that the CEMA algorithms are correct with respect to the average operational cost according to the confidence interval concept.

The concept of adding an integrated humidifier/dehumidifier has then been explored, and it has been seen that while the operational cost does not substantially increase and relative humidity levels are closer to comfort levels; more research is required in the area of the exact relationship, in terms of relative humidity, between an HVAC and a humidifier/dehumidifier.

Chapter 5: Conclusion

5.1 Concluding Remarks

The main goal in this thesis has been to address the pressing issue of the need to provide economical comfort control to homes. Comfort control is a must when any residential structure is designed, yet limited energy resources and an exploding population must be accounted for. Thus, economical comfort control must be achieved. Therefore, the objective of this thesis has been the development of a simple, cheap, easy-to-install, and efficient central energy-management system for residential spaces. This is of course an enormous undertaking, so the focus of this thesis has been the development of a software simulator to simulate the environment of this thesis.

First, Wireless Sensor Networks (WSNs) have been introduced and discussed and examples given to the types of applications that employ them. In addition, an overview of the current challenges facing WSNs has been shown, with current research tackling these challenges. Next, the concept of a smart home and what it is has been discussed and portrayed. Any smart home should have a communication network supporting it. Wireless networks have been deemed the most suitable form of communication due to the fact that any improvement over existing residential HVAC systems must not incur any major residential restructuring or rebuilding, to save costs.

Then, a brief overview of the engineering concepts behind a typical HVAC system has been presented. The fundamental law in HVAC systems is that energy is neither created nor destroyed, and only moves from one area to another. This leads to the fact that heat energy is transferred by conduction, convection, or radiation. This, in turn, leads to the psychometric chart, which describes how moist air behaves under multiple factors when exposed to the three types of energy transfers. In addition, multiple reference books deal with quantifying energy transfer

through myriad of sources inside a typical home [**ASH811**] [**Car65**], and these equations have been tabulated in Appendix C and used in the CEMA simulator. In addition, equations dealing with the electrical consumption of an HVAC system have been drawn up, deduced and explained.

After the CEMA simulator has been coded and run, the results have been obtained and presented. The first set of outputs are related to whether or not the CEMA simulator is able to maintain comfortable temperature levels, and the output revealed that for different zones and different temperature ranges, the CEMA simulator could maintain comfort. The second set of outputs is related to comparison of electrical consumption between the CEMA simulator and the HVAC system of a typical residence, simulated by DesignBuilder. The result is that, under exactly similar circumstances, the operational cost of the CEMA simulator is much less than that of the DesignBuilder simulator.

The next set of outputs has been related to verifying and validating the CEMA simulator outputs (namely the electrical consumption in four days) by performing confidence interval calculations. The resultant confidence interval values and graphs prove the correctness of the CEMA simulator, as the confidence interval values relative to the mean electrical consumption is a fraction of one percent.

Finally, a short investigation related to the tradeoff between the advantages of a humidifier/dehumidifier and its own electrical consumption has been presented. The result of this investigation has been that there are advantages to including a humidifier/dehumidifier, and that there is no significant increase in operational cost.

It is evident from the results of this thesis that economic comfort control is feasible when employing the CEMA system over current operational HVAC systems. There are infrastructure changes required when implementing CEMA using WSNs; for example, wiring the CEMA controller to the lighting system of the residence to be able to control the intensity. However, these changes are much cheaper to implement than the infrastructure changes required to upgrade and replace existing HVAC hardware, and this is the true value of using WSNs in CEMA. In addition, the operational cost savings rendered by CEMA, due to implementing WSNs as the supporting network, makes it a much more attractive option to implement.

5.2 Future Work

The work of this thesis is just the beginning; it only scratches the surface of the topic of economical comfort control. There is much more work to be done. First of all, more research needs to be done on the relationship between an HVAC system and a humidifier/dehumidifier in terms of relative humidity; the tradeoffs in using a humidifier/dehumidifier. The second area in need of further exploration is the implementation of the CEMA simulator in hardware, in order to check and see whether its results will be similar to the CEMA software simulator. This applies to actual HVAC systems, the circuit/circuits that will be used to implement the CEMA algorithms, and any connecting hardware. In addition, the CEMA simulator as it stands does not account for imperfect wireless Zigbee communication that is inevitable in the real world. This critical area has to be tested and analyzed. Another similar area in need to be worked on is the interference between the wireless network that the CEMA system is based on and background traffic, and how it will affect either network load(s). In addition, more CEMA algorithms have to be designed and implemented to cover all area of home automation, security, and control.

More work also must be done in refining the algorithms themselves and the simulator that they are based on. To prove CEMA's correctness, a comparison has been made between a multi-zone, WSN-based comfort control system and a typical one-zone HVAC system. However, a comparison needs to be made between an HVAC system without WSNs and CEMA, using the same upper and lower temperature levels, to further prove CEMA's feasibility. However, it is to be noted that the goal of CEMA is to be able to operate on existing types of HVAC systems, and the prevalent types of HVAC system in Canadian residences is the typical HVAC system with ±1°F around the single comfort temperature.

Finally, another line of research in this area involves the investigation of more complex control mechanisms for the three metrics of comfort used in this thesis, instead of the forced stability mechanism employed here. The implementation of these mechanisms will allow the comfort metrics to stay within the comfort levels even more than what is shown here, and thus incur even more savings.

References

[Als08] N. Alsharabi, L. R. Fa, F. Zing, and M. Ghurab, "Wireless sensor networks of battlefields hotspot: Challenges and solutions," in *6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops*, Berlin, Germany, 2008, pp. 192-196.

[And08] M. Anderson, et al., "MIMO Robust Control for HVAC Systems," *IEEE Transactions on Control Systems Technology*, vol. XVI, no. 3, 2008.

[ASH08] ASHRAE, "HVAC System Analysis and Selection," in *2008 ASHRAE Handbook - Heating, Ventilating, and Air-Conditioning Systems and Equipment*. United States of America: Knovel E-Books, 2008, ch. 1.

[ASH81] ASHRAE, "General Engineering Data," in *ASHRAE 1981 Fundementals*. Atlanta, Unites States of America: American Society of Heating, Refrigerating, and Air Conditioning Engineers Inc., 1981, ch. 2.

[ASH811] ASHRAE, "Load and Energy Calculations," in *ASHRAE 1981 Fundementals*. Atlanta, Unites States of America: American Society of Heating, Refrigerating, and Air Conditioning Engineers Inc., 1981, ch. 4.

[ASH82] ASHRAE, "Air-Conditioning and Heating Applications-Comfort," in *ASHRAE Handbook 1982 Applications.* Atlanta, United States of America: American Society of Heating, Refrigerating, and Air-Conditioning Engineers Inc., 1982, ch. 1.

[ASH821] ASHRAE, "Theory," in *ASHRAE 1981 Fundementals*. Atlanta: American Society of Heating, Refrigerating, and Air Conditioning Engineers Inc., 1981, ch. 1.

[Bai94] G. Baird, M. R. Donn, F. Pool, W. D. S. Brander, and C. S. Aun, *Energy Performance in Buildings*. Boca Raton, Florida, USA: CRC Press, Inc., 1994.

[Beg09] M. S. Beg, B. Fatima, J. Shashank, and R. Muzammil, "Bluetooth Based Implementations for Realising New Consumer Systems," in *International Multimedia, Signal Processing and Communication Technologies, 2009. IMPACT '09*, Aligarh, India, 2009, pp. 250-253.

[Bel081] A. A. Bell, "Equations," in *HVAC - Equations, Data, and Rules of Thumb*. McGraw-Hill, 2008, ch. 3, pp. 19-39.

[Bel082] A. A. Bell, "Conversion Factors," in *HVAC - Equations, Data, and Rules of Thumb*. United

States of America: McGraw-Hill, 2008, ch. 4.

[Cam06] E. Campo, S. Bonhomme, M. Chan, and D. Esteve, "Learning Life Habits and Practices: An Issue to the Smart Home," in *4th International Conference on Smart Homes and Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 355-358.

[CAN09] CANmet Energy. (2009, Mar.) Natural Resources Canada's CANMETenergy. [Online]. http://canmetenergy-canmetenergie.nrcan-rncan.gc.ca/eng/software_tools/hot3000.html

[Car65] Carrier Air Conditioning Company, "Load Estimating," in *Handbook of Air Conditioning System Design*. Unites States of America: McGraw-Hill Book Company, 1965, ch. 1.

[Car651] Carrier Air Conditioning Company, "All-Air Systems," in *Handbook of Air Conditioning Design*. Unites States of America: McGraw-Hill Book Company, 1965, ch. 10.

[Che08] J. Herbert, J. O'Donoghue, and X. Chen, "A Context-Sensitive Rule-based Architecture for a Smart Building Environment," in *Second International Conference on Future Generation Communication and Networking*, Sanya, Hainan Island, China, 2008, pp. 437-440.

[Chu07] X. Chu and R. Buyya, "Service Oriented Sensor Web," in *Sensor Networks and Configuration - Fundementals, Standards, Platforms, and Applications*, N. P. Mahalik, Ed. Springer, 2007, ch. 3, pp. 50-74.

[Dar] M. J. Darre. Psychometric Chart Use. Picture.

[Dar08] M. Darianian and M. P. Michael, "Smart Home Mobile RFID-Based Internet-of-Things Systems and Services," in *International Conference on Advanced Computer Theory and Engineering*, Phuket, Thailand, 2008.

[DeC10] K. De Craemer and G. Deconinck. (2010, Apr.) Lirias: Item 123456789/265822. [Online]. https://lirias.kuleuven.be/bitstream/123456789/265822/1/SmartMeteringCommStandards.pdf

[Des10] DesignBuilder Software Ltd. (2010, May) DesignBuilder - Building design, simulation and visualisation - Building Simulation... Made Easy. [Online]. http://www.designbuilder.co.uk/

[Fis01] G. S. Fishman, "Simulation in Prespective," in *Discrete-Event Simulator - Modeling, Programming, and Analysis*, P. Glynn and S. M. Robinson, Eds. New York, New York, Unites States of America: Springer Series in Operations Reserach, 2001, ch. 1, pp. 3-35.

[Foo06] V. F. S. Fook, E. Hao, S. Takahashi, and A. Aung, "Fiber Bragg Grating Sensor System for Monitoring and Handling Bedridden Patients," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 239-246.

[Gay06] V. Gay and P. Leijdekkers, "Around the Clock Personalized Heart Monitoring Using Smart

Phones," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 82-89.

[Gil07] R. Giladi, "Heterogeneous Building Automation and IP Networks Management," in *24th International Conference on Distributed Computing Systems Workshops*, Toronto, Ontario, Canada, 2007, pp. 636-641.

[Goh07] H. G. Goh, M. L. Sim, and H. T. Ewe, "Agriculture Monitoring," in *Sensor Networks and Configuration : Fundamentals, Standards, Platforms, and Applications*. Springer, 2007, ch. 21, pp. 439-462.

[Guo06] Q. Guo, "Availability-Constrained Shared Backup Path Protection for GMPLS-Based Spare Capacity Reconfiguration," Masters Thesis, OCIECE, University of Ottawa, Ottawa, 2006.

[Jai06] G. Jain, D. J. Cook, and V. Jakkula, "Monitoring Health by Detecting Drifts and Outliers for a Smart Environment Inhabitant," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 114-121.

[Jon94] W. P. Jones, *Air Conditioning Engineering*, 4th ed. London, Great Britain: Hodder Headline Group, 1994.

[Kas05] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newmann, "Communication Systems for Building Automation and Control," *Proceedings of the IEEE*, vol. XCIII, no. 6, pp. 1178-1203, Jun. 2005.

[Kee07] T. J. Keefe. (2007, Feb.) Community College of Rhode Island. [Online]. http://www.ccri.edu/physics/keefe/light.htm

[Kim07] D.-S. Kim and S. Y. Shin, "Wireless Pet Dog Managment Systems," in *Sensor Networks and Configuration : Fundamentals, Standards, Platforms, and Applications*, N. P. Mahalik, Ed. Springer, 2007, ch. 20, pp. 422-438.

[Kom09] M. Komareji, et al., "Simplified Optimal Control in HVAC Systems," in *IEEE Control Applications, (CCA) & Intelligent Control, (ISIC), 2009*, Saint Petersburg, Russia, 2009, pp. 1033-1038.

[Kou07] A. Kouba, M. Alves, and E. Tovar, "Time Sensitive IEEE 802.15.4," in *Sensor Networks and Configuration - Fundementals, Standards, Platforms, and Applications*, N. P. Mahalik, Ed. Springer, 2007, pp. 19-49.

[Krc07] S. Krco, et al., "Mobile Network Supported Wireless Sensor Network Services," in *IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2007*, Pisa, Italy, 2007, pp. 1-3.

[Lee06] B. Lee and H. Kim, "Design and Implementation of a Secure IBS platform using RFID and Sensor Network," in *IEEE Tenth International Symposium on Consumer Electronics (2006)*, St. Petersburg, Russia, 2006, pp. 1-4.

[Lee08] M.-h. Lee, K.-b. Eom, H.-j. Kang, C.-s. Shin, and H. Yoe, "Design and Implementation of Wireless Sensor Network for Ubiquitous Glass Houses," in *Seventh IEEE/ACIS International Conference on Computer and Information Science, 2008.*, Paris, France, 2008, pp. 397-400.

[Lee09] L. Leemis and S. Park. (2009, Sep.) Homepage of CS 426/526 --- Simulation --- Fall 2009 . [Online]. http://www.cs.wm.edu/~esmirni/Teaching/cs526/DESAFC-1.1.pdf

[Leo04] A. Leon-Garcia and I. Widjaja, "Peer-to-Peer Protocols and Data Link Layer," in *Communication Networks - Fundemental Concepts and Key Architecture*, S. W., Ed. New York, United States of America: McGraw-Hill, 2004, ch. 5, pp. 342-345.

[Lid07] C. Liding, Z. Ming, and Z. Bugong, "Research and Design of Intelligent Building Integrating Software Platform Based on Web," in *IEEE International Conference on Control and Automation (2007)*, Guangzhou, China, 2007, pp. 68-73.

[Lit07] C. Liting, J. Tian, and W. Jiang, "Information Fusion Technology and Its Application to Fire Automatic Control System of Intelligent Building," in *Proceedings of the 2007 International Conference on Information Acquisition*, Jeju Island, South Korea, 2007, pp. 445-450.

[Mac08] R. MacRuairi, M. T. Keane, and G. Coleman, "A Wireless Sensor Network Application Requirements Taxonomy," in *Second International Conference on Sensor Technologies and Applications*, Cap Esterel, France, 2008, pp. 209-216.

[Mah07] N. P. Mahalik, "Introduction," in *Sensor Networks and Configuration - Fundementals, Standards, Platforms, and Applications*, N. P. Mahalik, Ed. Springer, 2007, ch. 1, pp. 1-18.

[Man01] A. Manjeshwar and D. P. Agrawal, "TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks," in *15th International Parallel and Distributed Processing Symposium*, San Fransisco, California, 2001, pp. 2009-2015.

[McH06] M. McHugh and A. F. Smeaton, "Event Detection Using Audio in a Smart Home Context," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 39-46.

[Men] M.-S. Pan, L.-W. Yeh, Y.-A. Chen, Y.-H. Lin, and Y.-C. Tseng, "Design and Implementation of a WSN-Based Intelligent Light Control System," in *28th International Conference on Distributed Computing Systems Workshops*, Beijing, China, 2008, pp. 321-326.

[Mik00] M. Newman. (2000) BACnet, A Tutorial Overview. Online Presentation,

http://www.bacnet.org/Tutorial/HMN-Overview/sld001.htm.

[Mor07] P. A. Morreale, "Wireless Sensor Network Applications in Urban Telehealth," in *21st International Conference on Advanced Information Networking and Applications Workshops, 2007*, Niagara Falls, Ontario, Canada, 2007, pp. 810-814.

[Mor93] A. Moriaki and H. Fujiyoshi, "A Systematic Approach to Intelligent Building Design," *IEEE Communication Magazine*, vol. XXXI, no. 10, pp. 46-48, Oct. 1993.

[Nat09] National Resources Canada. (2009, Aug.) Intelligent Buildings. CANMETenergy. [Online]. http://canmetenergy-canmetenergie.nrcan-rncan.gc.ca/eng/buildings_communities/buildings/intelligent_buildings.html

[ODo06] J. O'Donoghue, J. Herbert, and P. Stack, "Remote Non-Intrusive Patient Monitoring," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 180-187.

[Oud06] J.-P. Oudet and P. Mabilleau, "In-motion Detection of Daily Life Activity Using Wireless Networked Smart Sensors," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 223-230.

[Pan] M.-S. Pan and Y.-C. Tseng, "Zigbee and Their Applications," in *Sensor Networks and Configuration : Fundamentals, Standards, Platforms, and Applications*, N. P. Mahalik, Ed. Springer, ch. 16, pp. 349-369.

[Pap09] N. Papadopoulos, A. Meliones, D. Economou, I. Karras, and I. Liverezas, "A Connected Home Platform and Development Platform for Smart Home Control Applications," in *7th IEEE International Conference on Industrial Informatics*, Cardin, UK, 2009, pp. 402-409.

[Par07] H. Park, J. Burke, and M. B. Srivastava, "Design and Implementation of a Wireless Sensor Network for Intelligent Light Control," in *6th International Symposium on Information Processing in Sensor Networks, 2007*, Cambridge, Massachusetts, 2007, pp. 370-379.

[Par071] T. J. Park, Y. J. Chon, D. K. Park, and S. H. Hong, "BACnet over ZigBee, A new approach to wireless datalink channel for BACnet," in *2007 5th IEEE International Conference on Industrial Informatics*, vol. I, Vienna, Austria, 2007, pp. 33-38.

[Pen09] H. Pensas, H. Raula, and J. Vanhala, "Energy Efficient Sensor Network with Service Discovery for Smart Home Environments," in *Third International Conference on Sensor Technologies and Applications, 2009*, Athens/Glyfada, Greece, 2009, pp. 399-404.

[Sha07] T. Shan, B. C. Lovell, and S. Chen, "Intelligent CCTV via Planetary Sensor Network," in *Sensor Networks and Configuration : Fundamentals, Standards, Platforms, and Applications.*

Springer, 2007, ch. 22, pp. 463-484.

[Sma09] Smart Meters Ontario. (2009) SMART METERS : What Would TOU Pricing Look Like Today?. [Online]. http://www.smartmetersontario.ca/index.cfm?page=TOUPricingToday

[Tah09] H. Tahir and M. Y. Javed, "Service Guarantees in Wireless Sensor Networks," in *International Conference on Emerging Technologies, 2009*, Islamabad, Pakistan, 2009, pp. 433-436.

[Tak09] Y. Takagi, K. Yonezawa, N. Nishimura, Y. Hanada, and K. Yamazaki, "Next Generation HVAC System for Office Buildings - The Optimal Control Structure for DHC Buildings," in *ICCAS-SICE*, Fukuoka, Japan, 2009, pp. 2002-2007.

[Tam06] T. Tamura, "A Smart House for Emergencies in the Eldery," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 7-11.

[Uma] M. Umar, U. Adeel, and S. Qamar, "Routing Protocols," in *Sensor Networks and Configuration - Fundementals, Standards, Platforms, and Applications*, N. P. Mahalik, Ed. Springer, 2007, ch. 7, pp. 143-166.

[Uzu07] Y. Uzunay and K. Bicakci, "SHA: A Secure Voice Activated Smart Home for Quadraplegia Patients," in *IEEE International Conference on Bioinformatics and Biomedicine Workshops*, San Jose, California, USA, 2007, pp. 151-158.

[Val06] M. A. Valero, et al., "A Multiagent Architectural Framework for Smart Homes to provide i-Care Services Supported by Decision Taking Procedures," in *4th International Conference on Smart Homes and Health Telematics*, Belfast, Northern Ireland, UK, 2006, pp. 363-366.

[Vip05] S. Vipul, A. Krause, C. Guestrin, J. H. Garrett Jr., and H. S. Matthews, "Intelligent Light Control using Sensor Networks," in *3rd International Conference on Embedded Networked Sensor Systems*, San Diego, California, 2005, pp. 218-229.

[Wan08] H. Wang, "A Robust Mechanism for Wireless Sensor Network Security," in *4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008*, Dalian, China, 2008.

[Wei98] R. M. Weiers, "Estimation from Sampling Data," in *Introduction to Business Statistics*. United States of America: Duxbury Press, 1998, ch. 8, pp. 273-309.

[Yam08] M. Yamaji, Y. Ishii, T. Shimamura, and S. Yamamoto, "Wireless sensor network for industrial automation," in *5th International Conference on Networked Sensing Systems*, Kanazawa, Japan, 2008, pp. 253-253.

[Yim07] Z. Yiming, Y. Xianglong, G. Xishan, Z. Mingang, and W. Liren, "A Design of Greenhouse Monitoring & Control System Based on ZigBee Wireless Sensor Network," in *International*

Conference on Wireless Communications, Networking and Mobile Computing, 2007, Shanghai, China, 2007, pp. 2563-2567.

[Zha07] Y. Zhang, L. Wang, and G. Zhenbin, "The Application of Wireless Sensor Networks with Multi-agent Systems," in *IET Conference on Wireless, Mobile and Sensor Networks, 2007*, Shanghai, China, 2007, pp. 197-200.

[Zha08] J. Zhang and V. Varadharajan, "A New Security Scheme for Wireless Sensor Networks," in *IEEE Global Telecommunications Conference, 2008*, New Orleans, Louisiana, USA, 2008.

[Zhe06] L. Zheng, "ZigBee Wireless Sensor Network in Industrial Applications," in *International Joint Conference SICE-ICASE, 2006*, Bexco, Busan, Korea, 2006, pp. 1067-1070.

[Zia06] T. Zia and A. Zomaya, "Security Issues in Wireless Sensor Networks," in *International Conference on Systems and Networks Communications, 2006*, Tahiti, French Polynesia, 2006.

[Zig06] Zigbee Alliance. (2006) Zigbee Specifications. [Online]. www.zigbee.org

Appendix A: CEMA Simulator Code

“decision_Handling_Module:”

```
/*******************************************************************************************
FILE: buildingConstruct.java
This class contains the implementations of the CEMA algorithms, in additon to functions
that initialize the simulation itself and allows the user to view the output data
IN: User inputs of simulation time, HVAC Type, Simulation's seaon, and whether a
humidifier is available
OUT: a output file, after the simulation has run completly, detailing the temperatures,
relative humidities, and lighting device wattage needed for maintaining "comfort"
*******************************************************************************************/



package decision_Handling_Module;


//list of imports needed for the simulation to work, including other packages in this project
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import simulator_Backbone.*;
import Queue_Element_Types.*;
import misc_Operations.*;
import input_Output_Manipulation.*;

public class buildingConstruct implements SharedConstants{

	double currentTime = 0;					//the simulator's timer
	int currentNumberOfDays = 0;	// the equivalent current number of seconds, used in HVAC load calculations
	double simulationTime;					//total simulation time
	seasonType TheSeason = null;	//the simulation's season
	float requiredLightIntensity = 0;

	Room Building[] = null;					//the building itself, consisting of an array of rooms
	DynamicEvent nextEvent;					//the current event being pulled from the queue and executed

	int roomID;							//the room ID of the event being processed
	float OutdoorTempInput;			//in Farenheit
	float OutdoorHumidityRatio;	//in grains/lb

	//ALL THE TYPES OF EVENTS BEING COVERED/////////
	DoorOpen doorOpenEvent;
	DoorClose doorCloseEvent;
	MeasureRoomVariables measureEvent;
	HVAC_Running Activate_HVAC;
	HVAC_Deactivation Deactivate_HVAC;
	normal_Temperature_Increase_Interval Temp_Increase;
	adjust_Outside_Conditions Adjust_Conditions;
	ExchangePeopleBetweenRooms TheExchange;
	HumidifierOn HumidifierActive;
	HumidifierOff HumidifierInactive;
	////////////////////////////////////////////

	//the following three booleans describing the state and type of the HVAC system
```

```
boolean HVAC_Running = false;
boolean VAV = false;
boolean HumidifierBased = true;

//the follolwing three variables describe stages in the heating/cooling load calculation
double CFM_DA = 0;
float Outside_Enthalpy = 0;
float Indoor_Enthalpy = 0;

//the following three variables detail the level of randomness in the rate of people enterting, staying, and leaving a room
double PoissionLamda = 0.1;
double ExponentialLambda = 0.2;
RandomNumberGenerators RandomValues = new RandomNumberGenerators(PoissionLamda, ExponentialLambda);

//this variable is an instance in a class that calculates all needed psychometric equations
psychrometricFunctions ThePsychrometricChart = new psychrometricFunctions();

CSV_Parser TheCSVParser = new CSV_Parser();          //for handling the output file itself (in the constructor of this class, all input files are read and parsed)
//for output manipulation when writing to the output file
float tempOutputTemp;
float tempOutputRH;

int ZoneCounter = 0;  //for use when initiating the room array (i.e: the building)

//for use in temperature movement when HVAC is not operating
float y_intercept = 0;
float y_intercept_Moisture_Content = 0;
float x_intercept = 0;
float T_mixed = 0;
float T_adp = 0;

//the even queue itself, whose size was arbitrarily set
EventQueue queue = new EventQueue(100000);

public buildingConstruct(double finalSimulationTime, boolean Humidifier, String SeasonType, boolean VAVSelected, boolean HumidifierSelected) throws Exception{

                //setting user inputs to their respective variables
                simulationTime = finalSimulationTime;
        HumidifierBased = Humidifier;
        TheCSVParser.ReturnParsedValues();
        VAV = VAVSelected;
                if(SeasonType == "Summer"){
                        TheSeason = seasonType.summer;
                }
                else if (SeasonType == "Fall"){
                        TheSeason = seasonType.fall;
                }
                else if (SeasonType == "Winter"){
                        TheSeason = seasonType.winter;
                }
                else if(TheSeason == seasonType.spring){
                        TheSeason = seasonType.spring;
                }
                //initiating the output file
                TheCSVParser.InitiateOutputFile("This is the output file containing all the data readings for all the rooms.");

                //setting the number of rooms in the home according to the user-inputted file
                Building = new Room[ZonesProperties.size()];

                //setting each room's properties on its own according to the assumptions stated in Chapter 3
                for(float[] TempRoomProperties: ZonesProperties){
                        if(ZoneCounter <= ZonesProperties.size()-1){
                                if((TheSeason == seasonType.summer)||(TheSeason == seasonType.spring)){
                                        Building[ZoneCounter] = new Room(ZoneCounter, (float)
(TypicalOutsideTemperaturesSummer[0]+15), (float)
psychrometricFunctions.PerformRHtoGRFunction((float)(TypicalOutsideTemperaturesSummer[0]+15),(float)
(TypicalOutsideRelativeHumiditySummer[0]/100),230), (float) TypicalLightIntensitiesSummer[0], (float)
(TypicalOutsideTemperaturesSummer[0]), (float)
```

```
psychrometricFunctions.PerformRHtoGRFunction((float)(TypicalOutsideTemperaturesSummer[0]),(float)
(TypicalOutsideRelativeHumidityWinter[0]/100),230), (float) TypicalLightIntensitiesSummer[0],TheSeason,currentTime);
                                                }else{
                                                        Building[ZoneCounter] = new Room(ZoneCounter, (float)
(TypicalOutsideTemperaturesWinter[0]+15), (float)
psychrometricFunctions.PerformRHtoGRFunction((float)(TypicalOutsideTemperaturesWinter[0]+15),(float)
(TypicalOutsideRelativeHumidityWinter[0]/100),230), (float) TypicalLightIntensitiesWinter[0], (float) (TypicalOutsideTemperaturesWinter[0]),
(float) psychrometricFunctions.PerformRHtoGRFunction((float)(TypicalOutsideTemperaturesWinter[0]),(float)
(TypicalOutsideRelativeHumidityWinter[0]/100),230), (float) TypicalLightIntensitiesWinter[0],TheSeason,currentTime);
                                                }
                                                if(TempRoomProperties[20] == 1){
                                                        Building[ZoneCounter].SetDoor();
                                                        queue.AddNewJob(new
DoorOpen(currentTime,doorAction.personIn,ZoneCounter));
                                                }
                                                //each room's variables are measured every twenty seconds and it is assumed people
randomly move between rooms
                                                queue.AddNewJob(new MeasureRoomVariables(12,ZoneCounter));
                                                queue.AddNewJob(new ExchangePeopleBetweenRooms(currentTime+ZoneCounter,
ZoneCounter));
                                                ZoneCounter++;
                                        }
                                }
                                //the outside conditions are assumed to change every hour (see the "adjust_Outisde_Conditions" event
                                queue.AddNewJob(new adjust_Outside_Conditions((currentTime+3600)));
        }//end of constructor


        //The method that takes each element in the queue and parses it, and acts accordingly
        @SuppressWarnings("static-access")
        public void InitiateSimulation() throws IllegalAccessException{

                //the simulation progresses until the arrival time of the new event exceeds the required simulation time

                while(currentTime <= simulationTime){

                        //the next lines (before the "if"'s) describe how an event is extracted, its arrival time taken and spread across all
the rooms of the building
                        nextEvent = queue.getEarliestJob();
                        queue.RemoveEarliestJob();
                        currentTime = nextEvent.getArrivalTime();
                        for (int i=0; i<=Building.length-1; i++){
                                Building[i].TheCurrentTime = currentTime;
                        }
                        //*************************************************************************
                        //The next if and else if statements detail what happen when the extracted
                        //event is one of 10 types of events that cover all aspects of HVAC and
                        //lighting control.
                        //*************************************************************************
                        //A door of room "i" opens....
                        if(nextEvent instanceof DoorOpen){
                                doorOpenEvent = (DoorOpen)nextEvent;
                                doorAction currentAction = doorOpenEvent.getDoorAction();
                                roomID = doorOpenEvent.GetRoomID();
                                Building[roomID].getDoor().setDoorStatus(open);
                                queue.AddNewJob(new DoorClose((currentTime+30),roomID));    //when the same door is closed
(after 30 seconds of the same door opening)


                                //when a person comes into room "i" from the outside, the system randomly decides how long this
person stays (exponentially) and when the next person comes in (using a poisson distribution)
                                if(currentAction == doorAction.personIn){
                                        Building[roomID].incrementCurrentNumberOfPeople();
                                        queue.AddNewJob(new
DoorOpen(currentTime+(RandomValues.ReturnPoissionVariate()*60*60),doorAction.personIn,roomID));                 //next person in
                                        queue.AddNewJob(new
DoorOpen(currentTime+(RandomValues.ReturnExponentialVariate()*60*60),doorAction.personOut,roomID));             //when this person out
                                }else{//when a person comes out, the number of people in the room decreases
                                        Building[roomID].decrementCurrentNumberOfPeople();
                                }
```

```
//A door of room "i" closes....
}else if (nextEvent instanceof DoorClose){
	//the same actions occur at the beginning of every event; the roomID and event execution time are extracted.
	doorCloseEvent = (DoorClose) nextEvent;
	roomID = doorCloseEvent.getRoomID();
	currentTime = nextEvent.getArrivalTime();
	//the only thing that happens when a door closes in the decrementation of the number of people in room "i"
	if(Building[roomID].getDoor().isDoorOpen() == open){
		Building[roomID].getDoor().setDoorStatus(closed);
	}

//Room "i" 's room temperature, RH and light intensity are measured and acted upon
}else if(nextEvent instanceof MeasureRoomVariables){
	//start of the event's parsing
	measureEvent = (MeasureRoomVariables) nextEvent;
	roomID = measureEvent.getRoomID();
	currentTime = nextEvent.getArrivalTime();
	//the variables are recorded
	measureEvent.changeRoomTemperature(Building[roomID].getRoomTemperature());
	measureEvent.changeMoistureRoomContent(Building[roomID].getRoomHumidity());
	measureEvent.changeLightIntensity(Building[roomID].getRoomLightIntentsity());
	//The instantaneous outputs are written to the output CSVfile
	tempOutputTemp = (measureEvent.getInstantRoomTemperature()-32)*5/9;
	tempOutputRH = ThePsychrometricChart.PerformGRtoRHFunction(measureEvent.getInstantRoomTemperature(), measureEvent.getInstantRoomMoistureContent(), 230);
	if(Building[roomID].isHVACOn() == true){
		TheCSVParser.WriteDataLine(new double [] {roomID,currentTime, tempOutputTemp, tempOutputRH, Building[roomID].RequiredLightWattage, Building[roomID].costOfRunningTheRoom,CalculateTotalBuildingCost(), ((Building[roomID].getOutsideTemperature()-32)*5/9), 1});
	} else if(Building[roomID].isHVACOn() == false){
		TheCSVParser.WriteDataLine(new double [] {roomID,currentTime, tempOutputTemp, tempOutputRH, Building[roomID].RequiredLightWattage, Building[roomID].costOfRunningTheRoom,CalculateTotalBuildingCost(), ((Building[roomID].getOutsideTemperature()-32)*5/9), 0});
	}
	tempOutputTemp = 0;
	tempOutputRH = 0;

	//the next variable measurement is scheduled 12 seconds later
	queue.AddNewJob(new MeasureRoomVariables(currentTime+12,roomID));

	//Here a decision whether to activate the HVAC or not is made
	//if the temperature is out of the requested lower and higher temperature range, and the HVAC is off, the HVAC is activated
	if(((Building[roomID].getRoomTemperature()-Building[roomID].getNeededUpperTemperature() > 0.1)||(Building[roomID].getRoomTemperature()- Building[roomID].getNeededLowerTemperature() < -0.1))&&(Building[roomID].isHVACOn() == false)){
		Building[roomID].HVAC_Activated();
		//the method of calculating the needed HVAC wattage differs according to type of HVAC
		if(VAV){
			Building[roomID].CalculateSensibleHeatLoad();
			Building[roomID].CalculateLatentHeatLoad();
			Building[roomID].EnergyUsed_HVAC = Math.abs(Building[roomID].getTotalLatentHeat()+Building[roomID].getTotalSensibleHeat());
			Building[roomID].EnergyUsed_HVAC  = Math.abs(Building[roomID].EnergyUsed_HVAC/(13000));	//from BTU/hr to needed wattage of the cooling/heating device assuming SEER of 13 and COP of 3.809

		}else{
			//the dehumidified air quantity equation 3-1 in Chapter 3 of my thesis report; the next two equations are in Appendix C
			CFM_DA = (((Building[roomID].getVentilation()*(Building[roomID].getRoomTemperature()+1))-(CalculateTotalSensibleHeatLoad()/1.08))/(Building[roomID].getOutsideTemperature()-(2*Building[roomID].getRoomTemperature())));
			T_mixed = (float) ((Building[roomID].getRoomTemperature()*(Building[roomID].getVentilation()/CFM_DA))+(Building[roomID].getOutsideTemperature()*(((CFM_DA)-(Building[roomID].getVentilation()))/CFM_DA)));
```

```
                                                T_adp = T_mixed-Building[roomID].getRoomTemperature();
                                                //enthalpy calculations used for calculating the wattage (Indoor means
Dewpoint)
                                                Outside_Enthalpy = (float) (0.24*T_mixed); //specific heat capacity of air
                                                Indoor_Enthalpy = (float) (0.24*T_adp);                 //specific heat
capacity of air
                                                Building[roomID].EnergyUsed_HVAC = 4.45*(Outside_Enthalpy-
Indoor_Enthalpy)*CFM_DA;
                                                Building[roomID].EnergyUsed_HVAC =
Math.abs(Building[roomID].EnergyUsed_HVAC/(13000));                     //from BTU/hr to needed wattage of the cooling/heating device
assuming SEER of 13 and COP of 3.809
                                                CFM_DA=0;
                                        }
                                        //the next event to check whether the HVAC keeps working or stop is the next second
                                        queue.AddNewJob(new
HVAC_Running(currentTime+1,roomID,measureEvent.getInstantRoomTemperature(), measureEvent.getInstantRoomMoistureContent(),
currentTime));
                                }
                                // this section deals with the light intensity calculations
                                // if the lights are ON, then the cost of operating them for 12 seconds is included in the room's cost
counter
                                if(Building[roomID].lightsOff == false){
                                        Building[roomID].costOfRunningTheRoom +=
(2.3*12/3600*Building[roomID].RequiredLightWattage/1000);
                                }
                                //if outside light intensity cannot provide for the total needed light intensity
                                if((Building[roomID].getOutsideLightIntensity()-Building[roomID].getNeededLightIntensity()) <=
0){
                                        Building[roomID].lightsOff = false;
                                        Building[roomID].ShadesOn = false;
                                        //The required light intensity is defined to be the difference between the needed and
outdoor light intensities
                                        requiredLightIntensity = -1*(Building[roomID].getOutsideLightIntensity()-
Building[roomID].getNeededLightIntensity());
                                        //The sole light source, an incadescent light bulb, is 100W and provides 1750 lumens
                                        Building[roomID].RequiredLightWattage = requiredLightIntensity/1750*100;

        Building[roomID].setRoomLightIntensity(requiredLightIntensity+Building[roomID].getOutsideLightIntensity());
                                        requiredLightIntensity = 0;

                                } else{//the incoming light intensity is more than what is asked for, and thus the lights are off and the
shades activate...
                                        Building[roomID].lightsOff = true;// will not calculate light bulb contribution to total cost
of maintenance
                                        requiredLightIntensity = 0;
                                        Building[roomID].RequiredLightWattage = 0;
                                        Building[roomID].ShadesOn = true;
                                        Building[roomID].setRoomLightIntensity(Building[roomID].getNeededLightIntensity());
                                }
                                //if there is a humidifier/dehumidifier and it's not working, activate it
                                if((HumidifierBased == true)&&(Building[roomID].IsHumidifierOn() ==
false)&&((Building[roomID].getRoomHumidity()-Building[roomID].getNeededLowerMoistureContent() < -
0.1)||(Building[roomID].getRoomHumidity()- Building[roomID].getNeededUpperMoistureContent() > 0.1))){//(((((TheSeason ==
seasonType.fall || TheSeason == seasonType.winter)&&(measureEvent.getInstantRoomMoistureContent() <
Building[roomID].getNeededMoistureContent()))||((TheSeason == seasonType.spring || TheSeason ==
seasonType.summer)&&(measureEvent.getInstantRoomMoistureContent() > Building[roomID].getNeededMoistureContent())))){
                                        queue.AddNewJob(new HumidifierOn(currentTime+1, roomID,
currentTime,measureEvent.getInstantRoomMoistureContent()));
                                }

                        //Room "i" 's HVAC is already working and this event checks to see whether the HVAC needs to continue
working or not
                        } else if(nextEvent instanceof HVAC_Running){
                                //start of the event's parsin
                                Activate_HVAC = (HVAC_Running) nextEvent;
                                roomID = Activate_HVAC.getRoomID();
                                currentTime = Activate_HVAC.getArrivalTime();
                                //HVAC working in cooling mode (summer or spring)
                                if(TheSeason == seasonType.spring || TheSeason == seasonType.summer){
```

```
                                                //if the room temperature is cool enough, close the HVAC and record the operational cost
for the period of operation
                                                if((Building[roomID].getRoomTemperature()-
Building[roomID].getNeededLowerTemperature() < 0.1)){
                                                        queue.AddNewJob(new HVAC_Deactivation(currentTime+1, roomID));
                                                        Building[roomID].TotalTimeElapsed += Math.abs(currentTime -
Activate_HVAC.getStartTime());

                                                        Building[roomID].TotalTimeElapsed /= 3600;              //to change it to
hours....

                                                        Building[roomID].costOfRunningTheRoom +=
(Building[roomID].TotalTimeElapsed*2.3*Building[roomID].EnergyUsed_HVAC/100); //assuming 2.3 cents/kWh

                                                        Building[roomID].EnergyUsed_HVAC = 0; //resetting the energy used after
the HVAC is shut off

                                                        Building[roomID].TotalTimeElapsed = 0;
                                                }else{
                                                        //if it is not, keep the HVAC running and decrease temperature and relative
humidity according to what was stated in Chapter 3 of my thesis
                                                        Building[roomID].setRoomTemperature((float) ((-1*0.002*(currentTime -
Activate_HVAC.getStartTime()))+Activate_HVAC.getStartingRoomTemperature()));

                Building[roomID].setRoomHumidity((float)(Activate_HVAC.getStartingRoomMoistureContent()*Math.pow((currentTime-
Activate_HVAC.getStartTime()), (-0.279/2))));
                                                        //the next scheduled check on whether to continue operating the HVAC is
after one second
                                                        queue.AddNewJob(new
HVAC_Running(currentTime+1,roomID,Activate_HVAC.getStartingRoomTemperature(),Activate_HVAC.getStartingRoomMoistureContent(),
Activate_HVAC.getStartTime()));
                                                }
                                        }
                                        else{       //for fall and winter, the same idea applies but vice-versa; when the temperature is high
enough, stop the HVAC
                                                if(Building[roomID].getRoomTemperature()-
Building[roomID].getNeededUpperTemperature() < -0.1){
                                                        Building[roomID].setRoomTemperature((float) ((0.002*(currentTime -
Activate_HVAC.getStartTime()))+Activate_HVAC.getStartingRoomTemperature()));

                Building[roomID].setRoomHumidity((float)(Activate_HVAC.getStartingRoomMoistureContent()*Math.pow((currentTime-
Activate_HVAC.getStartTime()), (-0.279/2))));
                                                        queue.AddNewJob(new
HVAC_Running(currentTime+1,roomID,Activate_HVAC.getStartingRoomTemperature(),Activate_HVAC.getStartingRoomMoistureContent(),
Activate_HVAC.getStartTime()));
                                                }else{
                                                        queue.AddNewJob(new HVAC_Deactivation(currentTime+1, roomID));
                                                        Building[roomID].TotalTimeElapsed += Math.abs(currentTime -
Activate_HVAC.getStartTime());
                                                        Building[roomID].TotalTimeElapsed /= 3600;              //to change it to
hours....
                                                        Building[roomID].costOfRunningTheRoom +=
(Building[roomID].TotalTimeElapsed*2.3*Building[roomID].EnergyUsed_HVAC/100);
                                                        Building[roomID].EnergyUsed_HVAC = 0; //resetting the energy used after
the HVAC is shut off
                                                        Building[roomID].TotalTimeElapsed = 0;
                                                }
                                                }
                                //HVAC section in Room "i" is stopped...
                                } else if(nextEvent instanceof HVAC_Deactivation){
                                        Deactivate_HVAC = (HVAC_Deactivation) nextEvent;
                                        roomID = Deactivate_HVAC.getRoomID();
                                        currentTime = Deactivate_HVAC.getArrivalTime();
                                        HVAC_Running = false;
                                        Building[roomID].HVAC_Deactivated();
                                        //after the HVAC is stopped, the temperature then takes its natural course in Room "i"
                                        queue.AddNewJob(new normal_Temperature_Increase_Interval(currentTime+1,roomID));
                                //Room "i" 's HVAC has stopped or never worked in the first place, and the temperature fluctuates according to
DesignBuilder's room temperature swing changes (see SharedConstantsInterface)
                                }else if(nextEvent instanceof normal_Temperature_Increase_Interval){
```

```
                                        //if the HVAC is not running and the room temperature is within bounds
                                        if((HVAC_Running == false)&&((((Building[roomID].getRoomTemperature()-
Building[roomID].getNeededLowerTemperature() > -0.1)&&(Building[roomID].getRoomTemperature()-
Building[roomID].getNeededUpperTemperature() < 0.1))))){
                                                Temp_Increase = (normal_Temperature_Increase_Interval) nextEvent;
                                                roomID = Temp_Increase.GetRoomID();
                                                currentTime = Temp_Increase.getArrivalTime();
                                                //in all seasons, the temperature increases or decreases to the nearest hour's
DesignBuilder Values in a linear fashion, in accordance to y = mx+c, where m is the slope
                                                //for summer and spring...
                                                if(TheSeason == seasonType.spring || TheSeason == seasonType.summer){

                                                        y_intercept = (float) (No_HVAC_Room_Temp_Summer[(int)
(((currentTime/3600 % 24)))]) - Building[roomID].getRoomTemperature();

                                                        y_intercept_Moisture_Content = (float)
(No_HVAC_Room_Moisture_Summer[(int) (((currentTime/3600 % 24)))]) - Building[roomID].getRoomHumidity();

                                                        currentNumberOfDays = (int) currentTime/86400;
                                                        x_intercept = ((float)
((currentNumberOfDays*86400)))+(float)(3600*((int)((currentTime-(currentNumberOfDays*86400))/3600)+1));
                                                        Building[roomID].setRoomTemperature((float)
((y_intercept/x_intercept)+Building[roomID].getRoomTemperature()));

                                                        Building[roomID].setRoomHumidity((float)
((y_intercept_Moisture_Content/x_intercept)+Building[roomID].getRoomHumidity()));

                                                        queue.AddNewJob(new
normal_Temperature_Increase_Interval(currentTime+1,roomID));
                                                }
                                                //or for fall and winter
                                                else{
                                                        y_intercept = (float)
(No_HVAC_Room_Temp_Winter[((int)((currentTime/3600%24)))]) - Building[roomID].getRoomTemperature();
                                                        y_intercept_Moisture_Content = (float)
(No_HVAC_Room_Moisture_Winter[(int) (((currentTime/3600 % 24)))]) - Building[roomID].getRoomHumidity();
                                                        currentNumberOfDays = (int) currentTime/86400;
                                                        x_intercept = ((float)
((currentNumberOfDays*86400)))+(float)(3600*((int)((currentTime-(currentNumberOfDays*86400))/3600)+1));
                                                        Building[roomID].setRoomTemperature((float)
((y_intercept/x_intercept)+Building[roomID].getRoomTemperature()));
                                                        Building[roomID].setRoomHumidity((float)
((y_intercept_Moisture_Content/x_intercept)+Building[roomID].getRoomHumidity()));
                                                        queue.AddNewJob(new
normal_Temperature_Increase_Interval(currentTime+1,roomID));
                                                }
                                        //if the temperature is out of bounds, the HVAC is activated, as in the event when the variables are
measured
                                        }else{
                                                Building[roomID].HVAC_Activated();
                                                HVAC_Running = true;
                                                if(VAV){
                                                        Building[roomID].CalculateSensibleHeatLoad();
                                                        Building[roomID].CalculateLatentHeatLoad();
                                                        Building[roomID].EnergyUsed_HVAC =
Math.abs(Building[roomID].getTotalLatentHeat()+Building[roomID].getTotalSensibleHeat());
                                                        Math.abs(Building[roomID].EnergyUsed_HVAC /= (13000));        //from
BTU/hr to needed wattage of the cooling/heating device assuming SEER of 13 and COP of 3.809
                                                        CFM_DA = 0;

                                                }else{
                                                        CFM_DA =
(((Building[roomID].getVentilation()*(Building[roomID].getRoomTemperature()+1))-
(CalculateTotalSensibleHeatLoad()/1.08))/(Building[roomID].getOutsideTemperature()-(2*Building[roomID].getRoomTemperature())));
                                                        T_mixed = (float)
((Building[roomID].getRoomTemperature()*(Building[roomID].getVentilation()/CFM_DA))+(Building[roomID].getOutsideTemperature()*(((C
FM_DA)-(Building[roomID].getVentilation()))/CFM_DA)));
                                                        T_adp = T_mixed-Building[roomID].getRoomTemperature();
                                                        Outside_Enthalpy = (float) (0.24*T_mixed); //specific heat capacity of air
```

```
                                                Indoor_Enthalpy = (float) (0.24*T_adp);                 //specific heat
capacity of air
                                                Building[roomID].EnergyUsed_HVAC = 4.45*(Outside_Enthalpy-
Indoor_Enthalpy)*CFM_DA;
                                                Math.abs(Building[roomID].EnergyUsed_HVAC/= 13000);
        //from BTU/hr to needed wattage of the cooling/heating device assuming SEER of 13 and COP of 3.809
                                                CFM_DA=0;
                                        }
                                        queue.AddNewJob(new
HVAC_Running(currentTime+1,roomID,Building[roomID].getRoomTemperature(), Building[roomID].getRoomHumidity(), currentTime));
                                }

                        //Room "i" 's outside conditions are adjusted every hour in accordance to DesignBuilder's exact values (see the
SharedConstants interface)
                        } else if (nextEvent instanceof adjust_Outside_Conditions){
                                Adjust_Conditions = (adjust_Outside_Conditions) nextEvent;
                                currentTime = Adjust_Conditions.getArrivalTime();
                                queue.AddNewJob(new adjust_Outside_Conditions((currentTime+3600)));

                                if(TheSeason == seasonType.spring || TheSeason == seasonType.summer){
                                        for(int i=0; i<= Building.length-1; i++){
                                                Building[i].setOutsideTemperature((float)
TypicalOutsideTemperaturesSummer[(int)((currentTime/3600%24))]);

        Building[i].setOutsideMoistureContent(ThePsychrometricChart.PerformRHtoGRFunction(Building[i].getOutsideTemperature(),
(float) ((float) 0.01*TypicalOutsideRelativeHumiditySummer[(int)((currentTime/3600%24))]),230));
                                                Building[i].setOutsideLightIntensity((float)
TypicalLightIntensitiesSummer[(int)(currentTime/3600%24)]);
                                        }
                                }else{
                                        for(int i=0; i<= Building.length-1; i++){
                                                Building[i].setOutsideTemperature((float)
TypicalOutsideTemperaturesWinter[(int)((currentTime/3600%24))]);

        Building[i].setOutsideMoistureContent(ThePsychrometricChart.PerformRHtoGRFunction(Building[i].getOutsideTemperature(),
(float) ((float) 0.01*TypicalOutsideRelativeHumidityWinter[(int)((currentTime/3600%24))]),230));
                                                Building[i].setOutsideLightIntensity((float)
TypicalLightIntensitiesWinter[(int)((currentTime/3600%24))]);
                                        }
                                }
                        //People are entering or leaving room "i" and going to another room within the home
                        }else if(nextEvent instanceof ExchangePeopleBetweenRooms){
                                //start of event's parsing
                                int OtherRoomID;
                                TheExchange = (ExchangePeopleBetweenRooms) nextEvent;
                                roomID = TheExchange.getRoomID();
                                OtherRoomID = roomID;
                                //if there is more than one room in the home
                                if(Building.length > 1){
                                        while((OtherRoomID == roomID)||(OtherRoomID > (Building.length-1))){
                                                OtherRoomID = (int)(Math.random()*10);    //another room is chosen at
random

                                        }
                                        currentTime = TheExchange.getArrivalTime();
                                        if(Building[roomID].ReturnTotalNumberOfPeople()>0){
                                                if(Building[OtherRoomID].ReturnTotalNumberOfPeople() < 100){
                                                        Building[roomID].decrementCurrentNumberOfPeople();
                                                        Building[OtherRoomID].incrementCurrentNumberOfPeople();
                                                }
                                        }
                                        //the next exchange happens (for room "i") after 30 minutes
                                        queue.AddNewJob(new
ExchangePeopleBetweenRooms((currentTime+(30*60)),roomID));
                                }
                        //the user chose to include a humidifier/dehumidifier option, and it has been turned on or is currently in
operation
                        } else if(nextEvent instanceof HumidifierOn){
                                //beginning to parse the event
                                HumidifierActive = (HumidifierOn) nextEvent;
```

```
                                        roomID = HumidifierActive.getRoomID();
                                        currentTime = HumidifierActive.getArrivalTime();
                                        Building[roomID].toggleHumidifier(true);
                                        //for spring and summer...
                                                if(TheSeason == seasonType.spring || TheSeason == seasonType.summer){
                                                        //...if the RELATIVE HUMIDITY, not the mositure content, is withing
bounds, stop the hum/dehum and calculate the operational cost of its usage

        if(ThePsychrometricChart.PerformGRtoRHFunction(Building[roomID].getRoomTemperature(),
Building[roomID].getRoomHumidity(), 230)- ThePsychrometricChart.PerformGRtoRHFunction(Building[roomID].getRoomTemperature(),
Building[roomID].getNeededLowerMoistureContent(), 230) < 0.1){
                                                                queue.AddNewJob(new HumidifierOff(currentTime+1, roomID));
                                                                Building[roomID].TotalTimeElapsedHumidifier =
Math.abs(currentTime - HumidifierActive.getStartingTime());

                                                                Building[roomID].TotalTimeElapsedHumidifier /= 3600;
        //to change it to hours....

                                                                Building[roomID].costOfRunningTheRoom +=
(Building[roomID].TotalTimeElapsedHumidifier*2.3*0.5/100);      //humidifier works at 3000 W

                                                                Building[roomID].TotalTimeElapsedHumidifier = 0;

                                                        }else{//if not, then keep the hum/dehumid activated according to Equation 3-3
in my Thesis

        Building[roomID].setRoomHumidity((float)(HumidifierActive.getStartingMoistureContent()*Math.pow((currentTime-
HumidifierActive.getStartingTime()), (-0.279/2))));
                                                                queue.AddNewJob(new
HumidifierOn(currentTime+1,roomID,HumidifierActive.getStartingTime(),HumidifierActive.getStartingMoistureContent()));
                                                        }
                                                        }
                                                //this also applies, but vice versa, to the fall and winter seasons
                                                else{

        if(ThePsychrometricChart.PerformGRtoRHFunction(Building[roomID].getRoomTemperature(),
Building[roomID].getRoomHumidity(), 230)- ThePsychrometricChart.PerformGRtoRHFunction(Building[roomID].getRoomTemperature(),
Building[roomID].getNeededUpperMoistureContent(), 230) < -0.1){

        Building[roomID].setRoomHumidity((float)(HumidifierActive.getStartingMoistureContent()*Math.pow((currentTime-
HumidifierActive.getStartingTime()), (0.279/2))));
                                                                queue.AddNewJob(new
HumidifierOn(currentTime+1,roomID,HumidifierActive.getStartingTime(),HumidifierActive.getStartingMoistureContent()));
                                                        }else{
                                                                queue.AddNewJob(new HumidifierOff(currentTime+1, roomID));
                                                                Building[roomID].TotalTimeElapsedHumidifier =
Math.abs(currentTime - HumidifierActive.getStartingTime());

                                                                Building[roomID].TotalTimeElapsedHumidifier /= 3600;
        //to change it to hours....

                                                                Building[roomID].costOfRunningTheRoom +=
(Building[roomID].TotalTimeElapsedHumidifier*2.3*0.5/100);      //humidifier works at 500 W

                                                                Building[roomID].TotalTimeElapsedHumidifier = 0;
                                                        }
                                                }
                                        //Room "i" 's hum/dehum is turned off
                                        } else if(nextEvent instanceof HumidifierOff){
                                                HumidifierInactive = (HumidifierOff) nextEvent;
                                                roomID = HumidifierInactive.getRoomID();
                                                currentTime = HumidifierInactive.getArrivalTime();
                                                //the turning off itself...
                                                Building[roomID].toggleHumidifier(false);
                                        }
                                }
        }//end of initiate simulation
        /*
         *
         */
```

```
	//This function displays the output CSV file should the user choose to press the button initiating so
	public void ShowOutputFile() throws IOException{
		Desktop.getDesktop().open(new File("C:\\Documents and Settings\\rzach007\\DesktopResultOutput.csv"));
	}
	//this function calculates the TOTAL instantaneous heat load across all rooms in a home; a simple summation
	private float CalculateTotalSensibleHeatLoad(){
		float tempTotalSensibleHeatLoad = 0;

		for (int i=0; i<= Building.length-1; i++){
			Building[i].CalculateSensibleHeatLoad();
			tempTotalSensibleHeatLoad += Building[i].getEffectiveSensibleHeat();
		}

		return tempTotalSensibleHeatLoad;
	}
	//this function calculates the total operational cost for the whole home; a simple summation of the instantaneous costs of all rooms
	private float CalculateTotalBuildingCost(){
		float tempTotalCost = 0;
		for (int i =0; i<= Building.length-1; i++){
			tempTotalCost+= Building[i].costOfRunningTheRoom;
		}
		return tempTotalCost;
	}

}//end of buildingConstruct.java

/*********************************************************************************************
File: Door.java
This class deifines the object "door", which is simply a door that opens and closes
 IN: Type of door, whether inner or outer
OUT: The instance of this class, with the user-inputted type of door
*********************************************************************************************/
package decision_Handling_Module;

import simulator_Backbone.*;

public class Door implements SharedConstants{
	doorType type;
	boolean openOrNot;

	//this function re-sets the type of door, whether inner or outer, if needed
	public Door(doorType tempDoorType){
		type = tempDoorType;
	}
	/*
	 * this function returns the type of door, whether inner or outer
	 */
	public doorType getDoorType(){
		return type;
	}
	/*
	 * this function checks to see whether the door is opened or not
	 */
	public boolean isDoorOpen(){
		return openOrNot;
	}
	/*
	 * this fucntion sets the door to open --> true, or closed --> false
	 */
	public void setDoorStatus(boolean statusDoor){
		openOrNot = statusDoor;
	}


}//end of Door.java

/*********************************************************************************************
Room.java
This class describes a "room" in the home; this rom contains door(s), walls, a light
sources, and people residing within its walls
```

```
 IN: Starting Room Conditions, starting simulator time, and outisde conditions
OUT: Object(s) which are implementations of this class, and which are used by the
simulator to simulate the CEMA algorithms. In particular, the sensible and latent heat
loads of this room can be instantaneously calculated from this class
*********************************************************************************/
package decision_Handling_Module;

import simulator_Backbone.*;

import misc_Operations.*;

public class Room implements SharedConstants{

	private int RoomID;
	//current room conditions
	private int CurrentNumberOfPeople;
	private float RoomTemperature;
	private float RoomMoistureContent;
	private float RoomLightIntensity;
	private Door door = null; //when and if needed, an external door is included here
	seasonType TheSeason = null;
	//outside room conditions
	private float OutsideTemperature;
	private float OutsideMoistureContent;
	private float OutsideLightIntensity;
	//instantaneous current simulator time
	public double TheCurrentTime;
	//needed room conditions
	private float Needed_Upper_Temperature;
	private float Needed_Upper_Moisture_Content;
	private float Needed_Lower_Temperature;
	private float Needed_Lower_Moisture_Content;
	private float Needed_Light_Intensity;
	//instantaneous required lighting device wattage for a constant luminosity level
	public float RequiredLightWattage = 0;
	//the minimum HVAC wattage required for "comfort" (temperature and RH)
	public double EnergyUsed_HVAC;
	//Variables used for calculating the running operational cost
	public double costOfRunningTheRoom = 0; //in Dollars
	public double TotalTimeElapsed = 0;		//in hours
	public double TotalTimeElapsedHumidifier = 0;		//in hours
	//booleans decribing the operation of the lighting and/or the shades in the room
	public boolean lightsOff = true;
	public boolean ShadesOn = false;
	//Variables and constants used in the calculations of the minimum HVAC wattage required (the constants are used as is from
DesignBuilder)
	float TempVentilation = -20;
	float TempInfiltration = -20;
	float SolarHeatGainFactor  = (float) 0.758;
	float StorageFactor = (float) 1.3;
	//the instance of the class which uses the external .dll file implementing all the psychrometric functions
	static psychrometricFunctions ThePsychrometricChart = new psychrometricFunctions();
	//booleans describing the current state of the HVAC, shades and humidifier
	private boolean HVAC_ON;
	private boolean Shades_On;
	private boolean Humidifier_On;
	//variables used to hold intermediate values in the calculations towards the minimum required HVAC wattage
	static public float TempLatentHeatLoad = 0;
	static public float TempSensibleHeatLoad = 0;
	static public float TotalLatentHeat = 0;
	static public float TotalSensibleHeat = 0;

	//the constructor, where all outside values are assigned to local variables
	public Room(int ID, float RoomTemp, float RoomHumid, float Light, float OutsideTemp, float MoistureContent, float
OutLightIntensity, seasonType Season, double currentTime)
	{
		RoomID = ID;
		RoomMoistureContent = RoomHumid;
		RoomTemperature = RoomTemp;
		RoomLightIntensity = Light;
```

```
		OutsideTemperature = OutsideTemp;
		OutsideMoistureContent = MoistureContent;
		OutsideLightIntensity = OutLightIntensity;
		CurrentNumberOfPeople = 1;
		Needed_Upper_Temperature = ZonesProperties.get(RoomID)[17];
		Needed_Upper_Moisture_Content = ZonesProperties.get(RoomID)[18]*100;
		Needed_Lower_Temperature = ZonesProperties.get(RoomID)[23];
		Needed_Lower_Moisture_Content = ZonesProperties.get(RoomID)[24]*100;
		Needed_Light_Intensity = ZonesProperties.get(RoomID)[19];
		TheCurrentTime = currentTime;
		TheSeason = Season;
		HVAC_ON = false;
		Shades_On = false;
		Humidifier_On = false;
}

//the following methods get or set all of the above-stated variables, and the methods' titles are self-explanatory
public boolean IsHumidifierOn(){
		return Humidifier_On;
}
public void toggleHumidifier(boolean Humidifier){
		Humidifier_On = Humidifier;
}
public float getEffectiveLatentHeat(){
		return TempLatentHeatLoad;
}
public float getEffectiveSensibleHeat(){
		return TempSensibleHeatLoad;
}
public float getTotalLatentHeat(){
		return TotalLatentHeat;
}
public float getTotalSensibleHeat(){
		return TotalSensibleHeat;
}
public float getVentilation(){
		return TempVentilation;
}
public int getRoomID(){
		return RoomID;
}
public void incrementCurrentNumberOfPeople(){
		if(CurrentNumberOfPeople < 100){
			CurrentNumberOfPeople++;
		}
}
public void decrementCurrentNumberOfPeople(){
		if(CurrentNumberOfPeople > 0){
		CurrentNumberOfPeople--;
		}
}
public int ReturnTotalNumberOfPeople(){
		return CurrentNumberOfPeople;
}

public Door getDoor(){
		return this.door;
}
public void SetDoor(){
		door = new Door(doorType.Outer);
}

public float getRoomTemperature(){
		return RoomTemperature;
}
/*
 *
 */
public void setRoomTemperature(float NewTemperature){
		RoomTemperature = NewTemperature;
```

```
}
public float getRoomHumidity(){
		return RoomMoistureContent;
}
public void setRoomHumidity(float NewMoistureContent){
		RoomMoistureContent = NewMoistureContent;
}
public float getRoomLightIntentsity(){
		return RoomLightIntensity;
}
public void setRoomLightIntensity(float newLightIntensity){
		RoomLightIntensity = newLightIntensity;
}
public void setOutsideTemperature(float newOutsideTemperature){
		OutsideTemperature = newOutsideTemperature;
}
public float getOutsideTemperature(){
		return OutsideTemperature;
}
public void setOutsideMoistureContent(float newMoistureContent){
		OutsideMoistureContent = newMoistureContent;
}
public float getOutsideMoistureContent(){
		return OutsideMoistureContent;
}
public void setOutsideLightIntensity(float newLightIntensity){
		OutsideLightIntensity = newLightIntensity;
}
public float getOutsideLightIntensity(){
		return OutsideLightIntensity;
}
public float getNeededUpperTemperature(){
		return Needed_Upper_Temperature;
}
public float getNeededUpperMoistureContent(){
		return Needed_Upper_Moisture_Content;
}
public float getNeededLowerTemperature(){
		return Needed_Lower_Temperature;
}
public float getNeededLowerMoistureContent(){
		return Needed_Lower_Moisture_Content;
}
public float getNeededLightIntensity(){
		return Needed_Light_Intensity;
}

public void HVAC_Activated(){
		HVAC_ON = true;
}
public void HVAC_Deactivated(){
		HVAC_ON = false;
}
public boolean isHVACOn(){
		return HVAC_ON;
}

public boolean areShadesOn(){
		return Shades_On;
}

public void ActivateShades(){
		Shades_On = true;
}
public void DeactivateShades(){
		Shades_On = false;
}
/*
 * This function calculates the sensible heat load of the room at any instant, exactly according to the calculations set in Appendix C,
and in the order of the equations set there
```

```
	 */
	public void CalculateSensibleHeatLoad()
	{
		TempVentilation = 0;
		TempInfiltration = 0;
		TempSensibleHeatLoad = 0;
		TempLatentHeatLoad = 0;
		TotalSensibleHeat = 0;
		TotalLatentHeat = 0;
		TempVentilation = (float)
((CurrentNumberOfPeople*20)+(0.33*ZonesProperties.get(RoomID)[0]*ZonesProperties.get(RoomID)[1]));
		if((door != null)&&(Door.open)){
			TempInfiltration = (float)(6.5*ZonesProperties.get(RoomID)[21]*ZonesProperties.get(RoomID)[22]);
		}else{
			TempInfiltration = 0;
		}
		//Solar Gain through glass
		TempSensibleHeatLoad+=
ZonesProperties.get(RoomID)[13]*ZonesProperties.get(RoomID)[15]*ZonesProperties.get(RoomID)[16]*SolarHeatGainFactor*0.85;

		//Transmission and Solar Gain through Walls and Roof + effect of outdoor air

		if((TheSeason == seasonType.summer)||(TheSeason == seasonType.spring)){
			TempSensibleHeatLoad+=
ZonesProperties.get(RoomID)[4]*ZonesProperties.get(RoomID)[5]*EquivalentTempDiffNotRoof[0][(int)(TheCurrentTime/3600%24)]*0.062;
//walls
			TempSensibleHeatLoad+=
ZonesProperties.get(RoomID)[6]*ZonesProperties.get(RoomID)[7]*EquivalentTempDiffNotRoof[1][(int)(TheCurrentTime/3600%24)]*0.062;
//walls
			TempSensibleHeatLoad+=
ZonesProperties.get(RoomID)[8]*ZonesProperties.get(RoomID)[9]*EquivalentTempDiffNotRoof[2][(int)(TheCurrentTime/3600%24)]*0.062;
//walls
			TempSensibleHeatLoad+=
ZonesProperties.get(RoomID)[10]*ZonesProperties.get(RoomID)[11]*EquivalentTempDiffNotRoof[3][(int)(TheCurrentTime/3600%24)]*0.062
; //walls


			TempSensibleHeatLoad+=
ZonesProperties.get(RoomID)[0]*ZonesProperties.get(RoomID)[1]*EquivalentTempDiffRoof[(int)(TheCurrentTime/3600%24)]*0.044; //roof

		}else{
			TempSensibleHeatLoad+=
((ZonesProperties.get(RoomID)[4]*ZonesProperties.get(RoomID)[5])+(ZonesProperties.get(RoomID)[6]*ZonesProperties.get(RoomID)[7])+(Zo
nesProperties.get(RoomID)[8]*ZonesProperties.get(RoomID)[9])+(ZonesProperties.get(RoomID)[10]*ZonesProperties.get(RoomID)[11]))*(Out
sideTemperature-RoomTemperature)*0.062;//walls
			TempSensibleHeatLoad+=
ZonesProperties.get(RoomID)[0]*ZonesProperties.get(RoomID)[1]*(OutsideTemperature-RoomTemperature)*0.044;//roof

		}
		TempSensibleHeatLoad +=
ZonesProperties.get(RoomID)[12]*ZonesProperties.get(RoomID)[15]*ZonesProperties.get(RoomID)[16]*(OutsideTemperature-
RoomTemperature)*(0.568); //through glass
		TempSensibleHeatLoad +=
2*ZonesProperties.get(RoomID)[0]*ZonesProperties.get(RoomID)[1]*0.656*(OutsideTemperature-RoomTemperature-5);
		if(TempInfiltration>TempVentilation){
			TempSensibleHeatLoad += (TempInfiltration-TempVentilation)*(OutsideTemperature-
RoomTemperature)*1.08;
		}
		TempSensibleHeatLoad += (CurrentNumberOfPeople*614.12);
		TempSensibleHeatLoad += 50*1.25*3.4;//assume sole light in area is 50W bulb
		TempSensibleHeatLoad+= -1*StorageFactor*ZonesProperties.get(RoomID)[0]*ZonesProperties.get(RoomID)[1];
		TotalSensibleHeat = (float) (TempSensibleHeatLoad+ TempVentilation*(OutsideTemperature-RoomTemperature)*1.08);


	}
	/*
	 * This function calculates the latent heat load of the room at any instant, exactly according to the calculations set in Appendix C, and
in the order of the equations set there
	 */
	public void CalculateLatentHeatLoad()
```

```
		{
			if(TempInfiltration > TempVentilation){
				TempLatentHeatLoad += (TempInfiltration - TempVentilation)*(OutsideMoistureContent-
RoomMoistureContent)*0.68;
			}
			TempLatentHeatLoad += (CurrentNumberOfPeople*20);
			TempLatentHeatLoad +=
((ZonesProperties.get(RoomID)[4]*ZonesProperties.get(RoomID)[5])+(ZonesProperties.get(RoomID)[6]*ZonesProperties.get(RoomID)[7])+(Zo
nesProperties.get(RoomID)[8]*ZonesProperties.get(RoomID)[9])+(ZonesProperties.get(RoomID)[10]*ZonesProperties.get(RoomID)[11]))*0.01
*(OutsideMoistureContent-RoomMoistureContent)*(0.79+0.23);
			TempLatentHeatLoad +=
2*ZonesProperties.get(RoomID)[0]*ZonesProperties.get(RoomID)[1]*10.764*0.01*(OutsideMoistureContent-RoomMoistureContent)*(0.5);
			TotalLatentHeat = (float) (TempLatentHeatLoad+(TempVentilation*(OutsideMoistureContent-
RoomMoistureContent)*0.68));
		}
}
"input_Output_Manipulation"
/*******************************************************************************************
File: CSV_Parser.java
This file handles parsing the input files containing the home's rooms' properties
and handles the output file where the instantaneous condition of all the rooms are
into
IN: The Input File "RoomProperties.csv"
OUT: The Output file "DesktopResultOutput.csv"
*******************************************************************************************/
package input_Output_Manipulation;
import java.util.Map;
import java.util.Map.Entry;
import java.util.*;
import java.io.*;

import net.sf.csv4j.CSVFileProcessor;
import net.sf.csv4j.CSVFieldMapProcessor;
import net.sf.csv4j.CSVWriter;
import simulator_Backbone.*;

public class CSV_Parser implements SharedConstants{

	//both files needed to write up the output file
	FileWriter fileWriter = null;
	CSVWriter csvWriter = null;
	public CSV_Parser(){
		try {
			//the output file is initiated
			fileWriter = new FileWriter( "C:\\Documents and Settings\\rzach007\\DesktopResultOutput.csv" );
			csvWriter = new CSVWriter( fileWriter );
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public void ReturnParsedValues()throws Exception{

		final CSVFileProcessor fp = new CSVFileProcessor();
		//here, the input file is processed
		fp.processFile( "C:\\Documents and
Settings\\rzach007\\workspace\\Rami_Henry_5012602_Thesis_Simulator\\src\\RoomProperties.csv", new CSVFieldMapProcessor() {
			@SuppressWarnings("unchecked")
			public void processDataLine(final int linenumber, final Map<String,String> fields) {
			//for each element in a data line, first it must be checked that the new data element being parsed is from the
same line (same room)
				//if not, then a new row is added to the array "ZoneProperties" (1). In either case, the header of each
piece of data is extracted and the
				//and the piece of data is then placed in a particular position in the array according to that header's
value, as is seen below.
				float[] TempRoomProperties = null;
			int OldLineNumber = 0;
			for ( final Entry field : fields.entrySet() ) {

				if(OldLineNumber!=linenumber){
					OldLineNumber = linenumber;
```

```
                ZonesProperties.add(new float[25]);                    //(1)
}if (field.getKey().equals("Length")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[0] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("Width")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[1] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("Height")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[2] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("Number of Indoor Walls")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[3] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("North Facing")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[4] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("North Area")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[5] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("East Facing")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[6] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("East area")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[7] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("South Facing")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[8] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("South Area")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[9] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("West Facing")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[10] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("West Area")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[11] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("Number of Outdoor Windows")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[12] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
                TempRoomProperties = null;
}else if (field.getKey().equals("Number of Sunlit Outdoor Windows")){
                TempRoomProperties = ZonesProperties.get(linenumber-2);
                TempRoomProperties[13] = Float.parseFloat(field.getValue().toString());
                ZonesProperties.set(linenumber-2, TempRoomProperties);
```

```
                        TempRoomProperties = null;
                }else if (field.getKey().equals("Number of Indoor Windows")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[14] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("Length Of Window")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[15] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("Width Of Window")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[16] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("NeededUpperTemperature")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[17] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("NeededUpperRelativeHumidity")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[18] = Float.parseFloat(field.getValue().toString())/100;
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("NeededLightIntensity")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[19] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("Number of Outdoor Doors")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[20] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("Length of Door")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[21] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("Width of Door")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[22] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("NeededLowerTemperature")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[23] = Float.parseFloat(field.getValue().toString());
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }else if (field.getKey().equals("NeededLowerRelativeHumidity")){
                        TempRoomProperties = ZonesProperties.get(linenumber-2);
                        TempRoomProperties[24] = Float.parseFloat(field.getValue().toString())/100;
                        ZonesProperties.set(linenumber-2, TempRoomProperties);
                        TempRoomProperties = null;
                }
        }
        }

        //the next two functions are related to the CSV file parsing technique used in this thesis
        @SuppressWarnings("unused")
        public void processHeaderLine(int arg0, List<String> arg1) {
                // TODO Auto-generated method stub

        }

public boolean continueProcessing()
{
        return true;
```

```
                        }
                    } );
            }
            /*
             * This function is used to write the header of the output file
             */
            public void InitiateOutputFile(String TheComment){
                    try {
                            csvWriter.writeCommentLine(TheComment);
                            csvWriter.writeLine( new String[] { "Room ID","Time(sec)", "Temp(C)", "Humidity(%)", "Light
Intensity","Cost of Running Room ($)","Cost of Running the whole building ($)", "Outside Tempersture(Celsius)", "IsHvacOn"} );
                    } catch (Exception e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    }
            }
            /*
             * This function is used each time a room's condition is recorded. This function actually writes the data to the file....line by line, for
every capture
             */
            public void WriteDataLine(double[] TheData){
                    try {
                            csvWriter.writeLine(new String[] {Double.toString(TheData[0]),
Double.toString(TheData[1]),Double.toString(TheData[2]),Double.toString(TheData[3]),Double.toString(TheData[4]),Double.toString(TheData[
5]),Double.toString(TheData[6]),Double.toString(TheData[7]),Double.toString(TheData[8])});
                    } catch (Exception e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    }
            }
            //this function closes the output file
            public void CloseOutputFile(){
                    try {
                            fileWriter.close();
                    } catch (Exception e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    }
            }

}
```

“Queue_Element_Types”

```
/*********************************************************************************************
[1] File: adjust_Outside_Conditions.java and others
These 10 files describe the 10 types of events that encompass all CEMA algorithms
currently coded in this simulator. These events are, in order of appearence:
[1]outside conditions that are adjusted every set time (one hour here)[2] door closing        *
[3] Door opening [4] People moving between rooms [5] The hum/dehum turned off
[6] The hum/dehum turned on [7] The HVAC turned off [8] The HVAC turned on
[9] The room's variables are captured, recorded and acted upon [10] The room's
temperature, without any man-made interference, changes
NOTE: All these classes inherit from the class DynamicEvent which generically describes
what an event should contain (here, it is the timestamp of the event)
*********************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class adjust_Outside_Conditions extends DynamicEvent implements SharedConstants {
            public adjust_Outside_Conditions(double time){
                    super(time);
```

```
		}
}
/*******************************************************************************************
[2] File: DoorClose.java
*******************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class DoorClose extends DynamicEvent implements SharedConstants{

		private int roomID;
		public DoorClose(double closingTime, int idRoom){
			super(closingTime);
			roomID = idRoom;	//the room ID where this door which is closing resides
		}
		/*
		 *
		 */
		public int getRoomID(){
			return roomID;
		}
}
/*******************************************************************************************
[3] File: DoorOpen.java
*******************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class DoorOpen extends DynamicEvent implements SharedConstants {

		private doorAction doorsAction;
		private int roomID;
		public DoorOpen(double time,doorAction state,int idRoom){
			super(time);
			doorsAction = state;
			roomID = idRoom;
		}

		public doorAction getDoorAction(){
			return doorsAction;
		}

		public int GetRoomID(){
			return roomID;
		}

}
/*******************************************************************************************
[4] File: ExchangePeopleBetweenRooms.java
*******************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class ExchangePeopleBetweenRooms extends DynamicEvent implements SharedConstants{

		private int roomID;
		public ExchangePeopleBetweenRooms(double Time, int idRoom){
			super(Time);
			roomID = idRoom;
		}
		/*
```

```
		 *
		 */
		public int getRoomID(){
			return roomID;
		}
}
/*******************************************************************************************
[5] File: HumidifierOff.java
*******************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class HumidifierOff extends DynamicEvent implements SharedConstants {

		private int roomID;
		public HumidifierOff(double closingTime, int idRoom){
			super(closingTime);
			roomID = idRoom;
		}

		public int getRoomID(){
			return roomID;
		}
}
/*******************************************************************************************
[6] File: HumidifierOn.java
*******************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class HumidifierOn extends DynamicEvent implements SharedConstants {

		private int roomID;
		private double StartingTime;
		private float StartMoisture;

		public HumidifierOn(double closingTime, int idRoom, double StartTime,float StartingMoistureContent){
			super(closingTime);
			roomID = idRoom;
			StartingTime = StartTime;
			StartMoisture = StartingMoistureContent;
		}

		public int getRoomID(){
			return roomID;
		}

		public double getStartingTime(){
			return StartingTime;
		}

		public float getStartingMoistureContent(){
			return StartMoisture;
		}
}
/*******************************************************************************************
[7] File: HVAC_Deactivation.java
*******************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;
```

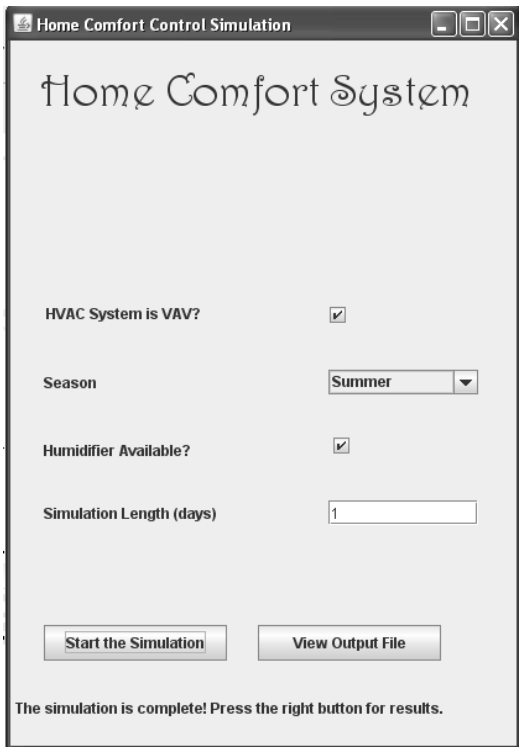
```
public class HVAC_Deactivation extends DynamicEvent implements SharedConstants {

	private int ID;

	public HVAC_Deactivation(double time, int room_ID){
		super(time);
		ID = room_ID;

	}

	public int getRoomID(){
		return ID;
	}

}
/*********************************************************************************
[8] File: HVAC_Running.java
*********************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class HVAC_Running extends DynamicEvent implements SharedConstants {

	private int ID;
	private double RoomTemperature_Initial;
	private double RoomMoistureContent;
	private double StartTime;

	public HVAC_Running(double time,int idRoom, double StartingRoomTemperature, double StartingRoomMoistureContent, double
StartingTime){
		super(time);
		ID = idRoom;
		//used for many types of calculations
		RoomTemperature_Initial = StartingRoomTemperature;
		RoomMoistureContent = StartingRoomMoistureContent;
		StartTime = StartingTime;
	}

	public int getRoomID(){
		return ID;
	}
	/*
	 * this function sets a new starting time for the HVAC
	 */
	public void setStartTime(int NewStartTime){
		StartTime = NewStartTime;
	}
	/*
	 * this function gets the starting time for the HVAC
	 */
	public double getStartTime(){
		return StartTime;
	}
	/*
	 * this function gets the initial temperature at which the HVAC was activated
	 */
	public double getStartingRoomTemperature(){
		return RoomTemperature_Initial;
	}
	/*
	 * this function sets the initial temperature at which the HVAC was activated
	 */
	public void setStartingRoomTemperature(double Temperature){
		RoomTemperature_Initial = Temperature;
	}
	/*
	 * this function gets the initial moisture content, in grains per pound at which the hum/dehum was activated
```

```
		 */
		public double getStartingRoomMoistureContent(){
			return RoomMoistureContent;
		}
}
/*****************************************************************************************
[9] File: MeasureRoomVariables.java
NOTE: Other than the usual function that is common in all event classes, this class
contains functions that gets or sets the below-stated room variables
*****************************************************************************************/
package Queue_Element_Types;
import simulator_Backbone.DynamicEvent;
import simulator_Backbone.SharedConstants;
public class MeasureRoomVariables extends DynamicEvent implements SharedConstants{

		//room variables to be captured
		float RoomTemp;
		float RoomMoistureContent;
		float RoomLightIntensity;
		int ID;
		public MeasureRoomVariables(double MeasuringTime, int roomID){
			super(MeasuringTime);
			ID = roomID;
		}

		public float getInstantRoomTemperature(){
			return RoomTemp;
		}
		public void changeRoomTemperature(float NewRoomTemperature){
			RoomTemp = NewRoomTemperature;
		}
		public void changeMoistureRoomContent(float NewMoistureContent){
			RoomMoistureContent = NewMoistureContent;
		}
		public void changeLightIntensity(float NewLightIntensity){
			RoomLightIntensity = NewLightIntensity;
		}
		public float getInstantRoomMoistureContent(){
			return RoomMoistureContent;
		}
		public float getInstantRoomLightIntensity(){
			return RoomLightIntensity;
		}
		public int getRoomID(){
			return ID;
		}
}
/*****************************************************************************************
[10] File: normal_Temperature_Increase_Interval.java
*****************************************************************************************/
package Queue_Element_Types;

import simulator_Backbone.DynamicEvent;

import simulator_Backbone.SharedConstants;

public class normal_Temperature_Increase_Interval extends DynamicEvent implements SharedConstants {
		private int ID;
		public normal_Temperature_Increase_Interval(double time, int roomID)
		{
			super(time);
			ID = roomID;
		}
		public int GetRoomID(){
			return ID;
		}
}
/*****************************************************************************************
FILE: Dynamic Event.java (property of PostDoc AbdelHamid El Shoul)
This class describes the generic event on which all CEMA simulator events are based on   *
```

```
IN: Event time
OUT: The generic event
*********************************************************************************/
package simulator_Backbone;

public class DynamicEvent {

	private double arrivalTime;

	public DynamicEvent(double time){
		arrivalTime = time;
	}

	public void setArrivalTime(double temptime){
		arrivalTime = temptime;
	}

	public double getArrivalTime(){
		return arrivalTime;
	}

}
/*********************************************************************************
FILE: SharedConstants.java
IN: NONE
OUT: The relevant classes which implement this interface
*********************************************************************************/
package simulator_Backbone;


import java.util.ArrayList;

public interface SharedConstants {
	public enum doorAction{personIn,personOut};
	public enum doorType{Inner, Outer};
	public final boolean closed  = false;
	public final boolean open  = true;
	public enum seasonType {summer,fall,winter,spring};
	public seasonType SimulationSeason = null;
	//these set of values correspond to the equivalent temperature difference every hour, in F, between the outer and inner surface of any
wall, in summer and spring only. These values are direct from [Car65], for the roof and for other surfaces, respectively
	public float [] EquivalentTempDiffRoof = {17,13,11,9,6,4,3,2,3,6,10,16,23,28,33,38,40,41,39,35,32,28,24,20};
	//this is a 2-D array corresponding to the different values, in 24 hours, for walls facing all four compass directions
	public float [] [] EquivalentTempDiffNotRoof = {{2,1,0,-1,-2,-3,-3,-4,-3,-2,-1,0,3,6,8,10,11,12,12,12,10,8,6,4},{4,3,1,1,0,-1,-
1,0,21,30,31,31,19,14,13,12,13,14,13,12,11,10,8,5},{2,1,1,0,-1,-1,-3,-4,-3,-
2,7,12,20,24,25,26,23,20,15,12,10,8,6,4},{5,4,3,3,2,2,1,0,0,0,2,4,7,10,19,26,34,40,41,36,28,16,10,6}};
	//The next set of arrays have names that are self-explanatory, suffice it to say that these values are direct from DesignBuilder; the
arrays' names describe the conditions on which these hourly figures were extracted
	public double [] TypicalOutsideTemperaturesSummer =
{69.13142857,68.48857143,67.46,66.48285714,66.50857143,67.56285714,69.62,71.72857143,73.94,76.25428571,76.97428571,79.54571429,82
.04,83.81428571,80.29142857,79.49428571,79.88,78.8,77.41142857,75.45714286,73.78571429,72.16571429,70.88,70.16};
	public double [] TypicalOutsideTemperaturesWinter = {-0.837142857,-1.3,-0.914285714,-1.325714286,-2.277142857,-2.74,-
2.251428571,-2.868571429,-
1.634285714,0.037142857,0.86,2.531428571,3.2,4.408571429,5.617142857,6.388571429,6.568571429,6.62,6.311428571,5.514285714,4.92285
7143,4.022857143,3.045714286,2.48};
	public double [] TypicalOutsideRelativeHumiditySummer  =
{85.83246256,85.78639416,88.43103999,90.26625791,91.21607487,90.30805586,85.37308856,80.36379834,76.3853175,72.48070352,69.3061
5606,66.23261462,60.19962176,55.83213774,61.73958621,64.08588286,64.20334634,67.29575288,70.47645127,73.7124735,75.41847672,78.3
0570848,81.6839628,83.23301566};
	public double [] TypicalOutsideRelativeHumidityWinter =
{64.6777482,65.902192,65.70864443,64.73698723,64.55612354,66.60960892,66.70596505,68.24011876,68.05978146,67.13938904,66.050345
7,64.8841179,64.77484236,64.56306584,64.02323068,63.66451707,63.61984287,63.01498718,63.88737117,64.1103096,63.97207406,64.68623
783,66.56755595,65.37778649};
	//...only the next two arrays were not from DesignBuilder; these were the author's educated guesses
	public double [] TypicalLightIntensitiesSummer = {0,5,0,0,0,5,10,20,30,40,60,80,80,80,60,60,50,40,30,20,5,0,0,0};
	public double [] TypicalLightIntensitiesWinter = {0,5,0,0,0,0,0,5,10,15,30,40,60,80,80,50,30,20,10,5,0,0,0,0};

	public double [] No_HVAC_Room_Temp_Summer = {77.21428571,76.86428571,76.43571429,76.00285714,
75.60428571,75.43285714,76.86571429,77,77.7,78.69571429,79.85571429,81.01714286,82.06714286,82.83571429,82.60857143,82.02142857,
77.13285714,77,77,77,77,77,77,77.76285714};
```

```
	public double [] No_HVAC_Room_Temp_Winter =
{18.16571429,17.02285714,16.41857143,15.95714286,15.44714286,14.98571429,14.66142857,14.52428571,15.22285714,16.05714286,17.28,17.91,18.33857143,18.74428571,19.08285714,19.29571429,19.31571429,19.20142857,21.07285714,21.96428571,22.32,22.54428571,22.56857143,20.60714286};
	public double [] No_HVAC_Room_Moisture_Summer =
{83.72904364,88.60007414,90.207659,88.23159821,85.15491857,83.77163357,80.82223814,79.98962171,83.94215571,88.37240957,92.06173079,94.20207093,95.45421757,95.86372071,95.25902171,86.49617993,74.80739921,69.79776193,69.25243979,70.6274475,71.84138607,72.77201436,76.35257229,68.12183993};
	public double [] No_HVAC_Room_Moisture_Winter =
{4.5112387,4.05889075,3.751055079,3.610698164,3.490264307,3.384127136,3.346013264,3.358163864,3.400688979,3.49088175,3.643757257,3.845063907,4.050840914,4.236375329,4.422151764,4.608162671,4.7676991,4.856703757,5.061191243,5.18783905,5.216733993,5.194557764,5.065045857,4.886098693};
	//the arraylist which holds all the rooms the user inputs from the "RoomProperties.csv"; initially it holds nothing; this arraylist represents the "home" as a whole
	static public ArrayList<float[]> ZonesProperties = new ArrayList<float[]>(0);
}
/*********************************************************************************
File: Main_Program.java
Most of this file is automatically generated as the GUI interface is being built. What will be shown in
Appendix A is only the code executed when the user presses the button initiating the simulation
IN: User inputs of simulation length in days, season, and whether there is a humidifier
/dehumidifier avaialbale, in additon to the type of HVAC being simulated
OUT: The output file continaing all the rooms' instantaneous conditions after the CEMA  *
algorithms have run for the stated period
*********************************************************************************/
	private JButton getSimualtionStart(){
		if (SimualtionStart == null) {
			SimualtionStart = new JButton();
			SimualtionStart.setBounds(new Rectangle(27, 474, 148, 29));
			SimualtionStart.setText("Start the Simulation");
			SimualtionStart.addActionListener(new java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					//The data inputted by the user is validated
					for(int i=0; i<=SimulationLengthInput.getText().length()-1; i++){
						if(!(Character.isDigit(SimulationLengthLabel.getText().charAt(i)) || (SimulationLengthLabel.getText().charAt(i) != '.'))){
							ValidOrNot = false;
							break;
						}
					}
					//if the inputted simulation length is not valid
					if(ValidOrNot == false){
						SimulationLengthInput.setText("");
						JOptionPane.showMessageDialog(null, "One of the inputs is not in the correct format. Please re-enter inputs");
					}
					else{//the input is valid....
						TypeOfSeason = (String) SeasonList.getSelectedItem();
						TimeOfSimulation = Double.parseDouble(SimulationLengthInput.getText())*24*60*60;

						try {
							//...and thus the building is initiated and the inputs read and parsed
							TheBuilding = new buildingConstruct(TimeOfSimulation, HumidifierAvailable,TypeOfSeason,VAVSelection.isSelected(),HumidifierSelection.isSelected());
						} catch (NumberFormatException e1) {
							e1.printStackTrace();
						} catch (Exception e1) {
							e1.printStackTrace();
						}
						try {
							//the simulation is run
							TheBuilding.InitiateSimulation();
						} catch (IllegalAccessException e1) {
							e1.printStackTrace();
						}
						//the user will see a confirmation of simulation completion and will be able to open the output file directly
						OutputFileView.setEnabled(true);
						ResultAnnouncement.setVisible(true);
```

```
                                                    }
                                                }
                                            });
                                }
                                return SimualtionStart;
}
```



Screenshot of simulator's GUI after completion of the simulation

Appendix B: Confidence Interval Calculations

The correctness of a simulator and its results is defined by the confidence interval [**Guo06**], [**Wei98**]. To explain this concept, the reader can assume that there exists a simulator with multiple sources of randomness. When the simulation is run with the same set of assumptions and inputs multiple times, the result of the simulator is expected to be different each time, given the assumption that each simulator run is statistically independent of the other. How different the results are is the central issue in understanding the concept of the confidence interval. If the results are close together, then the simulator is correct and acceptable. If the results are not close, then more care has to be taken in the design of the simulator [**Wei98**]. The definition of close is expressed as a percentage; for example, within what range of values of the result do 95% of the outputs of the runs lie? This range of values can be described as being between upper and lower bounds, and as being governed by the t-distribution [**Guo06**], as seen in the graph below:

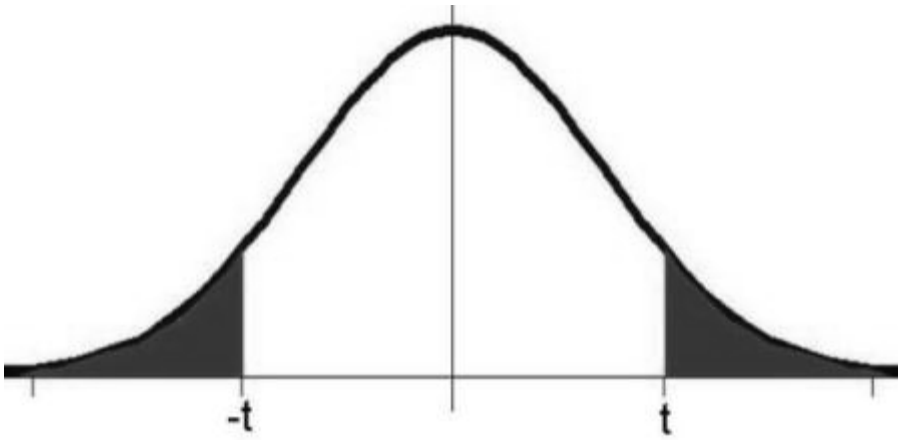


Figure B.1: The t-distribution

The percentage value expressed above is the white portion of the above normal distribution, and the rest is where the range of simulator output values is excluded. The value of t is half the excluded value. For example, if one were to consider 95% of the values of simulator output results, then the value of t would be 0.5*(1-0.95) = 0.025. The t-distribution is also governed by

the number of independent runs the simulation goes through, using the same set of inputs and assumptions for all of them.

Thus, to get the confidence interval, the mean of all the results of the simulator must be computed, based on the normal distribution function, where N is the number of runs of the simulator using a certain set of assumptions and inputs [**Guo06**], [**Wei98**]:

$$Mean = \frac{\sum_{x=1}^{N} V_x}{N}$$

The standard deviation is also computed based on the normal distribution function [**Guo06**], [**Wei98**]:

$$St.Dev. = \sqrt{\frac{\sum_{x=1}^{N}(V_x - Mean)^2}{N-1}}$$

Thus, the upper and lower bounds of the confidence interval is defined as the range of values higher than and lower than the mean where P% of the simulator outputs lie. It is computed as follows [**Guo06**] [**Wei98**]:

$$Confidence\ Interval = Mean \pm \frac{St.Dev.* T_{\left(\frac{100-P}{2*100}, N-1\right)}}{\sqrt{N}}$$

The term $T_{\left(\frac{100-P}{2*100}, N-1\right)}$ denotes a t-distribution factor based on that the value of t in the above graph is $\frac{100-P}{2*100}$, and that the confidence interval has N-1 degrees of freedom. This value of this factor is found in any available t-distribution table.

As can be seen from the above calculations, the correctness and validity of the simulator is more accurately proven when there are more independent simulator runs being executed. In this simulator, there have been four independent runs, which means three degrees of freedom. The percentage where the simulator run values were required to be within has been ninety-five percent.

Appendix C: Heating/Cooling Load Calculations

Preliminary Calculations

Contribution of Ventilation, in cfm

$$\sum_{p=1}^{P}[\text{Ventilation}_{\text{person}}] + (\text{A}_{\text{floor}} * 10.764 * \text{V}_{\text{m}^2})^{1} \quad (\mathbf{C.1})$$

Contribution of Infiltration, in cfm

$$\sum_{\acute{d}}^{\acute{D}}[\text{Infiltration}_{\text{open door}}]^{2} \quad (\mathbf{C.2})$$

Sensible Heat Calculations

Solar Gain through Glass, in BTU hr^{-1}

$$\sum_{g=\acute{1}}^{\acute{G}}[\text{A}_{\acute{g}} * \text{F}_{\text{SHG}} * 10.764 * \text{F}_{\text{SF}}]^{3} \quad (\mathbf{C.3})$$

Solar and Transmission Gain – Walls and Roof, in BTU hr^{-1}

$$\sum_{w=1}^{W}[\text{A}_{\text{w}} * 10.764 * \Delta\text{T}_{\text{wall,season}} * \text{U}_{\text{wall}}] + (\text{A}_{\text{roof}} * 10.764 * \Delta\text{T}_{\text{roof,season}} * \text{U}_{\text{roof}})^{4} \quad (\mathbf{C.4})$$

[1] A_{floor} refers to the floor area of the conditioned space, in m^2. $Ventilation_{person}$ refers to the ventilation standard per person in a residential space, in cfm. V_{m^2} refers to the ventilation standard per square foot of conditioned residential space, in cfm.

[2] $\acute{d}$ and $\acute{D}$ refer to the number of outdoor doors that are open.

[3] $\acute{G}$ and $\acute{g}$ refer to the number of sunlit windows. $A_{\acute{g}}$ refers to the area of each individual sun-lit window, in m^2. F_{SHG} refers to the peak solar heat gain for this particular window, in BTU hr^{-1} ft^{-2}. F_{SF} refers to the storage factor for this particular scenario, and is unitless.

Transmission Gain other than (5) , in BTU hr^{-1}

$$\sum_{g=1}^{G}[A_g * 10.764 * \Delta T_f * U_{Glass}] + \left(2 * A_{CorF} * 10.764 * U_{CorF,Up\ or\ Down} * (\Delta T_f - 5)\right) + \left(((\mathbf{2}) - (\mathbf{1})) * \Delta T_f * 1.08\right)^{5} \quad (\mathbf{C.5})$$

Internal Heat, in BTU hr^{-1}

$$\sum_{p=1}^{P}[\text{Heat Gain}_{sensible}] + (\text{Watt}_{Lights,Total} * 1.25 * 3.4) + \left(\sum_{app=1}^{APP}\left[W_{app} * (1 - eff_{app}) * 60 * \frac{1}{3413}\right]\right)^{6} \quad (\mathbf{C.6})$$

Storage Factor, in BTU hr^{-1}

$$(\text{Area}_{floor} * DF_{residential} * -1 * F_{SF})^{7} \quad (\mathbf{C.7})$$

Effect of Outdoor Air, in BTU hr^{-1}

$$((\mathbf{1}) * \Delta T_f * 1.08)^{8} \quad (\mathbf{C.8})$$

Effective Room Sensible Heat, in BTU hr^{-1}

(3)+(4)+(5)+(6)+(7)[9] (**C. 9**)

Total Room Sensible Heat, in BTU hr^{-1}

(9)+(8) (**C. 10**)

---

[4] W and w refer to the number of outside walls, $A_w$refers to the area of each outdoor wall, in m^2. ΔT_x refers to the equivalent temperature difference between outdoor and indoor environments, *not* the actual temperature difference. Its assumptions and equations are found in [**Car65**]. U_x refers to the thermal transmission coefficient of material x, in BTU $hr^{-1}ft^{-2}°F^{-1}$. NOTE: in the winter months, ΔT_x becomesΔT_f, the actual temperature difference, defined by outdoor-indoor temperatures.

[5] G and g refer to the total number of windows in the conditioned space, A_{CorF} is the area of the ceiling or floor of the conditioned space, in m^2. $U_{CorF,Up\ or\ Down}$ refers to the thermal conductivity of the ceiling or floor, in BTU $hr^{-1}ft^{-2}°F^{-1}$. The third term is included if and only if **(2) > (1)**.

[6] P and p refer to the number of people currently in the conditioned space. App and app refer to the number of appliances in the conditioned space. "eff" refers to the efficiency of the appliance, in a decimal format between 0 and 1. The factor $\frac{1}{3413}$ is the factor changing the value from BTU hr^{-1} to kW [**Bel082**].

[7] $DF_{residential}$ refers to the diversity factor for residential spaces

[8] F_{BF} refers to the bypass factor for residential spaces.

[9] If the HVAC system is based on CAV, the sum of all zones' sensible heat loads constitutes the Effective Sensible Heat Load

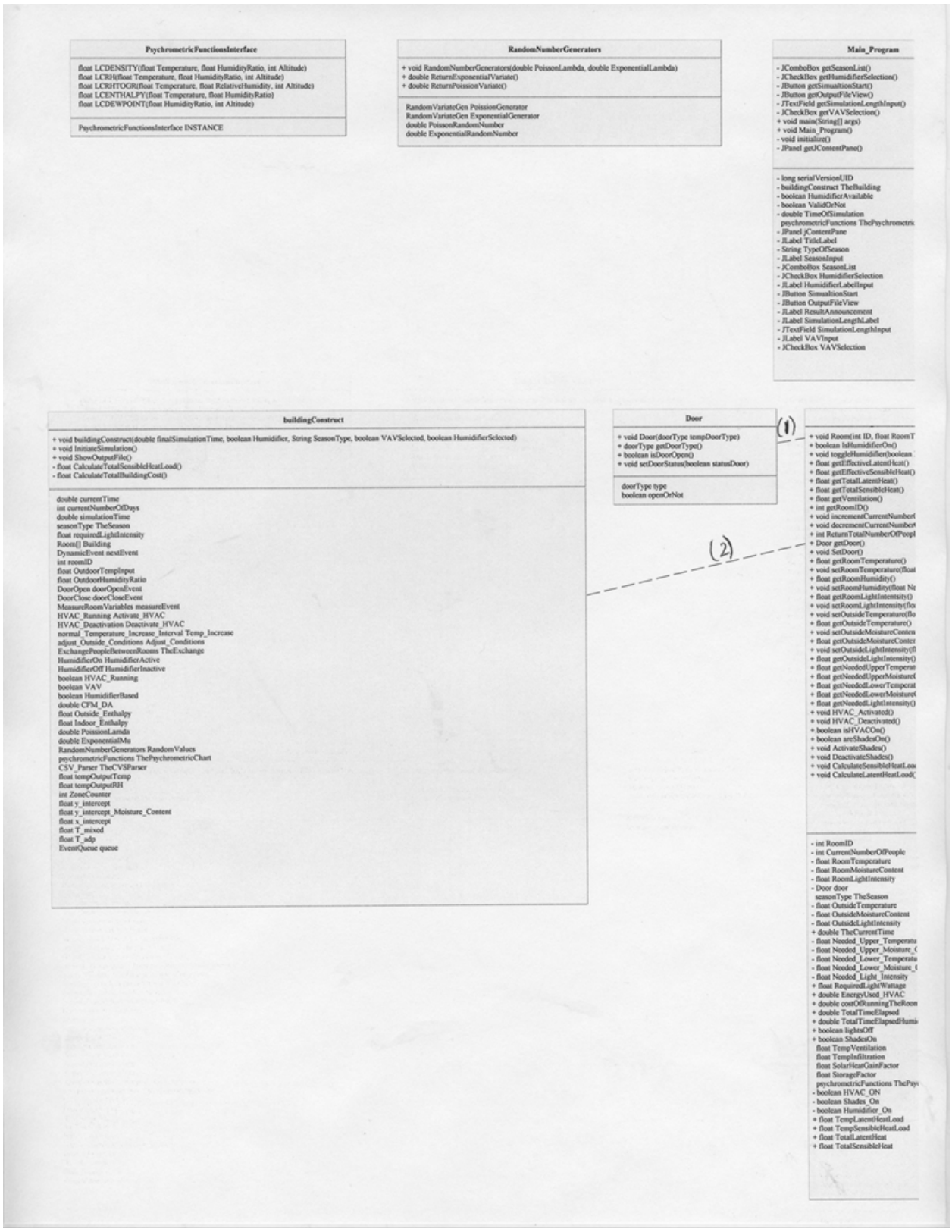
Latent Heat Calculations	
Latent Heat Gain/Loss from Infiltration, in BTU hr^{-1}	
$\left(((\mathbf{2}) - (\mathbf{1})) * \Delta MC * 0.68\right)$ [10]	$(\mathbf{C.11})$
Latent Heat Gain from People, in BTU hr^{-1}	
$\sum_{p=1}^{P} [\text{Heat Gain}_{\text{Latent}}]$	$(\mathbf{C.12})$
Effect from Water Vapour Transmission (same equation applies for Wall, ceiling, and floor; all elements must be added together) , in BTU hr^{-1}	
$\left(A_{\text{part of home}} * 10.764 * \frac{1}{100} * \Delta MC * H_2O \text{ Permeance}_{\text{part of home}}\right)$ [11]	$(\mathbf{C.13})$
Effective Room Latent Heat, in BTU hr^{-1}	
(11)+(12)+(13)	$(\mathbf{C.14})$
Effect of Outside Air, in BTU hr^{-1}	
$((\mathbf{1}) * \Delta MC * 0.68)$	$(\mathbf{C.15})$
Total Room Latent Heat, in BTU hr^{-1}	
(14)+(15)	$(\mathbf{C.16})$
Dehumidified Air Quantity, in cfm	
$\frac{(\mathbf{10})}{\mathbf{1.08} * (1 - F_{BF}) * (T_{room} - (\mathbf{18}))}$	$(\mathbf{C.17})$
Apparatus Dewpoint Temperature, t_{adp}, in °F	
$\frac{((\mathbf{19}) - T_{room})}{1 - F_{BF}}$	$(\mathbf{C.18})$
Mixed Air Temperature (see Section 3.1.5), in °F	
$\left(T_{room} * \frac{(\mathbf{1})}{(\mathbf{17})}\right) + \left(T_{outside} * \frac{(\mathbf{17}) - (\mathbf{1})}{(\mathbf{17})}\right)$	$(\mathbf{C.19})$

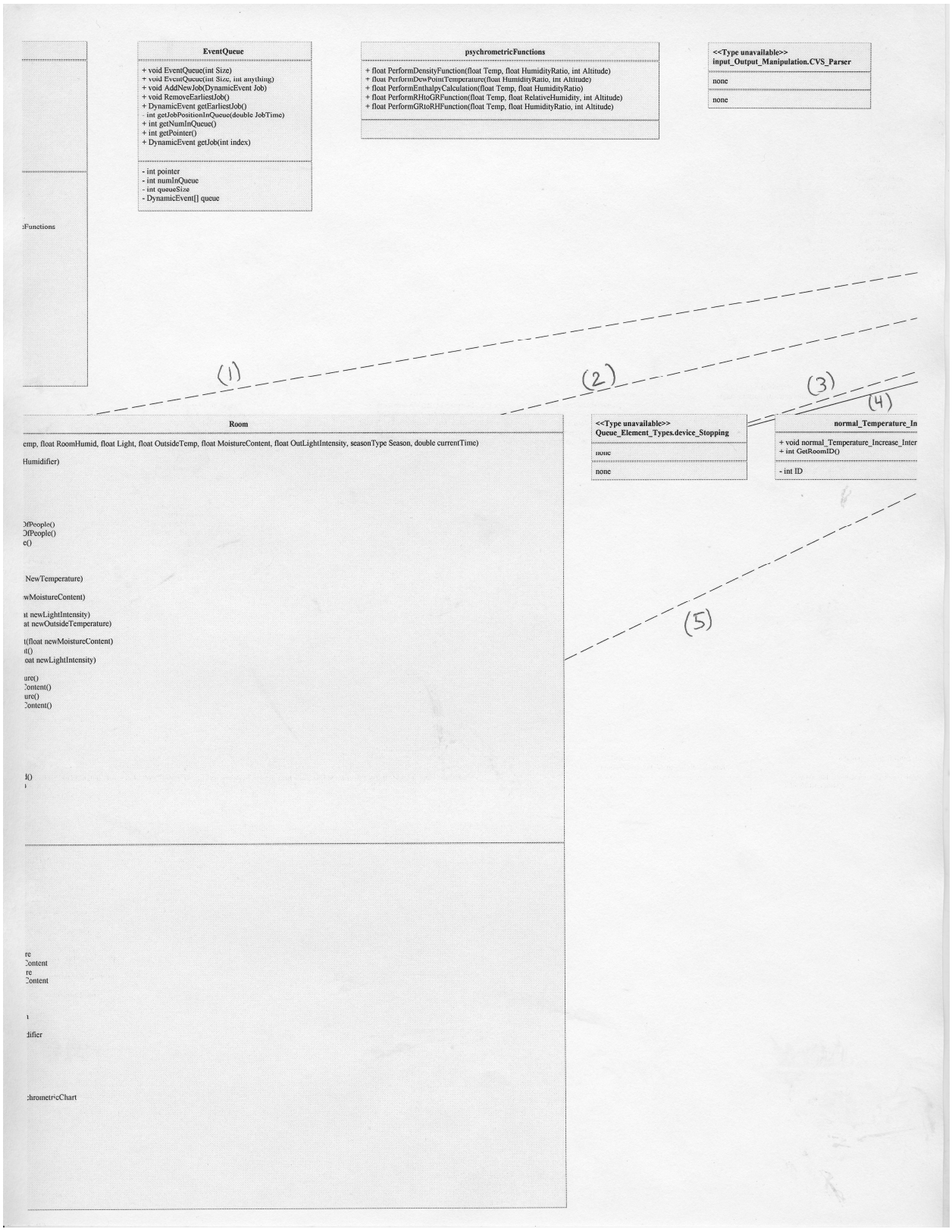
Table C.1: The zone load equations extracted from [**Car65**]

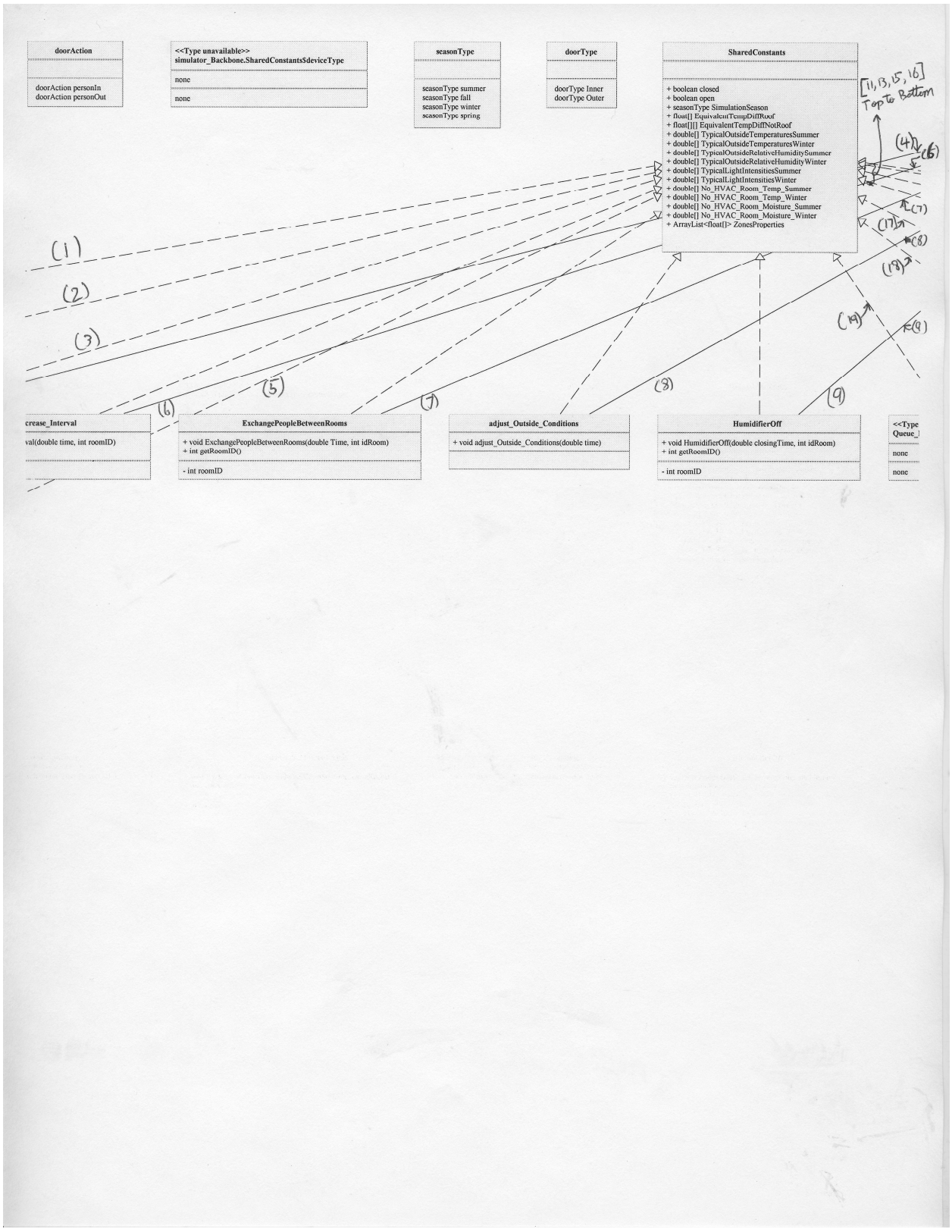
[10] This term is added if and only if **(2) > (1)**.
[11] "Part of home" refers to the wall, ceiling, or to the floor.

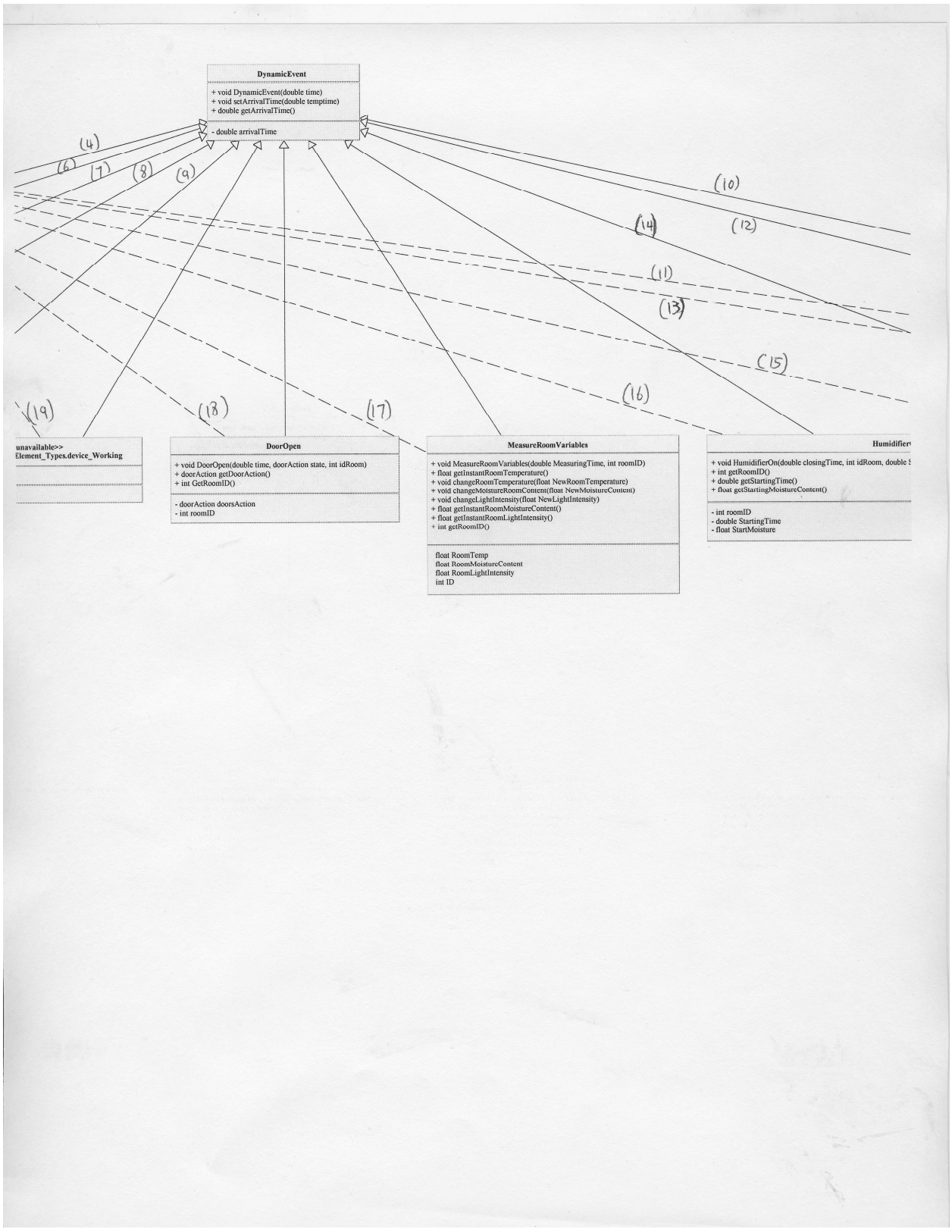
Appendix D: CEMA Simulator Class Diagram

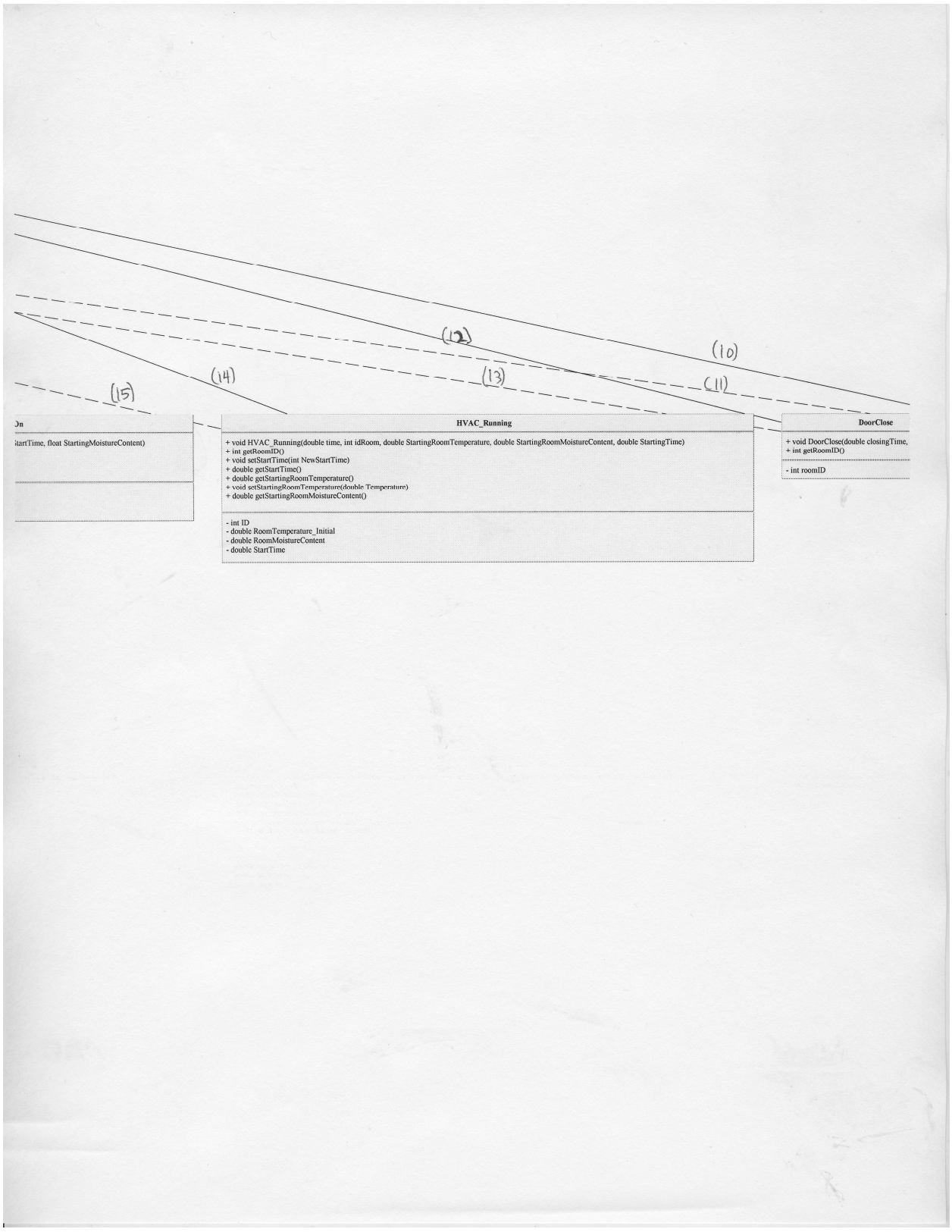
The UML diagram that starts from the next page has been automatically generated by using the Slime UML feature, built on the Eclipse IDE, and which is installable from http://www.slimeuml.de/. This powerful tool reverse engineers Java© code to automatically draw class diagrams that show the complete contents of the classes, the type of these contents (attributes, public methods, private methods, etc) and the relationship between the classes and each other. Another important note about the arrows seen in the ensuing diagram; one type is made of straight lines, and the other dotted lines. The straight lines are ones denote a concept in OOP called inheritance, meaning that all the parent classes' attributes are cloned to the child class. In this thesis, all the different types of queue elements inherit from the parent class DynamicEvent.java. The second type of arrows denote another concept in OOP called implementation, meaning that a generic class (in the diagram, it is SharedConstants.java) is implemented in different forms for different classes with different functionalities. The only thing in common between these classes is the attributes of the generic class, as was explained before.

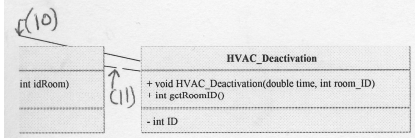

PsychrometricFunctionsInterface
float LCDENSITY(float Temperature, float HumidityRatio, int Altitude)
float LCRH(float Temperature, float HumidityRatio, int Altitude)
float LCRHTOGR(float Temperature, float RelativeHumidity, int Altitude)
float LCENTHALPY(float Temperature, float HumidityRatio)
float LCDEWPOINT(float HumidityRatio, int Altitude)
PsychrometricFunctionsInterface INSTANCE
RandomNumberGenerators
+ void RandomNumberGenerators(double PoissonLambda, double ExponentialLambda)
+ double ReturnExponentialVariate()
+ double ReturnPoissionVariate()
RandomVariateGen PoissionGenerator
RandomVariateGen ExponentialGenerator
double PoissonRandomNumber
double ExponentialRandomNumber
Main_Program
- JComboBox getSeasonList()
- JCheckBox getHumidifierSelection()
- JButton getSimualtionStart()
- JButton getOutputFileView()
- JTextField getSimulationLengthInput()
- JCheckBox getVAVSelection()
+ void main(String[] args)
+ void Main_Program()
- void initialize()
- JPanel getJContentPane()
- long serialVersionUID
- buildingConstruct TheBuilding
- boolean HumidifierAvailable
- boolean ValidOrNot
- double TimeOfSimulation
psychrometricFunctions ThePsychrometrix
- JPanel jContentPane
- JLabel TitleLabel
- String TypeOfSeason
- JLabel SeasonInput
- JComboBox SeasonList
- JCheckBox HumidifierSelection
- JLabel HumidifierLabelInput
- JButton SimualtionStart
- JButton OutputFileView
- JLabel ResultAnnouncement
- JLabel SimulationLengthLabel
- JTextField SimulationLengthInput
- JLabel VAVInput
- JCheckBox VAVSelection
buildingConstruct
+ void buildingConstruct(double finalSimulationTime, boolean Humidifier, String SeasonType, boolean VAVSelected, boolean HumidifierSelected)
+ void InitiateSimulation()
+ void ShowOutputFile()
- float CalculateTotalSensibleHeatLoad()
- float CalculateTotalBuildingCost()
double currentTime
int currentNumberOfDays
double simulationTime
seasonType TheSeason
float requiredLightIntensity
Room[] Building
DynamicEvent nextEvent
int roomID
float OutdoorTempInput
float OutdoorHumidityRatio
DoorOpen doorOpenEvent
DoorClose doorCloseEvent
MeasureRoomVariables measureEvent
HVAC_Running Activate_HVAC
HVAC_Deactivation Deactivate_HVAC
normal_Temperature_Increase_Interval Temp_Increase
adjust_Outside_Conditions Adjust_Conditions
ExchangePeopleBetweenRooms TheExchange
HumidifierOn HumidifierActive
HumidifierOff HumidifierInactive
boolean HVAC_Running
boolean VAV
boolean HumidifierBased
double CFM_DA
float Outside_Enthalpy
float Indoor_Enthalpy
double PoissionLamda
double ExponentialMu
RandomNumberGenerators RandomValues
psychrometricFunctions ThePsychrometricChart
CSV_Parser TheCVSParser
float tempOutputTemp
float tempOutputRH
int ZoneCounter
float y_intercept
float y_intercept_Moisture_Content
float x_intercept
float T_mixed
float T_adp
EventQueue queue
Door
+ void Door(doorType tempDoorType)
+ doorType getDoorType()
+ boolean isDoorOpen()
+ void setDoorStatus(boolean statusDoor)
doorType type
boolean openOrNot
(1)
(2)
+ void Room(int ID, float RoomT
+ boolean IsHumidifierOn()
+ void toggleHumidifier(boolean
+ float getEffectiveLatentHeat()
+ float getEffectiveSensibleHeat()
+ float getTotalLatentHeat()
+ float getTotalSensibleHeat()
+ float getVentilation()
+ int getRoomID()
+ void incrementCurrentNumberO
+ void decrementCurrentNumberO
+ int ReturnTotalNumberOfPeopl
+ Door getDoor()
+ void SetDoor()
+ float getRoomTemperature()
+ void setRoomTemperature(float
+ float getRoomHumidity()
+ void setRoomHumidity(float Ne
+ float getRoomLightIntensity()
+ void setRoomLightIntensity(flo
+ void setOutsideTemperature(flo
+ float getOutsideTemperature()
+ void setOutsideMoistureConten
+ float getOutsideMoistureConter
+ void setOutsideLightIntensity(fl
+ float getOutsideLightIntensity()
+ float getNeededUpperTemperat
+ float getNeededUpperMoisture(
+ float getNeededLowerTemperat
+ float getNeededLowerMoisture(
+ float getNeededLightIntensity()
+ void HVAC_Activated()
+ void HVAC_Deactivated()
+ boolean isHVACOn()
+ boolean areShadesOn()
+ void ActivateShades()
+ void DeactivateShades()
+ void CalculateSensibleHeatLoa
+ void CalculateLatentHeatLoad(
- int RoomID
- int CurrentNumberOfPeople
- float RoomTemperature
- float RoomMoistureContent
- float RoomLightIntensity
- Door door
seasonType TheSeason
- float OutsideTemperature
- float OutsideMoistureContent
- float OutsideLightIntensity
+ double TheCurrentTime
- float Needed_Upper_Temperatu
- float Needed_Upper_Moisture_(
- float Needed_Lower_Temperatu
- float Needed_Lower_Moisture_(
- float Needed_Light_Intensity
+ float RequiredLightWattage
+ double EnergyUsed_HVAC
+ double costOfRunningTheRoon
+ double TotalTimeElapsed
+ double TotalTimeElapsedHumi
+ boolean lightsOff
+ boolean ShadesOn
float TempVentilation
float TempInfiltration
float SolarHeatGainFactor
float StorageFactor
psychrometricFunctions ThePsy
- boolean HVAC_ON
- boolean Shades_On
- boolean Humidifier_On
+ float TempLatentHeatLoad
+ float TempSensibleHeatLoad
+ float TotalLatentHeat
+ float TotalSensibleHeat


EventQueue
+ void EventQueue(int Size)
+ void EventQueue(int Size, int anything)
+ void AddNewJob(DynamicEvent Job)
+ void RemoveEarliestJob()
+ DynamicEvent getEarliestJob()
- int getJobPositionInQueue(double JobTime)
+ int getNumInQueue()
+ int getPointer()
+ DynamicEvent getJob(int index)
- int pointer
- int numInQueue
- int queueSize
- DynamicEvent[] queue
psychrometricFunctions
+ float PerformDensityFunction(float Temp, float HumidityRatio, int Altitude)
+ float PerformDewPointTemperature(float HumidityRatio, int Altitude)
+ float PerformEnthalpyCalculation(float Temp, float HumidityRatio)
+ float PerformRHtoGRFunction(float Temp, float RelativeHumidity, int Altitude)
+ float PerformGRtoRHFunction(float Temp, float HumidityRatio, int Altitude)
<<Type unavailable>>
input_Output_Manipulation.CVS_Parser
none
none
(1)
(2)
(3)
(4)
(5)
Room
emp, float RoomHumid, float Light, float OutsideTemp, float MoistureContent, float OutLightIntensity, seasonType Season, double currentTime)
Humidifier)
<<Type unavailable>>
Queue_Element_Types.device_Stopping
none
none
normal_Temperature_In
+ void normal_Temperature_Increase_Inter
+ int GetRoomID()
- int ID


doorAction
doorAction personIn
doorAction personOut
<<Type unavailable>>
simulator_Backbone.SharedConstants$deviceType
none
none
seasonType
seasonType summer
seasonType fall
seasonType winter
seasonType spring
doorType
doorType Inner
doorType Outer
SharedConstants
+ boolean closed
+ boolean open
+ seasonType SimulationSeason
+ float[] EquivalentTempDiffRoof
+ float[][] EquivalentTempDiffNotRoof
+ double[] TypicalOutsideTemperaturesSummer
+ double[] TypicalOutsideTemperaturesWinter
+ double[] TypicalOutsideRelativeHumiditySummer
+ double[] TypicalOutsideRelativeHumidityWinter
+ double[] TypicalLightIntensitiesSummer
+ double[] TypicalLightIntensitiesWinter
+ double[] No_HVAC_Room_Temp_Summer
+ double[] No_HVAC_Room_Temp_Winter
+ double[] No_HVAC_Room_Moisture_Summer
+ double[] No_HVAC_Room_Moisture_Winter
+ ArrayList<float[]> ZonesProperties
[11,13,15,16]
Top to Bottom
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
crease_Interval
val(double time, int roomID)
ExchangePeopleBetweenRooms
+ void ExchangePeopleBetweenRooms(double Time, int idRoom)
+ int getRoomID()
- int roomID
adjust_Outside_Conditions
+ void adjust_Outside_Conditions(double time)
HumidifierOff
+ void HumidifierOff(double closingTime, int idRoom)
+ int getRoomID()
- int roomID
<<Type
Queue_
none
none


DynamicEvent
+ void DynamicEvent(double time)
+ void setArrivalTime(double temptime)
+ double getArrivalTime()
- double arrivalTime
(4)
(6)
(7)
(8)
(9)
(10)
(12)
(14)
(11)
(13)
(15)
(16)
(17)
(18)
(19)
unavailable>>
Element_Types.device_Working
DoorOpen
+ void DoorOpen(double time, doorAction state, int idRoom)
+ doorAction getDoorAction()
+ int GetRoomID()
- doorAction doorsAction
- int roomID
MeasureRoomVariables
+ void MeasureRoomVariables(double MeasuringTime, int roomID)
+ float getInstantRoomTemperature()
+ void changeRoomTemperature(float NewRoomTemperature)
+ void changeMoisutreRoomContent(float NewMoisutreContent)
+ void changeLightIntensity(float NewLightIntensity)
+ float getInstantRoomMoistureContent()
+ float getInstantRoomLightIntensity()
+ int getRoomID()
float RoomTemp
float RoomMoistureContent
float RoomLightIntensity
int ID
Humidifier
+ void HumidifierOn(double closingTime, int idRoom, double
+ int getRoomID()
+ double getStartingTime()
+ float getStartingMoistureContent()
- int roomID
- double StartingTime
- float StartMoisture


(12)
(10)
(14)
(13)
(11)
(15)
)n
itartTime, float StartingMoistureContent)
HVAC_Running
+ void HVAC_Running(double time, int idRoom, double StartingRoomTemperature, double StartingRoomMoistureContent, double StartingTime)
+ int getRoomID()
+ void setStartTime(int NewStartTime)
+ double getStartTime()
+ double getStartingRoomTemperature()
+ void setStartingRoomTemperature(double Temperature)
+ double getStartingRoomMoistureContent()
- int ID
- double RoomTemperature_Initial
- double RoomMoistureContent
- double StartTime
DoorClose
+ void DoorClose(double closingTime,
+ int getRoomID()
- int roomID


(10)
int idRoom)
(11)
HVAC_Deactivation
+ void HVAC_Deactivation(double time, int room_ID)
+ int getRoomID()
- int ID