A Distributed Intelligent Lighting Solution and the Design and Implementation of a Sensor Middleware System

by

Michael Fischer
Bachelor of Engineering, University of Victoria, 2001
Master of Applied Science, University of Victoria, 2008

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the department of Computer Science

Supervisory Committee

A Distributed Intelligent Lighting Solution and the Design and Implementation of a Sensor Middleware System

by

Michael Fischer
Bachelor of Engineering, University of Victoria, 2001
Master of Applied Science, University of Victoria, 2008

Supervisory Committee

Dr. Kui Wu (Department of Computer Science)
Supervisor

Dr. Pan Agathoklis (Department of Electrical Engineering)
Co-Supervisor

Abstract

Supervisory Committee
Dr. Kui Wu (Department of Computer Science)
Supervisor
Dr. Pan Agathoklis (Department of Electrical Engineering)
Co-Supervisor

This thesis addresses a multi-phase research and development project that spanned nearly four years, targeted at providing an ultra high-efficiency, user-friendly, and economic intelligent lighting solution for commercial facility applications, initially targeting underground parking specifically. The system would leverage the strengths of four key technologies: high brightness white Light Emitting Diodes (LEDs), wireless sensor and actuator networks, single board computers, and cloud computing. An introduction to these technologies and an overview of how they were combined to build an intelligent lighting solution is given, followed by an in-depth description of the design and implementation of one of the main subsystems – the Sensor Middleware System – residing on a single board computer.

Newly-available LED luminaires (a.k.a. light fixtures) bring the combination of high efficiency, reliability, illumination quality, and long-lifetime to the lighting market. Emerging low-power – and recently low-cost – 802.15.4 wireless networks offer high controllability and responsiveness to deployed luminaires and sensors. The cost-associativity, low maintenance, and easy build-up of Internet Data Center "cloud" computing resources make data collection and remote management infrastructure for Building Automation Systems accessible to even small companies. Additionally, these resources can be much more appropriately sized and allocated, which reduces energy use.

These technologies are combined to form an Intelligent Lighting System (ILS). Fitting well within the Internet of Things paradigm, this highly distributed messaging-based "system of systems" was designed to be reliable through loose coupling – spanning multiple network layers and messaging protocols. Its goal was to deliver significant energy savings over incumbent technologies, configurable and responsive lighting service

behaviour, and improved experience for users within the facility (pedestrians and drivers) and those interacting with its web-based tools (building managers and ILS administrators). The ILS was partitioned into three main subsystems as follows. The installed Wireless Field Network (WFN) of luminaires and sensors provided coordinated scheduled and real-time output level adjustment (i.e. dimming), with the help of motion sensor triggers. The Monitoring and Configuration System (MCS) in the cloud provided remote data collection and a web-based monitoring and configuration Graphical User Interface application. Network hardware and Message-Oriented Middleware (MOM) were responsible for tying these subsystems together. The MOM layer that provided the message brokering, translating, envelope wrapping, and guaranteed delivery services between the WFN and MCS, as well as field supervisory and quality-of-service functions for the WFN, was called the Sensor Middleware System (SMS). It was hosted on a single board computer located at the facility.

Table of Contents

Supervisory Committee ii
Abstract iii
Table of Contents v
List of Tables vi
List of Figures vii
List of Acronyms xii
Acknowledgements xiii
Dedication xiv
1 Introduction 1
1.1 Motivation 2
1.2 Enabling Technologies 6
1.2.1 High Brightness White Light Emitting Diodes 6
1.2.2 Building Automation Systems 9
1.2.3 Low-Power Wireless Networks 11
1.2.4 Single Board Computers 12
1.2.5 Cloud Computing and Web Applications 12
2 Overview of the Intelligent Lighting System 15
2.1 Application Domain 17
2.2 System Requirements 18
2.3 System Domain 20
2.4 Wireless Field Network (WFN) 22
2.4.1 Wireless Networking Overview 22
2.4.2 Coordinator-Gateway 25
2.4.3 LED Luminaires 26
2.5 Monitoring and Configuration System (MCS) 29
2.6 Sensor Middleware System (SMS) 34
3 Design and Implementation of the Sensor Middleware System 37
3.1 Requirements 38
3.2 Use Case Model 40
3.3 Messaging Between ILS Subsystems 41
3.4 Message Classes 48
3.4.1 Intra-Facility TCP Datagrams 49
3.4.2 Extra-Facility REST Requests 51
3.5 WFN Node State Data Model 53
3.6 Software Implementation 54
3.6.1 Software Packages and Components 55
3.6.2 Tasks and Worker Processes 62
4 Testing and Performance 69
4.1 ILS Performance 70
4.2 SMS Performance 72
5 Conclusions and Future Work 74
Bibliography 76

List of Tables

Table 2.1: The functional and feature requirements of the Intelligent Lighting System. 20
Table 3.1: The list of requirements for the Sensor Middleware System..... 40
Table 3.2: Mapping of tasks that can be assigned to Celery worker processes for execution, and the queues the worker processes listen on to receive the execution requests and input parameters. 64
Table 4.1: Performance summary of the LED lighting retrofit at the pilot demonstration site, as compared to the legacy lighting installation for LED retrofit only (~65% energy savings), and LED retrofit with ILS control (~90% energy savings). 72

List of Figures

Figure 1.1: Percentage of total energy use in Canada in 2009, broken down by sector (left), and percentage of total commercial building energy use broken down by service / equipment type (right). Lighting accounts for 10.6% of all commercial building energy use, responsible for 6.0 Mt of CO_2 equivalent GHG emissions annually [**1**]. 3
Figure 1.2: Percentage of total energy use in the U.S. in 2010, broken down by sector (left), and percentage of total commercial building energy use broken down by service / equipment type (right). Lighting accounts for 14.0% of all commercial building energy use [**2**] 3
Figure 1.3: Approximate luminous efficacies for common lighting technologies (as of 2013), shown for the bare lamp / LED package, as well as when integrated into a luminaire – taking into account the entire system, including driver, thermal, and optical losses. Only LED technology is expected to make any substantial improvements in luminous efficacy in the near future [**6**]. 7
Figure 1.4: Cumulative probability of failure of LED luminaires. The MTTF for an LED luminaire is ~52,000 hours. Failure mechanisms related to the LED driver power supply and the LEDs themselves are singled out for comparison, indicating that most LED luminaires fail because of driver power supply failures rather than LEDs themselves (which tend to gradually dim rather than fail catastrophically, and there are typically many LEDs in one luminaire) [**9**]. 8
Figure 1.5: Typical lumen depreciation curves for various lighting technologies. High-power (high brightness) LEDs exhibit a very gradual decrease in luminous output over time versus all technologies with the exception of linear T8 fluorescents, however the MTTF of linear fluorescents is only on the order of 20,000 hours (the time horizon shown here), versus over 50,000 hours for LEDs [**9**]. 9
Figure 1.6: The typical three-tiered architecture of a Building Automation System [**14**]. 10
Figure 1.7: A common way to view the abstraction of cloud computing – the infrastructure provider represents the Internet Data Center (IDC) and virtual machine resources on which a service provider can run a standard or custom software stack to provide web-based services to clients. The actual physical computing and network resources of the infrastructure provider are obscured (and generally unimportant) to the service provider and service clients, thus the "cloud" metaphor (Davide Lamanna, 2011). 14
Figure 2.1: High-level architectural diagram of the Intelligent Lighting System (ILS). At the BAS field level the Wireless Field Network (WFN) consists of luminaires, each with an embedded System-on-Chip with wireless radio, that are coordinated in control actions and data collection by Coordinator-Gateway (CG) devices at the top of each WFN. The automation level is shared by CGs and the Sensor Middleware System (SMS), implementing lighting zone coordination, scheduled operations, sensor data reporting, configuration update command handling, network health monitoring, and firmware update functionalities. The management level is located offsite in the cloud (Internet Data Center infrastructure), implemented with virtual machines hosting monitoring and configuration databases and associated data intake and query engines, and web-facing Graphical User Interfaces. 15

Figure 2.2: A photograph of the multi-level underground parking area used in the demonstration and pilot scale installations of the ILS. A section of the original lighting installation is shown, in total consisting of 89 x 140 Watt MH HID luminaires. The very limited spectral coverage of MH lighting is evident by the orange hue and poor colour rendering of the vehicles in the image, as well as poor contrast between directly lit and shaded areas (these 89 MH HID luminaires were replaced by 52 white LED luminaires at the pilot stage). 18
Figure 2.3: UML conceptual class diagram of the ILS domain model, showing the attributes key to tying together this distributed messaging-based system, as well as those that define the behavioural parameters of the installed luminaires. Emphasis is given in this diagram to entities of the ILS physically residing within the commercial building facility. 22
Figure 2.4: Star and cluster tree wireless network topologies (supported by JenNet [**47**] and ZigBee [**26**]). Data packet routing is simple as only one possible route exists between any two nodes. Nodes in the hierarchy are referred to as *parent* or *child* nodes, depending on their relative position to other nodes. In the cluster tree topology, Router nodes act as parents within a given cluster, typically serving as data and collection / command distribution points for the End Device children. Note that there theoretically may be many levels of Router nodes, and End Devices are optional. 23
Figure 2.5: Mesh wireless network topology (supported by ZigBee [**26**]). Data packet routing is complex and limitations exist on the number of hops possible between any two devices 24
Figure 2.6: The protocol stack layers for the IEEE 802.15.4 standard. Only the bottom two layers are common among JenNet, ZigBee, and other such proprietary networks, leaving considerable flexibility for 3rd parties to create customized wireless network topologies and many other features [**26**] 24
Figure 2.7: A block diagram of the NXP JN5148 System-on-Chip used to implement JenNet [**47**] 25
Figure 2.8: Conceptual design of the luminaire. Complete telemetry reporting and redundancy in control functionalities and critical lighting components, as well as dynamic fault recovery, make the luminaire highly reliable and fail-safe (H. Davis 2012). 27
Figure 2.9: State machine of the event-based behaviour of the luminaire. Periodic reporting by the luminaire to the CG, and network management activities (e.g. connection / re-connection sequences) are not depicted here. 28
Figure 2.10: A screenshot of the summary view of the web-based MCS GUI. The view defaults to present the past 7 days of daily ILS energy use as well as the past 24 hours of hourly ILS energy use activity for a Facility. 32
Figure 2.11: A screenshot of a sensor data report view of the web-based MCS GUI. This example shows energy use for a selected luminaire (address 6.0) for a specific queried date and time range. 32
Figure 2.12: A screenshot of a sensor data report view of the web-based MCS GUI. This example shows the state of an integrated passive infrared motion detection sensor for a selected luminaire (address 6.0) for a specific queried date and time range. 33
Figure 2.13: A screenshot of a floor plan view of the web-based MCS GUI. This example shows a section of the layout of the underground parking garage pilot facility, indicating luminaire node locations and operating states with coloured circles. Solid

green circles indicate an installed and commissioned node that is operating normally, blue indicates a currently selected node, and red outlines indicate nodes detected as having a *disconnected* communication state (i.e. they are currently unreachable)..... 33
Figure 2.14: A photo of the Raspberry Pi (Model B, Rev 2) Single Board Computer [**35**]. 36
Figure 3.1: UML context diagram illustrating the use cases that the SMS is responsible for supporting..... 41
Figure 3.2: Architecture of the Transmission Control Protocol / Internet Protocol (TCP/IP) stack [**59**]..... 42
Figure 3.3: The *Message Broker* design pattern [**54**]. A central broker can receive messages from multiple sources, determine the destination, and direct the message to the correct channel so as to reach the destination..... 43
Figure 3.4: UML sequence diagram showing the message-based interaction flow beginning at a luminaire and (optionally) ending at the MCS. A luminaire sends an asynchronous event message to its CG over the wireless network without acknowledgement. Depending on the message type, the CG can then translate and forward it over the LAN to the SMS. Similarly, depending on the message type, the SMS can then translate and forward the message over the Internet to the MCS..... 44
Figure 3.5: The *Translator* pattern [**54**]. A key function of the SMS is to translate incoming messages from one format to another for subsequent transmission to another subsystem. The *Translator* pattern is implemented in the SMS for communications in both directions, from the CG through to the MCS, and vice-versa. 45
Figure 3.6: UML sequence diagram showing the message-based interaction flow beginning at the MCS and (optionally) ending at a luminaire node. The MCS sends an asynchronous command message to a facility's SMS over the Internet and waits for acknowledgement. Depending on the message type, the SMS can then translate and forward it over the LAN to the CG associated with the target luminaire node (or send to the CG itself). Similarly, depending on the message type, the CG can then translate and forward the message over the wireless network to the luminaire node. 46
Figure 3.7: The *Envelope Wrapper* pattern [**54**]. It is used to wrap data inside an envelope that is compliant with the messaging system. The destination subsystem unwraps the message upon arrival, revealing the original message contents. 47
Figure 3.8: Encapsulation of message data in from the link layer to the application layer using TCP/IP [**61**]. Intra-facility messages between the SMS and CGs are wrapped in TCP/IP datagrams (labeled here as the TCP segment). Messages between the SMS and MCS are wrapped in HTTP REST requests at the application layer. The destination IP address is provided in the IP header, and the destination port is provided in the TCP header. For HTTP, the REST request header is provided in the application header. 47
Figure 3.9: The *Guaranteed Delivery* pattern [**54**]. To ensure messages are delivered to their intended destination, the messaging middleware is responsible for storing them on disk and retrying transmission until acknowledged successful by the recipient. 48
Figure 3.10: UML class diagram describing the data payload of TCP datagram messages sent by facility CGs to the SMS..... 50
Figure 3.11: UML class diagram describing TCP datagram messages sent by the SMS to the CGs managing WFNs within its facility..... 51

Figure 3.12: UML class diagram describing HTTPS REST request messages sent by the SMS to the MCS. 52
Figure 3.13: UML class diagram describing HTTPS REST request messages received by the SMS from the MCS. 53
Figure 3.14: UML class diagram describing the data model of the WFN Node State Database implemented by the SMS. 54
Figure 3.15: Software package diagram showing the various software layers of the SMS: Domain, Application, Presentation, and Technical Services. All of the packages used were either developed in Python and/or offer a Python API. 55
Figure 3.16: Implementation of two of the WFN->SMS Messages classes shown in Figure 3.10 (*ValidEventObject* and *SingleMotionSensingEvent*, respectively). 56
Figure 3.17: Sample code showing usage of the Python HTTP *requests* module, with SSL certificates and no 3rd party authority verification. 56
Figure 3.18: Two samples of JSON object templates that form the payloads of an outgoing *LuminaireEnergyReport* message to the MCS, and an incoming *DownloadNodeFirmware* message from the MCS. 57
Figure 3.19: A screenshot of the main page of the SMS Administration Web Application, implemented using Django's built-in Admin GUI and Authorization plugins [**52**]. It allows for easy inspection and adjustment of WFN Node State Database entries, and periodic and event-driven Celery task and worker process status and configurations. 58
Figure 3.20: Code snippet, using Django's data model description syntax, of the *Node* class of the WFN Node State Data Model. 60
Figure 3.21: UML component diagram showing the main application layer components of the SMS for a single minimal deployment. Execution flows are grouped into logically parallel channels, one each for MCS→WFN and WFN→MCS communications processing, and one for handling scheduled tasks. 62
Figure 3.22: Execution of the task `send_msg_to_mcs()` is requested via the Celery `apply_async()`call, which puts a message on the `http_msg_tx_q` (for an *HTTPMessageSenderWorker* process to consume). 63
Figure 3.23: UML activity diagram describing the execution algorithm of the `process_tcp_msg()` task, which is consumed from the *TCPMessageInQueue* and carried out by a *TCPMessageProcessorWorker* process. 65
Figure 4.1: The physical layout of the ILS pilot scale installation. Two of the three levels of the spiral configuration underground parking facility are shown, with luminaires represented by yellow circles (each with an embedded PIR motion sensor facing downward). Lighting zones are indicated by the dashed suffix of each alphanumeric text code below each luminaire. Microwave motion sensors are represented by blue rectangles, their effective sensing arcs and ranges shown in blue shading. These are positioned to detect pedestrians as they enter the parking area from stairwells and elevator lobbies, outside of luminaire PIR motion sensor coverage, as well as moving vehicles that are cold (i.e. recently started, and thus not having a significant infrared heat signature). The single Coordinator-Gateway is represented by the grey box labeled "CG", located where a LAN connection was made available. The SMS is not shown, being located in the facility server room on another floor (H. Davis 2014). 69

Figure 4.2: A screenshot of the summary view of the web-based GUI of the Prototype MCS, summarizing the key historical performance and activity metrics of the ILS. The System Activity pane shows aggregate luminaire load as it varies according to motion detection events (and scheduled operating policies)..... 70
Figure 4.3: A screenshot of the fixture (luminaire) view of the web-based GUI of the pMCS, allowing the user to drill down to query historical activity, current operating state values, and fixed attributes of specific luminaires in the WFN. Activity for a single luminaire is shown, its load in Watts fluctuating with motion detection events. 71
Figure 4.4: A screenshot of the summary view of the web-based MCS GUI for the pilot ILS facility. The view defaults to present both the past 7 days of daily energy use and the past 24 hours of hourly energy use. 71

List of Acronyms

AMQP	Asynchronous Message Queuing Protocol
API	Application Programming Interface
AWS	Amazon Web Services
BAS	Building Automation System
BEMS	Building Energy Management System
CCTV	Closed Circuit Television
CG	Coordinator-Gateway
CPU	Central Processing Unit
FIFO	First In First Out
GHG	Greenhouse Gas
GUI	Graphical User Interface
HID	High Intensity Discharge
HTTP	Hyper Text Transfer Protocol
HVAC	Heating, Ventilation and Cooling
IDC	Internet Data Center
ILS	Intelligent Lighting System
JMS	Java Messaging Service
JSON	JavaScript Object Notation
LAN	Local Area Network
LCU	Light Control Unit
LED	Light Emitting Diode
MAC	Media Access Control
MCS	Monitoring and Configuration System
MH	Metal Halide
MOM	Message-Oriented Middleware
MSMQ	Microsoft Message Queuing
MTTF	Mean Time to Failure
NTP	Network Time Protocol
PCB	Printed Circuit Board
PIR	Passive Infrared
PWM	Pulse Width Modulation
RAM	Random Access Memory
REST	Representational State Transfer
RF	Radio Frequency
SBC	Single Board Computer
SMS	Sensor Middleware System
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Universal Modeling Language
URL	Universal Resource Locator
VM	Virtual Machine
WCU	Wireless Control Unit
WFN	Wireless Field Network

Acknowledgements

I would like to sincerely thank Dr. Pan Agathoklis and Dr. Kui Wu for taking me on as a graduate student, and coming through to help me whenever I needed it throughout my degree. Your patience, support, guidance, and encouragement were very much appreciated. I would like to acknowledge the very hard work and camaraderie of my colleagues Gaspare Boscarino, Howard Davis, Gail Forbes, Alan Graves, Jin Lei, Younes Rashidi, and Blair Wilson. You are all talented professionals, and incredibly decent people. Thank you for sharing this interesting adventure with me, and making it enjoyable. I learned much from each of you, and am happy to have spent that time together and gotten to know you.

Dedication

This is dedicated to the late Dr. Siavash Vojdani,

who mentored me and believed in my abilities,

beyond what I realised I possessed.

1 Introduction

Lighting is a major factor in workplace comfort, productivity, and stress. Modern commercial environments strive to improve these conditions by selection of better lighting technologies (together with wall, floor, and furniture colours and textures) to produce a more pleasing photometric experience for occupants. Intelligent lighting systems are tasked with ensuring that occupants receive adequate amounts and quality of illuminance needed at all times to work comfortably and safely, while at the same time reducing energy use (and related greenhouse gas emissions) and cost of operation through lighting controls (on/off/dimming). Time of day scheduling and motion sensors are the most common efficiency measures, and where applicable, making use of daylight infiltration – often called daylight "harvesting" – provides additional opportunities to save energy and improve occupant comfort. Continuing improvements in light source technologies are benefiting intelligent lighting systems performance. High brightness white Light Emitting Diodes (LEDs) are surpassing incumbent light sources in luminous efficacy (lumens per Watt), as well as offering more fine-grained and responsive dimming controllability, improved spectral quality, and drastically reduced maintenance and replacement requirements. They have also become cost competitive with incumbent lighting technologies in commercial applications.

High brightness white LED luminaires are a cornerstone to the intelligent lighting solution we developed that is described in this thesis. We identified advancements in computing and networking technologies – namely cloud computing, Single Board Computers, and low-power wireless networks – that could be combined with LED lighting to further improve upon the state-of-the-art. The integration of these technologies held promise to exceed the performance of existing intelligent lighting systems across metrics of energy use and greenhouse gas emissions, operation and maintenance costs, operator ease of use, and occupant satisfaction.

Our intelligent lighting solution was applied to commercial indoor parking to prove these assumptions – an application typified by entire lighting installations being left powered at full output 24 hours per day, 7 days per week. This application and

environment was ideal for proving key features and functionalities of intelligent lighting at demonstration and pilot scales, where users are generally passing through the areas only for short periods (meaning temporary interruption or system glitches are generally more tolerable). Many of the features and functionalities to be proven in this context are common with other commercial building applications such as warehouse lighting, common spaces, and office lighting. This was thus seen as a logical first step toward developing an intelligent lighting product that would potentially expand to address all these areas within the commercial and institutional buildings lighting market.

The nature of the technologies we chose naturally led to a messaging-based distributed system solution, spanning from wireless field sensors and actuators to Internet Data Centers, potentially thousands of kilometers apart. This thesis begins with describing the motivation behind developing a new distributed LED-based Intelligent Lighting System, then introduces the existing and emerging technologies that made its development feasible, followed by an overview of the system architecture and main subsystem components of its implementation. The focus then narrows to the software design and implementation of one subsystem – the Sensor Middleware System. This dedicated Linux-based Single Board Computer forms the critical link responsible for providing the messaging interface between the installed physical lighting assets in the field and distant cloud-based infrastructure and intelligent lighting management software. The remainder of the thesis will summarize the performance of the Intelligent Lighting System developed, at both demonstration and pilot scales.

1.1 Motivation

According to the Natural Resources Canada Energy Use Data Handbook 2012 [**1**], in Canada in 2009 buildings accounted for 30.5% (2,608 PJ) of total energy use and 27.8% (128.8 Mt of CO_2 equivalent) of greenhouse gas (GHG) emissions. As shown in Figure 1.1, commercial and institutional buildings in particular accounted for 13.9% (1,186.0 PJ) of total energy use, 13.1% (60.9 Mt of CO_2 equivalent) of GHG emissions. Within the sector of commercial and institutional buildings, the top three energy users were space heating, lighting, and water heating. Lighting was responsible for 10.6% of energy use (126.0 PJ) and 9.8% (6.0 Mt of CO_2 equivalent) of GHG emissions, due to

emissions in primary electricity production. Figure 1.2 shows a similar energy use breakdown percentagewise for the U.S. for 2010, space cooling taking the place of water heating, mostly due to the warmer average climate than Canada.

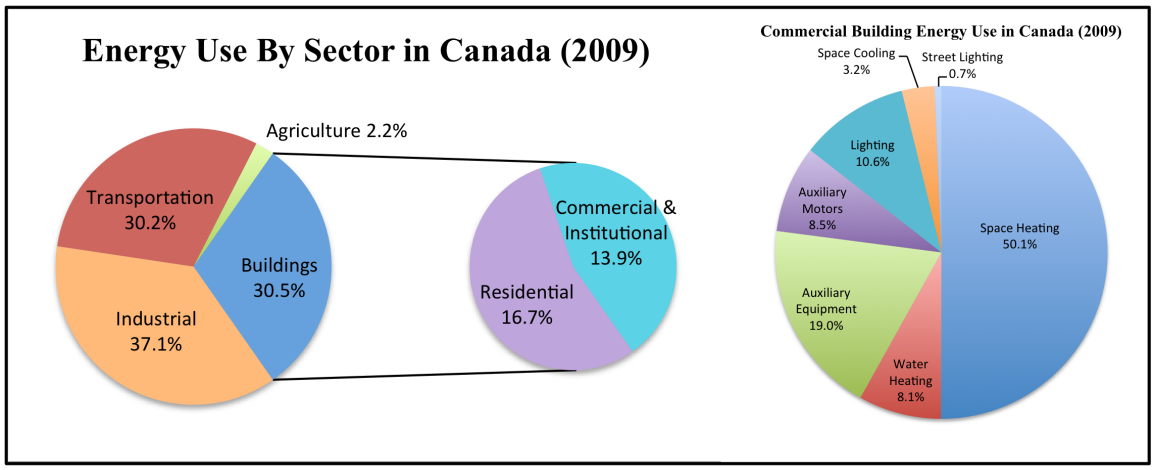

Figure 1.1: Percentage of total energy use in Canada in 2009, broken down by sector (left), and percentage of total commercial building energy use broken down by service / equipment type (right). Lighting accounts for 10.6% of all commercial building energy use, responsible for 6.0 Mt of CO_2 equivalent GHG emissions annually [**1**].

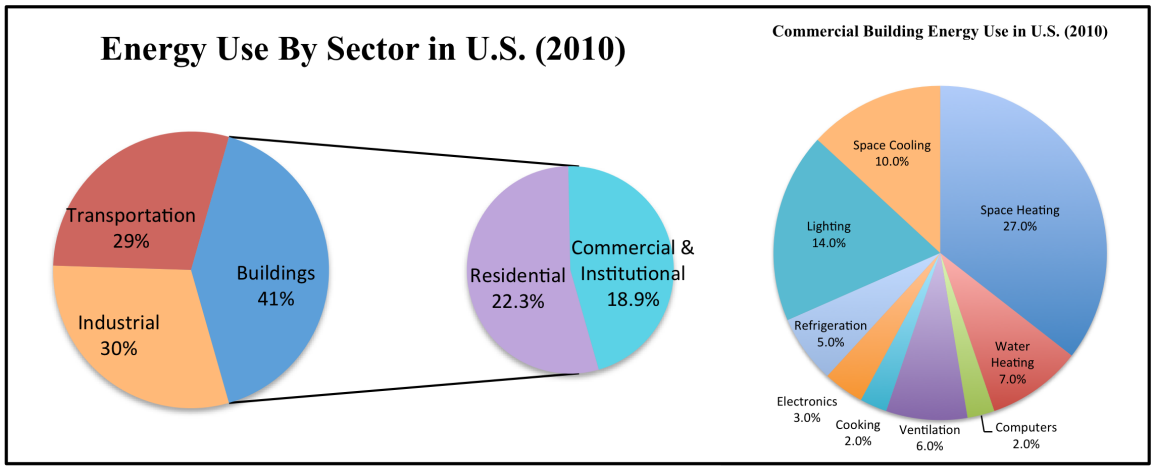

Figure 1.2: Percentage of total energy use in the U.S. in 2010, broken down by sector (left), and percentage of total commercial building energy use broken down by service / equipment type (right). Lighting accounts for 14.0% of all commercial building energy use [**2**].

The commercial and institutional building sector in Canada has shown significant growth in electricity use, as the service and information technology (IT) sectors have grown as a result of structural changes to the economy. This has resulted in an increase in the number of commercial buildings and floor space, and thus larger areas to heat, cool

and illuminate, while computers, printers and other electrical appliances have become ubiquitous. Between 1990 and 2006 (the last year for which data is available), commercial and institutional floor space increased 31% and 10%, respectively [**3**]. This trend has obvious implications for the continued growth of lighting as a major energy user in this sector.

As lighting is powered purely by electricity generation (i.e. there is no opportunity for fuel switching), GHGs generated on its behalf can vary widely depending on the jurisdictional electricity generation mix. For example, British Columbia, its electricity generation being approximately 90% hydroelectric, has an emissions intensity of 9.0 gCO_2 equivalent per MWh; Alberta, being largely coal, has a much higher emissions intensity of 270 gCO_2 equivalent per MWh [**3**] [**4**]. Thus, seeking efficiencies in lighting has the potential for significantly improving the direct ecological footprint of a building. In areas with lower electricity generation emissions intensities, reducing the amount of aggregate load on the electrical grid through improved energy efficiency in lighting can allow for switching other building services to using electricity from higher emissions sources (e.g. electric heat pumps in place of oil fired furnaces).

We strove to develop an LED luminaire-based intelligent lighting solution that would improve energy efficiency and greenhouse gas emissions, would improve user experience, and be economical, robust and long-lasting. We chose to address lighting in a commercial building context instead of residential, for a number of reasons:

- commercial lighting is more standardized in form factor and installation methods (e.g. tiled ceilings),
- desirable lighting levels for particular use cases / tasks are standardized by an international body [**4**],
- the range of style options in commercial lighting is smaller, and individual preferences are less important,

- commercial lighting is operated according to more predictable schedules and patterns, thus user-friendly automation is easier to achieve,
- interior layouts of buildings are less varied, and architectural lighting design is more standardized,
- energy and cost savings potential on a per-customer or per-building basis is generally greater because of greater daily lighting operating hours and economies of scale of large installations, and thus the payback time on investment in an intelligent lighting system is lesser (and return on investment is greater),
- commercial lighting has a higher duty cycle (i.e. more operating hours over a year), and maintenance / lamp replacement of incumbent lighting technologies constitute a major cost component in operation and maintenance,
- commercial customers are generally more able to afford (or access financing) efficiency upgrades,
- commercial customers are generally more accustomed to some level of automation in their building systems, including lighting.

Within the commercial building sector we chose to specialize in one intelligent lighting application to begin with: indoor and covered parking areas. This was judged to be ideal for the development and test of intelligent lighting features and functionalities at demonstration and pilot scales. Many of these features and functionalities are common between this application and others within a commercial building context such as warehouse lighting, common spaces, and office lighting. Given that indoor parking areas are typically illuminated 24 hours per day, the gross energy and cost savings potential for customers installing a high efficiency lighting system was greatest within the commercial building context. Additionally, temporary service interruption, operation glitches, and engineers/integrators working on-site with the intelligent lighting system are generally

less disruptive when happening in areas that occupants do not spend much time in throughout the course of a day.

1.2 Enabling Technologies

Typical features of existing automated lighting systems in commercial building applications include luminaire zoning, motion / occupancy sensing, dimmable luminaires, scheduled switching or dimming, and daylight "harvesting" (i.e. luminaires are dimmed in response to increased daylight infiltration). To enable these features, luminaires[1] are typically paired with sensors (either integrated or externally connected), and some kind of coordinated control among luminaires is implemented via a network. It is the sensing of environmental conditions and events, networking and addressability among sensors and actuators (luminaires) – and algorithms to coordinate control among these elements in order to meet simultaneous efficiency and user satisfaction performance goals – that are essential in making a lighting system "intelligent". The motivations of the ILS have already been stated; designing a solution to address them was made possible by the combination of the technologies summarized in the following subsections of this chapter.

1.2.1 High Brightness White Light Emitting Diodes

Lighting in commercial applications has been dominated for the past few decades by linear fluorescent and High Intensity Discharge (HID) lamp technologies. Luminous efficacy, the measure of lumens of output brightness per Watt, has reached over 100 lm/W and 110 lm/W, respectively [**5**] [**6**], as shown in Figure 1.3. While efficiencies of these technologies have improved over time, they are not expected to improve much further [**7**], and operating mean time to failure (MTTF) has been limited to 20,000 hours [**5**] and 24,000 hours [**6**], respectively. This can result in lighting maintenance costs (lamp replacement plus labour) account for a significant portion of total building operation and maintenance budgets. Coupling these lighting technologies with intelligent controls can also be problematic. On-switching with fluorescents can have a delay lasting up to a couple seconds, and HID lamps can require up to 10 minutes warm-up time, and thus are unsuitable for any motion sensing-based control [**6**], and dimming is not possible with HID lighting.

[1] Luminaires are commonly referred to as "light fixtures", or just "fixtures".

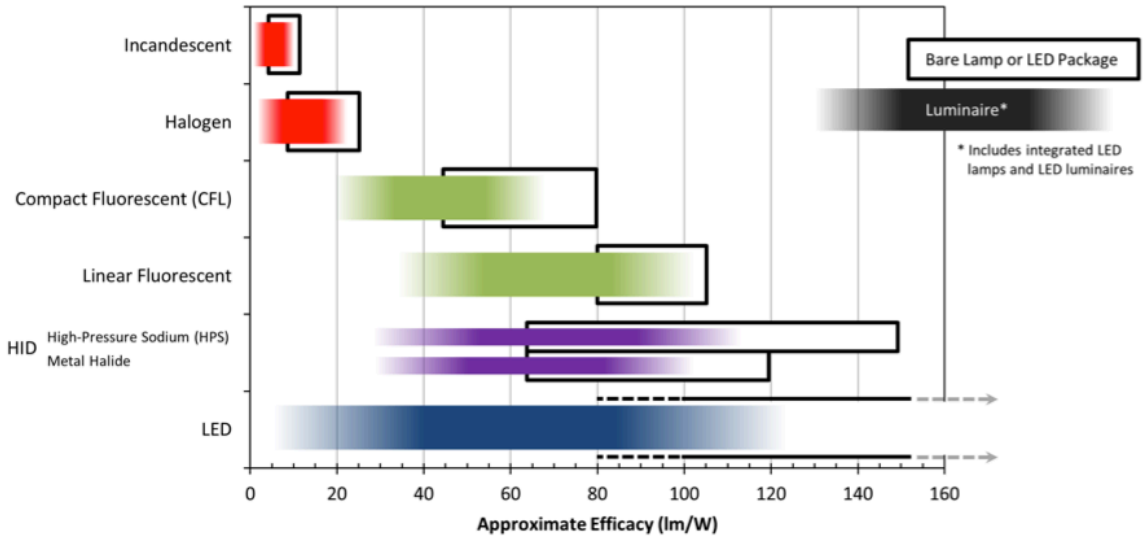

Figure 1.3: Approximate luminous efficacies for common lighting technologies (as of 2013), shown for the bare lamp / LED package, as well as when integrated into a luminaire – taking into account the entire system, including driver, thermal, and optical losses. Only LED technology is expected to make any substantial improvements in luminous efficacy in the near future [**7**].

Light Emitting Diode (LED) solid-state device technology has improved significantly in recent years. The invention of the blue LED in 1995 finally led to the ability to produce light that appeared roughly white, when used in conjunction with a phosphor optical coating. Further improvements in spectral coverage from this technology made for higher quality white light, with a range of colour temperatures (e.g. various degrees of "warm" and "cool" white). Increases in luminous efficacy, combined with better physical designs for junction heat dissipation – a consequence of the higher power requirements of high brightness LEDs – have enabled LEDs to become a new alternative for electric space and task lighting. LEDs are also highly directional, meaning less light is wasted through reflection and diffusion within the luminaire (much of the light being lost before it even leaves the enclosure) [**8**]. LEDs are able to be precisely and instantly dimmed using Pulse Width Modulation (PWM) because of their inherently high semiconductor switching speed.

The high efficiency, long lifetime, and directional nature of LED luminaires make them ideal for commercial applications. As of 2014, LED luminaires for indoor commercial applications had an average luminous efficacy of 86 lm/W, and reached over

130 lm/W on the high end [**7**]. LED luminaires now on the market have a MTTF of over 52,000 hours (~6 years of continuous operation), and are often quoted as having lifetimes up to 100,000 hours [**5**]. Failures most often derive from components other than the LEDs themselves, driver power supply circuitry being the least reliable, as illustrated in Figure 1.3 [**9**]. It is worth noting that LED luminaires typically employ arrays of many high brightness LEDs, wired in parallel in higher quality designs, such that an open circuit failure of one or more LEDs is almost unnoticeable. However, LEDs typically do not fail in this way if adequate heat sinking is used. As shown in Figure 1.5, as with all incumbent lighting technologies, LEDs lose brightness over their lifetime, but do it very gradually rather than burn out suddenly. This decrease in light output over time is known as lumen depreciation, and standard industry practice is to consider a drop to 70% of original output (L_{70}) as the point when a luminaire should be replaced. Depending on the application or market, however, even a 50% depreciation (L_{50}) can be considered acceptable [**9**].

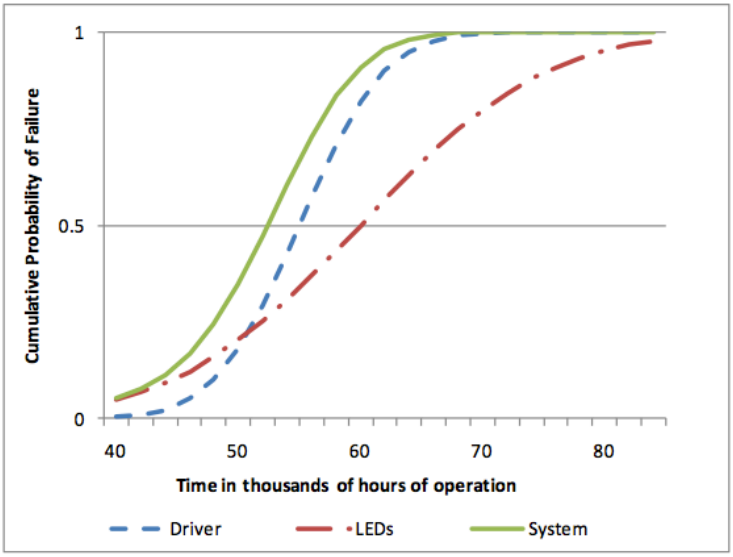

Figure 1.4: Cumulative probability of failure of LED luminaires. The MTTF for an LED luminaire is ~52,000 hours. Failure mechanisms related to the LED driver power supply and the LEDs themselves are singled out for comparison, indicating that most LED luminaires fail because of driver power supply failures rather than LEDs themselves (which tend to gradually dim rather than fail catastrophically, and there are typically many LEDs in one luminaire) [**9**].

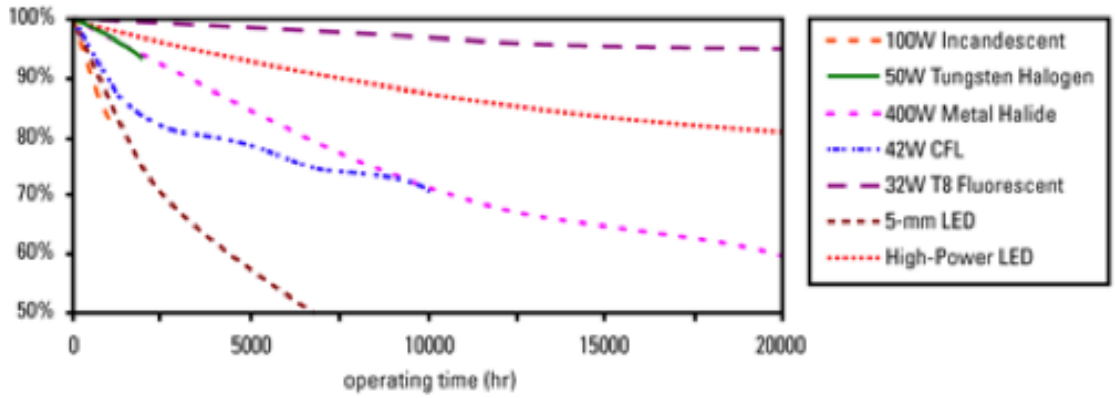

Figure 1.5: Typical lumen depreciation curves for various lighting technologies. High-power (high brightness) LEDs exhibit a very gradual decrease in luminous output over time versus all technologies with the exception of linear T8 fluorescents, however the MTTF of linear fluorescents is only on the order of 20,000 hours (the time horizon shown here), versus over 50,000 hours for LEDs [**9**].

LEDs are beginning to compete successfully with incumbent lighting technologies across a number of applications due to their ability to offer high quality and cost-effective performance. A number of recent market studies report that LED luminaires are becoming common in covered parking [**10**], as well as walkway and outdoor area lighting, street lights [**11**], and warehousing [**12**], and predict that current rapid improvement in LED performance coupled with falling prices will sharply increase the demand for LEDs for lighting applications across many sectors in coming years [**13**].

1.2.2 Building Automation Systems

Large commercial buildings built within the past 30 years commonly have a Building Automation System (BAS) installed, enabling centralized and coordinated control of building subsystems including the following: heating, ventilation, and air conditioning (HVAC), lighting, fire/gas alarms, public address / emergency sound systems, closed circuit television (CCTV) and security / access control, and transportation (elevators, escalators, and conveyor belts).

A subset of the BAS, and a term sometimes used interchangeably, is the Building Energy Management System (BEMS), the goal of which is to minimize energy use while maintaining or even improving occupant comfort. Energy efficiency has become of the utmost importance in new building design, and in renovation and retrofit of existing buildings. Building automation enables the holistic coordination of control and

performance monitoring necessary to make optimal use of energy in building subsystems, beyond what is possible by standalone local loop or manual control.

As illustrated in Figure 1.6 [**14**], BAS control in commercial buildings is generally implemented using a three-tiered building automation network. A supervisory controller at the management level – typically a desktop computer or network server – is connected through a wired network to the automation level nodes – local unit controllers associated with a zone or building subsystem. These in turn are connected through a wired field network to sensors and actuators that provide the physical interface with the subsystem or environment. Examples of sensors include occupancy, temperature, humidity, and ambient light sensors. Examples of actuators include heating duct valves and fans and lighting circuit relays / dimmers). Common wired building automation protocols [**15**], found mostly in large commercial buildings, include BACnet [**16**], Modbus [**17**] LonWorks / LonTalk [**18**] [**19**], and EIB/KNX [**20**].

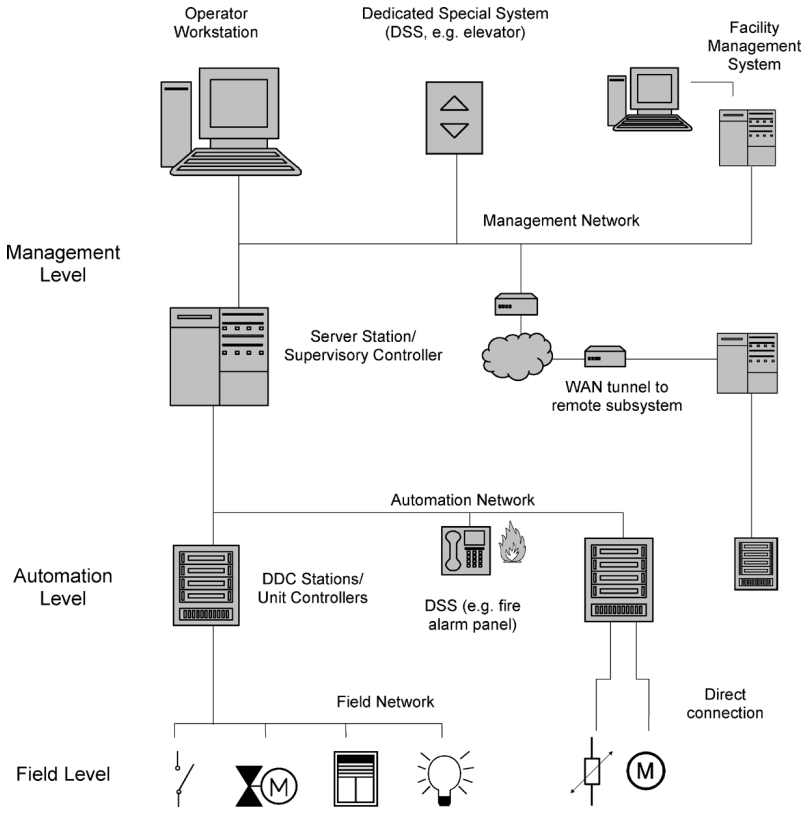

Figure 1.6: The typical three-tiered architecture of a Building Automation System [**14**].

1.2.3 Low-Power Wireless Networks

Although significant energy efficiency and user comfort benefits have come from BAS, wired BAS network installation can be very costly and disruptive, particularly in retrofit [**14**] [**15**]. Installing a BAS will often not be an option for small or medium sized commercial building owners or operators.

Recent advances in reliable low-power wireless networks, and falling costs in hardware manufacturing, promise advantages over traditional wired BAS architectures [**21**] [**22**] [**23**]. Installation costs and complexity are significantly reduced, and the high spatial and temporal granularity and flexibility promised by easily deployable low-cost wireless sensors and actuators will enable further advancement in BAS control strategies and algorithms for maximizing energy savings and occupant comfort [**24**] [**25**]. Some early low-power wireless network protocols that have been advocated for or used in BAS applications include ZigBee [**26**], JenNet [**27**] Z-Wave [**28**], EnOcean [**29**], Bluetooth [**30**], and KNX/RF [**20**], and a number of proprietary solutions. The past decade has seen a large increase in the use of low-power wireless networks in BAS, and projections for growth in market share are promising [**25**].

Together with gateway devices, low-power wireless networks provide two-way connectivity between uniquely identifiable sensors and actuators in the field and the BAS automation and management levels. Low-power wireless networks are sometimes referred to as the key enabler to the Internet of Things [**31**]. The Internet of Things (IoT) is a term coined [**32**] to describe an emerging paradigm in distributed systems.

The Internet of Things (IoT) is the interconnection of uniquely identifiable embedded computing devices within the existing Internet infrastructure. Typically, IoT is expected to offer advanced connectivity of devices, systems, and services that goes beyond machine-to-machine communications (M2M) and covers a variety of protocols, domains, and applications. [**33**]

Low-power wireless networks offer improved automation solutions in many areas besides commercial building automation that are considered part of the IoT domain,

including environmental monitoring, health care, industrial processes, intelligent traffic systems, smart homes, and the Smart Grid [**34**].

1.2.4 Single Board Computers

The past few years have seen a proliferation of the 'pocket size' Single Board Computer (SBC). Because of increasingly higher levels of hardware integration and falling manufacturing costs, these devices have enabled small companies and hobbyists with very limited budgets to develop projects that in the past would have required a full size desktop computer or laptop to handle the computational load and offer the range of operating system and I/O options often required. Coming from the other side, custom embedded solutions usually require more time-consuming and costly iterations on electronics design, parts selection and procurement, printed circuit board (PCB) fabrication, assembly, and test – and such solutions are not easily adapted for changing requirements. With their small form factors and low power requirements, SBCs are blurring the line between more general-purpose higher-level solutions and traditional function-specific embedded systems development. SBCs enable developers to take advantage of more feature-rich operating systems, higher-level programming languages, and modular hardware peripherals and I/O interfaces. A few currently available SBCs are potentially good fits for BAS applications at the automation level, as they offer adequate processing power and memory as well as LAN and WiFi connectivity options to implement middleware to connect, and offer services to, field level devices and management level computers. A number of SBCs currently on the market that can be suitably used in this role for prototyping include the Raspberry Pi [**35**], Arduino [**36**], and Beagleboard [**37**]. Commercial grade offerings exist, such as the eBox [**38**], and BAS-specific Advantech Energy Data Concentrator [**39**], and MangoES [**40**] SBC systems.

1.2.5 Cloud Computing and Web Applications

A large rise of the use of Internet Data Centers (IDCs) by the masses began at the end of the 1990s along with significant bandwidth improvements for most Internet users in the developed world [**41**]. Enterprise web applications began to emerge that were hosted on IDC infrastructure "in the cloud" – the metaphor referring to the obscuration of the computing and network resources as seen by the user – and in 2002 Amazon Web

Services (AWS) [**42**] emerged offering a set of cloud-based services including computation and data storage. In 2006 AWS began offering its Elastic Compute Cloud (EC2) [**43**], where users (usually service provider companies) could rent the use of virtual machine (VM) instances with a selection of computing resource options (i.e. different CPU speed and RAM combinations), with access to and control of most of the software stack as far down as the kernel [**44**]. Periodic automatic backup and VM instance snapshot services were also made available to support high data integrity and scaling. Higher-level cloud resource offerings came from other companies, such as Google's App Engine [**45**] which allows users to construct traditional web applications with a set of tools and constraints on application structure that enforce separation of stateless computation and stateful storage [**44**]. These constraints make scaling, availability, and data integrity for these applications as essentially built-in features.

Depending on the nature of the use case and application, service providers thus have options ranging from full configurability of cloud VM software stacks to rapid build-up application toolkits. These offerings from infrastructure providers have come to be known as Platform-as-a-Service (PaaS). Cloud computing resources have thus come to be viewed as a utility, and have some key advantages for companies using this infrastructure to provide Software-as-a-Service (SaaS), versus the traditional approach of running web applications on private data center infrastructure. These include the following:

- the ability to scale computing resources up and down according to time demand of services, versus paying for and building out private infrastructure for peak load and underutilizing these resources for the vast majority of time,
- continuously maintained computing infrastructure by a 3rd party (no need to monitor, upgrade, or replace aging assets),
- optional redundancy in computational and storage nodes to improve reliability and availability, and

- organizations that do large batch processing can leverage the "cost associativity" [**44**] [**45**] of cloud computing to finish jobs faster (i.e. using one cloud VM for 100 hours is equivalent to using 100 VMs for 1 hour).

These advantages are being realised in building automation, with the Management level increasingly moving into the cloud [**15**] [**14**].

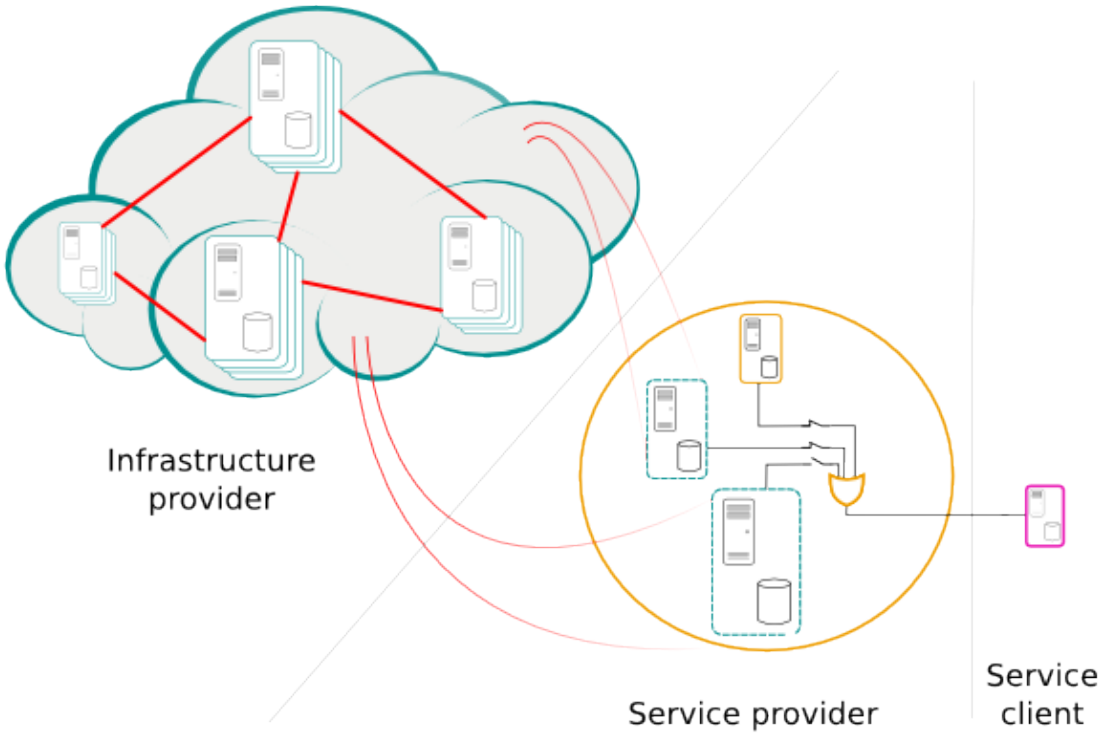

Figure 1.7: A common way to view the abstraction of cloud computing – the infrastructure provider represents the Internet Data Center (IDC) and virtual machine resources on which a service provider can run a standard or custom software stack to provide web-based services to clients. The actual physical computing and network resources of the infrastructure provider are obscured (and generally unimportant) to the service provider and service clients, thus the "cloud" metaphor (Davide Lamanna, 2011).

2 Overview of the Intelligent Lighting System

The Intelligent Lighting System is a solution to one subsystem of building automation: lighting. It consists of four distinct component types, spanning the three traditional levels of the Building Automation System hierarchy. It was designed with the possibility in mind of acting as a BAS backbone to support other building subsystems in the future as well, such as HVAC and security. The high-level architecture we developed for the ILS is shown in Figure 2.1.

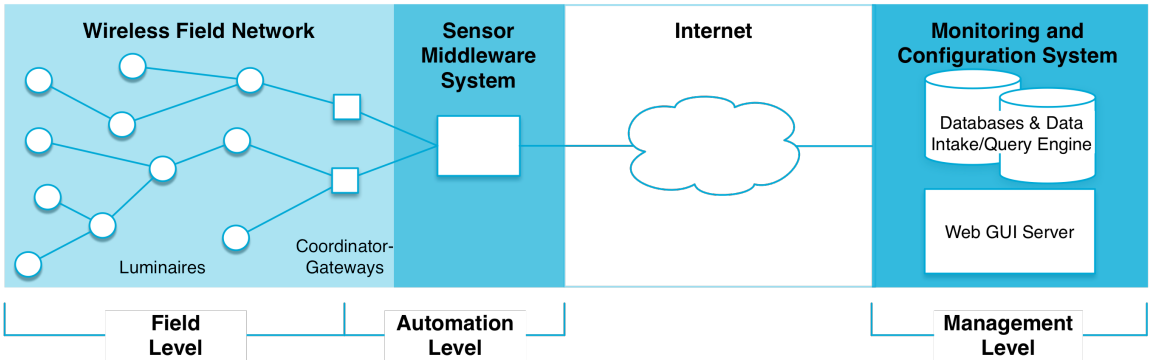

Figure 2.1: High-level architectural diagram of the Intelligent Lighting System (ILS). At the BAS field level the Wireless Field Network (WFN) consists of luminaires, each with an embedded System-on-Chip with wireless radio, that are coordinated in control actions and data collection by Coordinator-Gateway (CG) devices at the top of each WFN. The automation level is shared by CGs and the Sensor Middleware System (SMS), implementing lighting zone coordination, scheduled operations, sensor data reporting, configuration update command handling, network health monitoring, and firmware update functionalities. The management level is located offsite in the cloud (Internet Data Center infrastructure), implemented with virtual machines hosting monitoring and configuration databases and associated data intake and query engines, and web-facing Graphical User Interfaces.

At the field level, the Wireless Field Network (WFN) of the ILS is a two-way sensor and actuator network of individually addressable luminaires, with Coordinator-Gateway (CG) nodes responsible for wireless network membership management, control coordination, and sensor data collection. The automation level functionalities are shared by the CGs and the Sensor Middleware System (SMS), implementing lighting zone coordination, scheduled operations, sensor data reporting, configuration update command handling, network health monitoring, and firmware update functionalities. The SMS acts as a message broker and protocol translator, and also offers field network supervisory services including wireless network health monitor, remote software update manager, and troubleshooting interface host. The management level is handled by the Monitoring and Configuration System (MCS), and is implemented on Internet Data Center infrastructure,

distributed across a number of virtual machines. Data arriving continuously from the field is archived into a time series database via a data intake subsystem, and a web-based Graphical User Interface (GUI) provides users with the capability to remotely configure the operation parameters of intelligent lighting installations as well as query and visualize various ILS performance metrics.

An overview of the design of these ILS subsystems will be given in the remainder of this chapter, and in-depth coverage of the design and implementation of the SMS will be given in Chapter 3.

While spanning the layers of traditional BAS, the ILS is a distributed system solution that fits the new Internet of Things (IoT) paradigm, versus the Machine-to-Machine (M2M) paradigm that traditional BAS largely fits into. IoT differs from M2M in a few key ways [**46**]:

- IoT solutions use IP-based networks to interface field devices to middleware and/or cloud layers; M2M solutions use point-to-point communications with wired or cellular wireless networks, a much less scalable scheme.

- IoT solutions rely more on software, whereas M2M solutions rely more on embedded hardware. This makes IoT solutions more flexible for providing access to a wide variety of internal and external customers, in any location with Internet connectivity (desktop or mobile).

- IoT solutions emphasize the ability to centrally aggregate and visualize data, perform “big data” analysis, and offer enterprise applications leveraging this analysis to provide improved service and/or business performance and innovation; M2M solutions typically target single point service management applications, with data not usually integrated with enterprise applications.

2.1 Application Domain

The application domain of commercial indoor / covered parking lighting can be generally characterized as follows:

- ceiling heights can vary from between ~ 2 m to ~ 6 m, but average ~ 2.1 m,
- choices of different luminaire form factors is limited,
- horizontal piping and ducting are common obstacles to luminaire installation,
- vertical pillars are common obstacles to direct illumination,
- luminaire and sensor hardware and cabling must be tolerant to temperature and humidity ranges for the climate, but do not have to be weatherproof (i.e. no direct exposure to water or solar radiation),
- floors are concrete and are often polished (reflective),
- occupant traffic can be both human and vehicle,
- occupants can enter and exit through potentially many places (e.g. vehicle ramps, elevator lobbies, and stairwell entrance doors),
- there is typically little or no daylight infiltration, thus lighting is crucial to occupant safety,
- occupants tend to transit through indoor parking areas, for short durations generally less than 5 minutes and concentrated into high traffic periods (i.e. morning arrival, lunch break, evening departure), and
- quality of service (QoS) expectations are lower than continuously occupied spaces (i.e. temporary service interruption, system bugs, and engineers/technicians working on-site are generally more tolerable and less disruptive to occupants).

The commercial building that hosted the demonstration and pilot installations of the ILS contained a spiral ramp multi-level underground parking area. The existing lighting system consisted of Metal Halide High Intensity Discharge (MH HID) lamps, each drawing 140 Watts of power. Figure 2.2 gives a photograph of the original lighting installation. The very limited spectral coverage of MH HID lighting is evident by the orange hue and poor colour rendering of the vehicles in the image, as well as contrast between directly lit and shaded areas.

Figure 2.2: A photograph of the multi-level underground parking area used in the demonstration and pilot scale installations of the ILS. A section of the original lighting installation is shown, in total consisting of 89 x 140 Watt MH HID luminaires. The very limited spectral coverage of MH lighting is evident by the orange hue and poor colour rendering of the vehicles in the image, as well as poor contrast between directly lit and shaded areas (these 89 MH HID luminaires were replaced by 52 white LED luminaires at the pilot stage).

2.2 System Requirements

The ILS was envisaged as a distributed system as shown in Figure 2.1 in order to combine the unique advantages of each of the enabling technologies into a significantly improved commercial lighting solution, both from an energy efficiency and a user experience standpoint. Its web-based GUI was to serve a variety of user classes, initially including authorized ILS administrators and engineers, and customer facility managers. To promise an acceptable level of overall distributed system reliability, loose coupling was to be employed. ILS subsystem faults should be handled gracefully, and data loss should be minimized. Availability of web-based user services should be good, but it is not paramount nor critical to the on-the-ground responsiveness or overall quality of illumination provided by the luminaires. The degree to which each of the above general requirements – reliability, availability, fault handling, and data integrity – can be met is

always subject to cost trade-offs, and specific numeric targets were not specified for these during the stages of ILS research and development described in this thesis. The list of functional and feature requirements for the ILS are summarized in Table 2.1.

Req #	**Short Name**	**Description**
1	Addressability	Luminaires shall be individually addressable.
2	Network connectivity	Luminaires shall be interconnected within a field network. The 2.4GHz radio band – allotted for use cases that include building automation systems – has been chosen as the preferred medium for the Wireless Field Network.
3	Remote monitoring and configuration	A bridge from the Wireless Field Network to the Internet shall be implemented, and remote monitoring and configuration of the facility side of the ILS shall be provided to system administrator, engineer, and customer users.
4	Dimming	Luminaires shall be dimmable through an output intensity range of 10 – 100%.
5	Dual light dimming levels	During normal operation, a luminaire's light output level will be confined to two possible values – ocs*Low* and ocs*High* – which will be adjustable by system operators.
6	Motion sensing	Motion sensors shall be used to detect occupants (humans and vehicles) and trigger luminaire dimming changes from *ocsLow* to *ocsHigh* output.
7	Responsiveness / Low Latency	Luminaire dimming change actions triggered on motion sensing events should take 1 second maximum to be carried out.
8	Luminaire Behavioural Operating Policies	Upon a motion sensing event, a luminaire's light output level will transition to a specified *ocsHigh* level and remain there for a predefined period *ocsDelay*, after which time it will transition to a specified *ocsLow* output level (in the absence of additional motion sensing events within the *ocsDelay* period, which would have the effect of resetting the timer). A combination of *ocsLow*, *ocsHigh*, and *ocsDelay* parameters constitutes an *Operating Policy*, which will be remotely adjustable by authorized system operators.
9	Operating Policy Scheduling	Luminaire *Operating Policies* shall be assignable to *Schedules*. A single *Operating Policy* may be active at any given time on a luminaire; the *Schedule* defines the period during which the *Operating Policy* is active (identical *Operating Policies* may be assigned to different *Schedules*). *Schedules* may not overlap temporally.
10	Luminaire Zoning	Luminaires shall be able to be assigned into logical groups called *Zones* by system operators. Luminaires assigned to the same *Zone* will share *Schedules* and associated *Operating Policies*. Luminaires may not be members of more than one *Zone* simultaneously. Luminaires are to be assigned into *Zones* remotely by authorized system users.
11	Telemetry Monitoring	A central database (or multiple databases) will maintain up-to-date status data for luminaires for system health monitoring and troubleshooting, and historical time series data at 5-minute temporal granularity for performance visualization and analysis, including energy use and motion sensor event time series.
12	Graphical User Interface	A GUI will enable easy remote configuration of the ILS *Zones*, *Policies*, and *Schedules*, and firmware update functionalities. It will also provide graphical ILS performance reporting and time series data visualization / plotting functionalities, and system warning/error condition notifications.
13	Spatial Model	The system shall contain a 2D spatial model of the interior building layout (at least for the areas where the intelligent lighting system is installed)

		with knowledge of the physical location of each installed luminaire.
14	Reliability, Fault Tolerance, and Recovery	The system shall maintain a high degree of reliability, with failover modes implemented to ensure occupant safety, no data loss, and quick recovery to normal operation.
15	Remote Firmware Update	The firmware/software running on embedded field devices shall be remotely updateable.
16	Security	The ILS shall employ adequate and up-to-date web and network security technologies to guard against a range of known techniques that might allow an unauthorized user to disable, damage, or gain control of the system, or eavesdrop or intercept data transmissions.
17	Easy, quick, and repeatable system instance build-up	New instances of the ILS shall be easy and quick for system administrators, engineers, and installers to build-up and commission, in a standardized and repeatable fashion.
18	Scalability	The ILS shall be scalable using multiples of the same architectural component types, as needed for the size and/or fault-tolerance requirements of the application.

Table 2.1: The functional and feature requirements of the Intelligent Lighting System.

2.3 System Domain

The UML domain model given in Figure 2.3 represents the conceptual classes of the ILS, showing their attributes that are key to tying together this distributed messaging-based system, as well as those that define the behavioural parameters of the luminaires. Emphasis is given to entities of the ILS physically residing within the commercial building facility (i.e. the MCS is represented as a single entity instead of showing detail of the distributed virtual machine nodes that comprise it; incoming Internet traffic is handled and distributed by a load balancer). The *Facility* class captures the basic uniquely identifiable information about the facility (metadata such as customer account information are left out here). The *LANRouter-InternetGateway* class represents the combined gateway and router (generally combined into one piece of hardware) used to connect the facility to the Internet. It is assigned a public IP address by the facility's Internet Service Provider, or existing network administrator if using the customer's existing LAN infrastructure, and itself hosts a private ILS LAN. There can be redundant instances of *LANRouter-InternetGateway* for higher network reliability. The Sensor Middleware System is one member of this ILS LAN and presents three listening ports, one for TCP traffic from all WFNs within the facility, one for HTTP traffic from the MCS, and one for HTTP traffic on the SMS Administration Web Application it hosts. Each CG hardware device manages a WFN and provides a gateway to the ILS LAN, and

is represented by the *Coordinator-Gateway* class. It listens on one TCP port for messages from the SMS. The *Node* class represents all entities that can be members of a WFN, which at this point includes the WFN's single *Coordinator-Gateway* and all luminaires (up to a limit of ~100), represented by the subclass *Luminaire*. Each *Node* has a unique address within its WFN, and the *Coordinator-Gateway* is always at address 0. The *Zone* class represents (optional) logical groupings of luminaires within a WFN, generally spatially clustered in reality to form a lighting zone, wherein the event-based lighting behaviour of all group members is coordinated in real-time by the CG. The event-based lighting behavioural parameters of a luminaire – the *ocsConfig*, *ocsLow*, *ocsHigh*, and *ocsDelay*[2] – are conceptualized as an *OperatingPolicy*. Scheduled changes in these parameters are handled by the CG at the *Zone* level by way of sending identical parameter update commands to all member luminaires. The *Schedule* conceptual class represents the Unix Cron string[3] that defines the times when the *OperatingPolicy* for each *Luminaire* within a *Zone* is to be changed, and each instance of *Schedule* is given a unique identifier and user-entered name. Through the MCS GUI many possible *Schedule* instances can be assigned to a given *Zone*; logic in the MCS prevents assigning temporally overlapping *Schedules* to a *Zone*. Lastly, the conceptual classes *PassiveInfraredMotionSensor* and *MicrowaveMotionSensor* represent integrated Passive Infrared (PIR) and optional (external) active microwave Doppler effect motion sensors. The toggling states of these sensors are the triggers of event-based luminaire behaviour.

[2] The *ocs* prefix is an abbreviation for "occupancy sensing" and is used to indicate the relatedness of the *Low*, *High*, *Delay*, and *Config* parameters.

[3] A Cron string is a 5-digit code scheme that allows for very flexible periodic or one-shot scheduling of tasks, originally implemented on the Unix operating system.

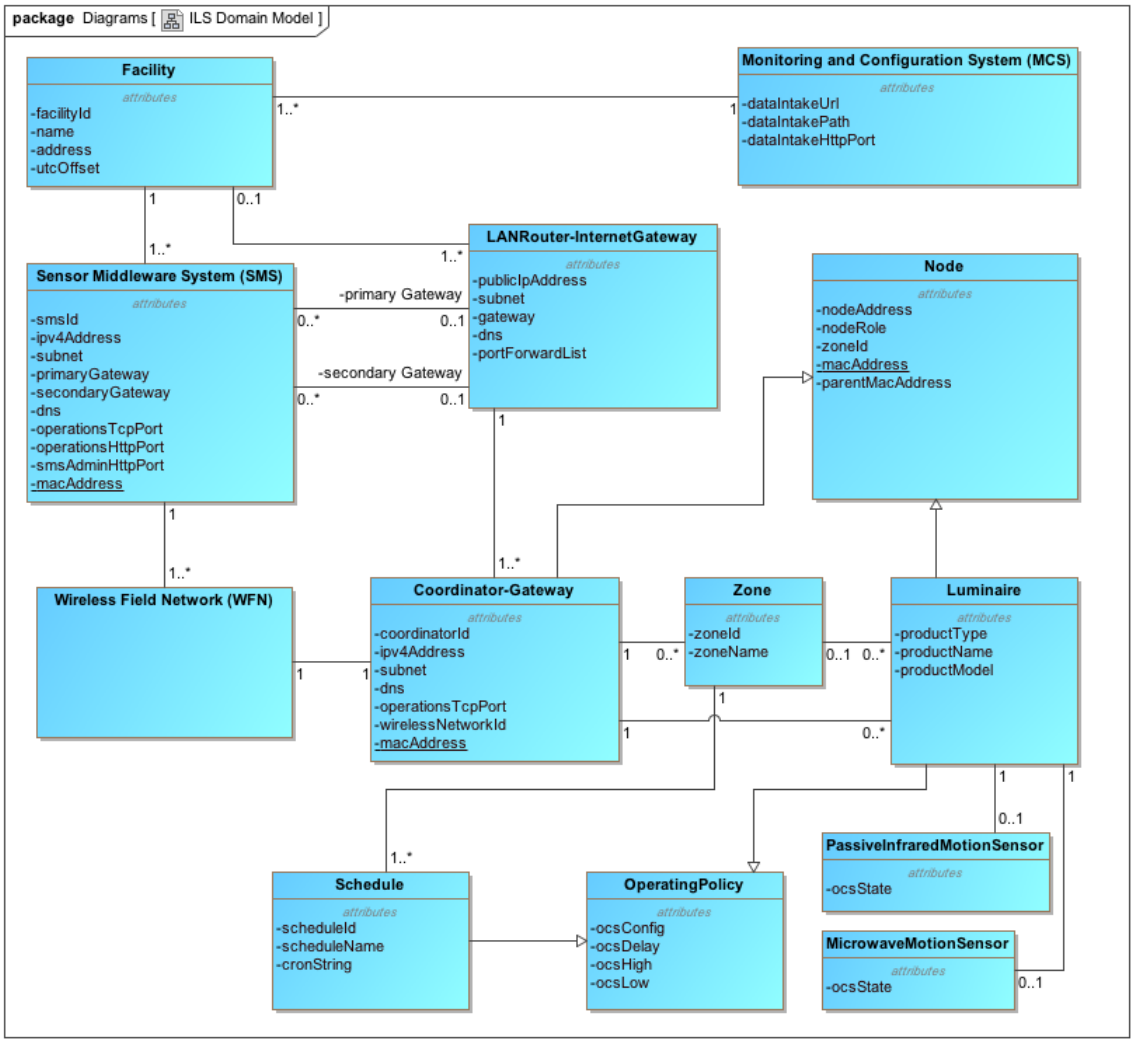

Figure 2.3: UML conceptual class diagram of the ILS domain model, showing the attributes key to tying together this distributed messaging-based system, as well as those that define the behavioural parameters of the installed luminaires. Emphasis is given in this diagram to entities of the ILS physically residing within the commercial building facility.

2.4 Wireless Field Network (WFN)

The field network is the lowest level of the Building Automation System architectural hierarchy, consisting of field actuator and sensor components and the embedded electronic and processing components that provide real-time actuator control, sensor monitoring, and field networking capabilities. The following subsections will give an overview of the Wireless Field Network (WFN) developed for the ILS.

2.4.1 Wireless Networking Overview

The wireless technology chosen for the ILS WFN is called JenNet, developed by Jennic, Inc. [**27**]. It is based upon the IEEE 802.15.4 standard [**47**], supporting a star or

cluster tree topology (Figure 2.4) that allows for very extended multi-hop network formation and maintenance with straightforward data packet routing (given that there can only be one possible route between any two nodes on the network)[4]. Figure 2.6 provides a conceptual illustration of the 802.15.4 protocol stack, where the ILS-specific application is implemented in C code at the Application Layer, and the proprietary implementations (e.g. JenNet) are precompiled at the application support layer. The Media Access Control (MAC) layer is responsible for services of device association / disassociation with the network, access control to shared channels, beacon generation (if applicable), and guaranteed timeslot management (if applicable) [**48**]. The physical layer handles the interface to the physical transmission media – in this case the 2.4 GHz radio band – and is responsible for channel assessment, bit-level modulation/demodulation, and packet synchronization.

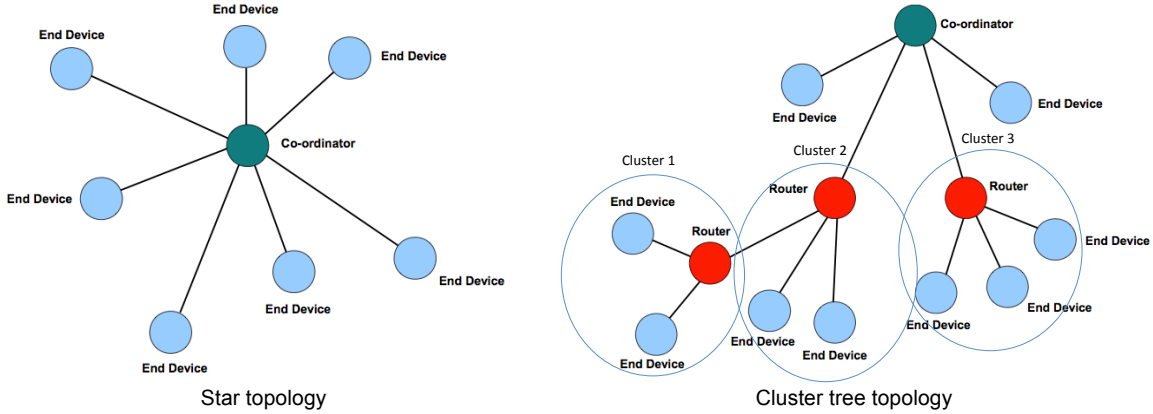

Figure 2.4: Star and cluster tree wireless network topologies (supported by JenNet [**48**] and ZigBee [**26**]). Data packet routing is simple as only one possible route exists between any two nodes. Nodes in the hierarchy are referred to as *parent* or *child* nodes, depending on their relative position to other nodes. In the cluster tree topology, Router nodes act as parents within a given cluster, typically serving as data and collection / command distribution points for the End Device children. Note that there theoretically may be many levels of Router nodes, and End Devices are optional.

[4] This simplicity, and no required licencing or royalty fees were factors in selecting JenNet [**27**] for use in the ILS WFN over ZigBee [**26**], another popular 802.15.4-based wireless network which uses a mesh topology, as depicted in Figure 2.5.

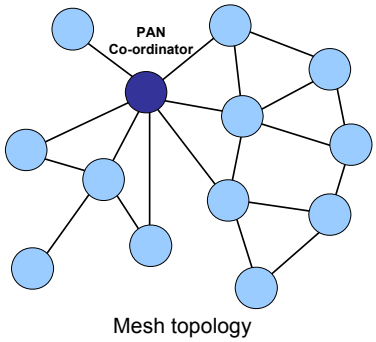

Figure 2.5: Mesh wireless network topology (supported by ZigBee [**26**]). Data packet routing is complex and limitations exist on the number of hops possible between any two devices.

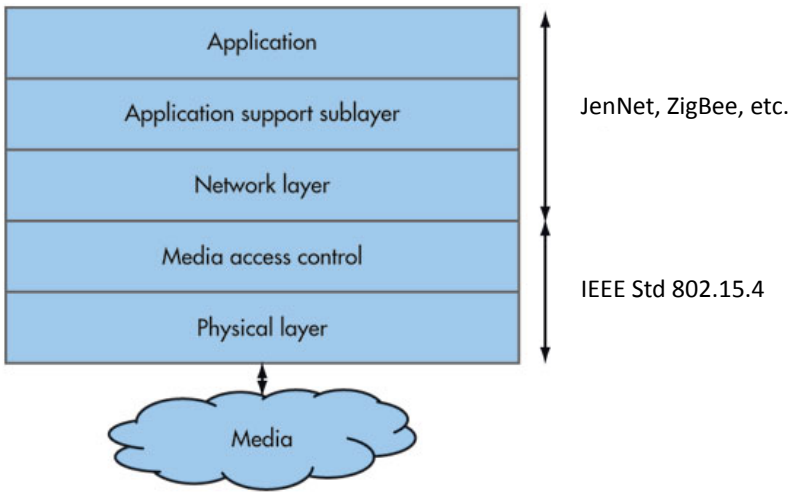

Figure 2.6: The protocol stack layers for the IEEE 802.15.4 standard. Only the bottom two layers are common among JenNet, ZigBee, and other such proprietary networks, leaving considerable flexibility for 3rd parties to create customized wireless network topologies and many other features [**26**].

The ILS WFN implementation employs JenNet in a cluster tree topology, consisting of a JenNet coordinator node at the top and router nodes implemented within the luminaires to string together an extended lighting network[5]. Both the coordinator and router nodes contain an embedded NXP JN5148 32-bit System-on-Chip (SoC) [**48**] for wireless network connectivity. A block diagram of the NXP JN5148 SoC architecture is given in Figure 2.7.

[5] No end devices were implemented at the time of writing, but the intent was to eventually use them for standalone motion or ambient light sensor nodes.

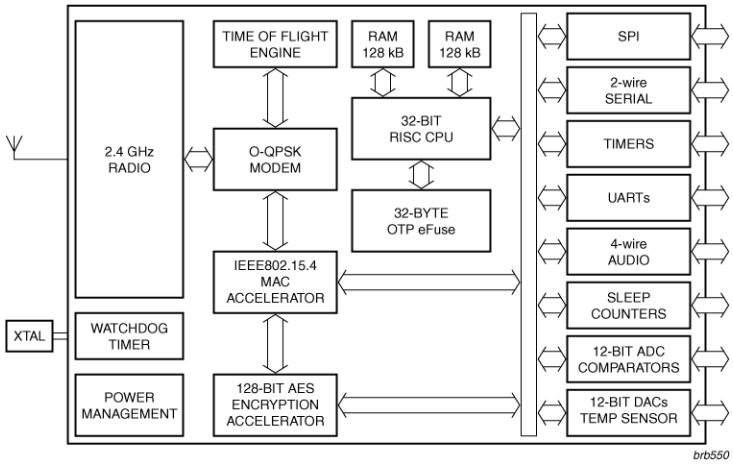

Figure 2.7: A block diagram of the NXP JN5148 System-on-Chip used to implement JenNet [**48**].

2.4.2 Coordinator-Gateway

The embedded Coordinator-Gateway (CG) device serves the dual purpose of WFN coordinator, as well as the communication gateway between the WFN and the TCP/IP LAN. It is installed in a location where an Ethernet connection is available and the physical unit has a decent radio line-of-sight to at least one luminaire among the group it is to manage as a single network. There may be many CG devices present and connected to the same LAN in an ILS installation, each hosting a distinct and separate wireless network (and JenNet allows for multiple networks to work in close coexistence). Responsibilities of the CG in managing the JenNet wireless network include authorizing node membership, enabling and repairing node connectivity, relaying messages and data between the luminaire nodes and the remote Monitoring and Configuration System (MCS), and coordinating control actions and reporting for luminaires grouped into zones.

Each CG has one NXP[6] JN5148 System-on-Chip (SoC) for coordinating one JenNet wireless network [**27**] [**48**] and running a modified version of the open source Jennic Intelligent Lighting Reference Design firmware [**49**] at the application layer. The reference design firmware was modified to introduce new functional behaviours, an expanded wireless node command set, and additional sensor types and reporting schemes. An ARM Cortex M3 processor [**50**] with an integrated Ethernet interface is included for LAN connectivity, implementing the gateway functionality using an open source TCP/IP stack. The CG is powered by 5 VDC via Power-over-Ethernet and hosts a USB interface

[6] Jennic, Inc. was acquired by NXP, Inc. in 2010.

for debugging or reprogramming the JN5148, and a microSD dock to host external Flash memory cards for manual loading of firmware images and diagnostic data logging for the Cortex M3 processor.

2.4.3 LED Luminaires

The luminaires of the ILS consist of a combination of a 3rd party high brightness LED luminaire designed for covered parking areas [**51**] (powered by a wired 120VAC or 347VAC supply), and additional integrated electronics and embedded systems. These luminaires serve as ideal wireless node platforms, being that they are located on the ceiling where line-of-sight to neighbouring wireless nodes is generally good. Additionally, they are provided constant AC power, which allows for higher power and more frequent radio operation and thus longer range and potentially better cluster tree stability [**27**].

The conceptual design of the luminaire is shown in Figure 2.8. A DC Power Supply converts AC line voltage to 24 VDC for driving the HB-LED arrays, and 5 VDC is supplied for the control electronics and sensors. Passive Infrared (PIR) motion sensors are integrated into each unit[7], and external long-range active Doppler effect microwave motion sensors can be attached by cable. A 32-bit NXP JN5148 System-on-Chip (SoC) provides wireless network connectivity and runs a modified version of the open source Jennic Intelligent Lighting Reference Design firmware [**49**] [**51**], similar to that running on the CG, at the application layer. Additions to the Application Programming Interface (API) provide set/get functionalities for remote configuration of lighting control behaviour, network behaviour, and data and sensor event reporting logic and timing (both event-based and periodic messaging) of the luminaire. The JN5148 SoC resides in an externally-docked plastic enclosure, this packaged unit being termed the Wireless Control Unit (WCU). This form factor was chosen to avoid potential radio shielding effects of the luminaire chassis and accommodate easy WCU swap replacement for problem units or hardware updates.

[7] An ambient light sensor is also shown, which was not included in ILS development for the underground parking garage use case.

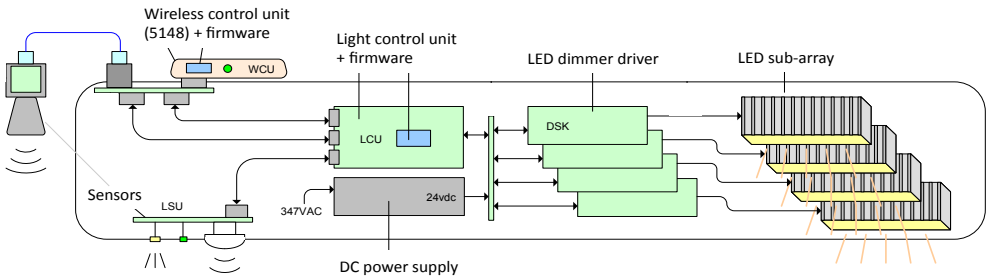

Figure 2.8: Conceptual design of the luminaire. Complete telemetry reporting and redundancy in control functionalities and critical lighting components, as well as dynamic fault recovery, make the luminaire highly reliable and fail-safe (H. Davis 2012).

An 8-bit Microchip PIC18F25K20 [**52**] microcontroller embedded within the luminaire itself is employed as a Light Control Unit (LCU). The LCU is responsible for controlling a Pulse Width Modulation (PWM) dimming signal for four high brightness LED sub-arrays of the luminaire via four LED dimmer drivers. Changes in *Operating Policy* – the combination of the *ocsLow*, *ocsHigh*, and *ocsDelay* values – are issued from the CG as a function of a *Schedule* becoming active, and are passed to the LCU from the WCU. Internal and externally connected motion sensors are monitored by the LCU to execute appropriate event-based lighting control actions with minimal latency based upon the currently loaded *Operating Policy*. The Boolean *ocsConfig* parameter of a luminaire determines whether it will operate in concert with others within its *Zone* (zonal mode), or independently (standalone mode) according to its currently assigned values for *ocsLow, ocsHigh,* and *ocsDelay* (if no *Zone* is assigned, standalone mode is the default). Zonal luminaire lighting action coordination (i.e. toggling between *ocsLow* and *ocsHigh* output levels) is conducted by the CG, essentially making the luminaire operate as a slave to the CG while in this mode. When in zonal mode, a luminaire does not transition to *ocsLow* output if any other luminaires in its zone still have occupied states (tracked by the CG). The CG will issue a message to all luminaires in a zone when all motion sensor states have transitioned to unoccupied (i.e. their *ocsDelay* periods have all expired without being reset by subsequent motion detections). This event-based lighting behaviour of a luminaire can be modeled by the state machine in Figure 2.9.

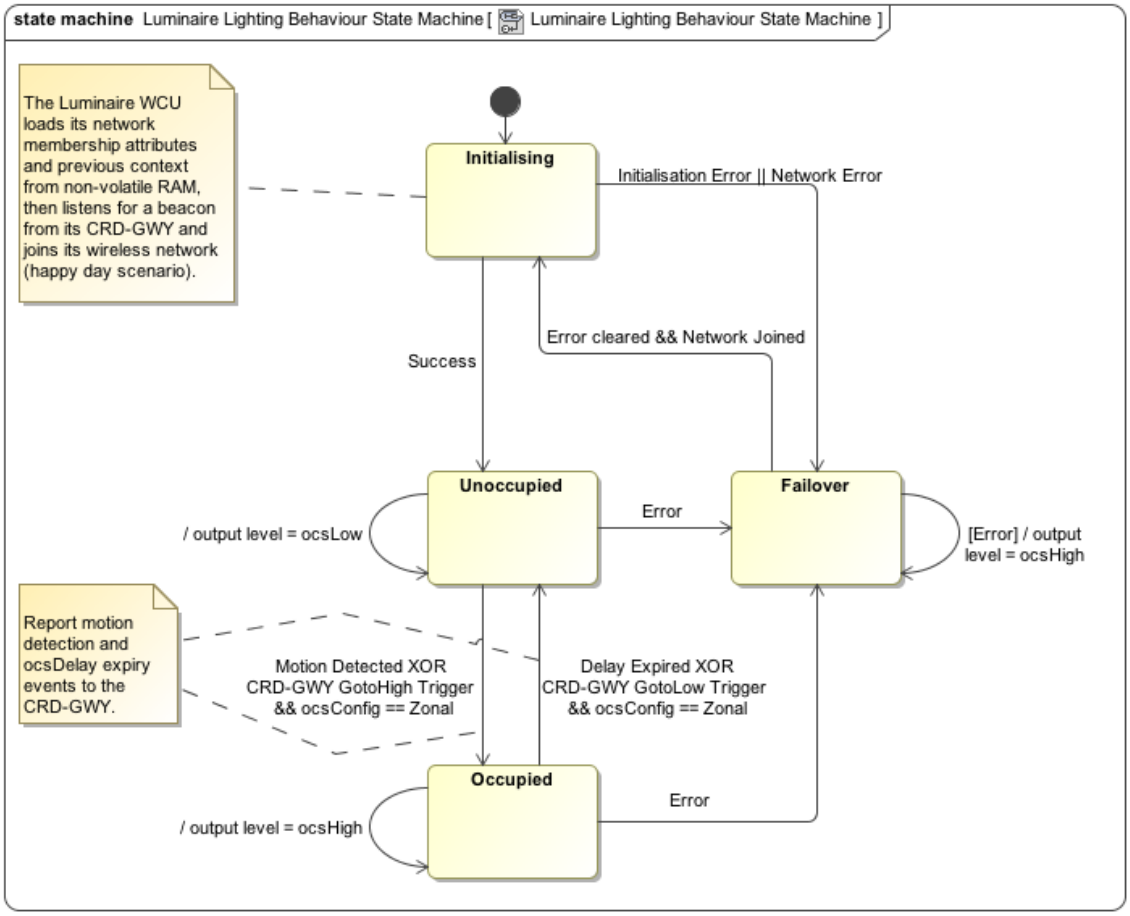

Figure 2.9: State machine of the event-based behaviour of the luminaire. Periodic reporting by the luminaire to the CG, and network management activities (e.g. connection / re-connection sequences) are not depicted here.

For example, a motion detection event would result in the LCU adjusting the light output from the luminaire's *ocsLow* policy level (e.g. 10%) to its *ocsHigh* policy level (e.g. 75%), holding at this level for its *ocsDelay* period (e.g. 5 minutes) and then returning to the default *ocsLow* level (given that no additional motion detections occurred during this period, which would result in resetting the delay timer). Luminaire control actions can also be triggered by the WCU in one of two ways: 1) as a result of receiving a trigger from the CG (when a control action is being coordinated among luminaires within a zone), or 2) when transitioning in or out of failover modes (a luminaire will remain at its *ocsHigh* output level for the duration of any error conditions).

A number of fault tolerance features have been designed into the WFN hardware and firmware. These include wireless network coexistence, security, and fault recovery features built into the JenNet stack that will not be covered here (for more information

see [**27**]), and some customized to our lighting application. Fault tolerance features at the application layer of the luminaire embedded software include the following:

- In the event that a luminaire loses its wireless connectivity, or the WCU itself fails or becomes unresponsive, the LCU will revert to a fail-safe standalone mode driving the luminaire LEDs at the *ocsHigh* output level.

- The WCU employs a watchdog timer in the JN5148 to self-reset if has gone into an unresponsive state, and reloads the previous JenNet network and lighting application contexts from non-volatile memory during initialisation to attempt to quickly rejoin its assigned wireless network and return to normal operation.

Throughout normal operation, luminaires send a combination of asynchronous messages from the application layer to the CG that can be event-based or regularly periodic. Event-based messages pertain to motion detection events. Periodic messages include reports of luminaire state (including instantaneous power and luminaire output levels) and settings, accumulated energy usage over the time period, running firmware version and new firmware download status, and network connectivity heartbeats. Many of these messages are passed on by the CG via the LAN to the Sensor Middleware System (SMS) for handling. Message descriptions and SMS handling of these messages will be covered in Chapter 3.

2.5 Monitoring and Configuration System (MCS)

The Monitoring and Configuration System (MCS) was implemented on Amazon Web Services (AWS) Internet Elastic Compute Cloud (EC2) [**43**] Internet Data Center (IDC) infrastructure. AWS EC2 was chosen to host the MCS because of high reliability and availability guarantees, and optional storage redundancy and automatic backup services. EC2 instances come with user-friendly web-based management consoles and an API that makes EC2 instances easy to create, configure, replicate and scale up (e.g. converting to larger instances with greater computing resources and/or formations of cooperating and/or redundant EC2 nodes).

The MCS was developed in two phases. A prototype MCS (pMCS) proved a number of intelligent lighting monitoring and configuration functionalities, and was the technological precursor to the combined solution of the enterprise MCS and the SMS. The pMCS demonstrated two main capabilities:

- receiving and archiving of ILS facility WFN status and performance data using IDC infrastructure, and
- hosting web-based Graphical User Interfaces (GUIs) on IDC infrastructure to provide visual reporting and basic analytics of ILS performance from a desktop or mobile web browser

The pMCS would poll the CG at the top of a WFN directly over the Internet (via facility LAN infrastructure) every minute for status, energy use, and occupancy state reports for all luminaires, then store the telemetry data within the responses in a time series database. It provided a basic user-facing reporting GUI web application [**53**] with auto-generated summary reports and search and plot functionalities for energy use and motion sensing data, and an administrator-level advanced time series query and plotting engine [**54**] with access to all telemetry data. The pMCS was tested and proved at the demonstration scale, with a WFN consisting of one CG and four luminaires (a performance summary of the demonstration scale deployment, and screenshots from the pMCS GUI, are given in Chapter 4).

The next phase of development for the MCS was to implement an enterprise-level commercial solution, feature-rich and scalable as needed so as to be capable of handling a virtually limitless number of facilities, each with many thousands of luminaires. Additional features and improvements to be developed for the MCS included the following:

- Easy creation and configuration of new MCS AWS EC2 virtual machine instances, scalable as needed for additional/larger ILS installations. Improve MCS reliability through optional component redundancy, network load

balancing, and automatic database backup / mirroring (ILS requirements in Table 2.1, #17, 18 and 14 respectively).

- Remote configuration of ILS facility subsystems via the MCS. Features made available to authorized users included assigning luminaires to zones, assigning zone-specific lighting schedules, and providing remote firmware updates for WFN nodes (Table 2.1, #10, 9, and 15).

- A spatial model of facility floor plans and a graphical map-based depiction in the GUI of the physical locations of luminaires. This would allow authorized users to easily find and interact with luminaire attributes and time series data, adjust settings for luminaires (and zones of luminaires) and receive error condition notifications in an intuitive manner (Table 2.1, #13 and 12).

- Monitoring a larger set of WFN node state attributes and events, including catching communication state (network connectivity) dropouts and generating graphical alerts for the ILS Administrator (Table 2.1 #14).

- Improving overall visual presentation and interactive capabilities in the web-based GUI, and creating additional support for use on mobile smart phones and tablet devices.

Similar to the pMCS, the enterprise MCS implemented two databases, one for storing quasi-static luminaire attribute values, and another for storing and retrieving potentially large volumes of continuously arriving time series data. Figures 2.10, 2.11, 2.12, and 2.13 show screenshots of some of the views provided by the MCS GUI, illustrating the implementation of some of the above features.

Figure 2.10: A screenshot of the summary view of the web-based MCS GUI. The view defaults to present the past 7 days of daily ILS energy use as well as the past 24 hours of hourly ILS energy use activity for a Facility.

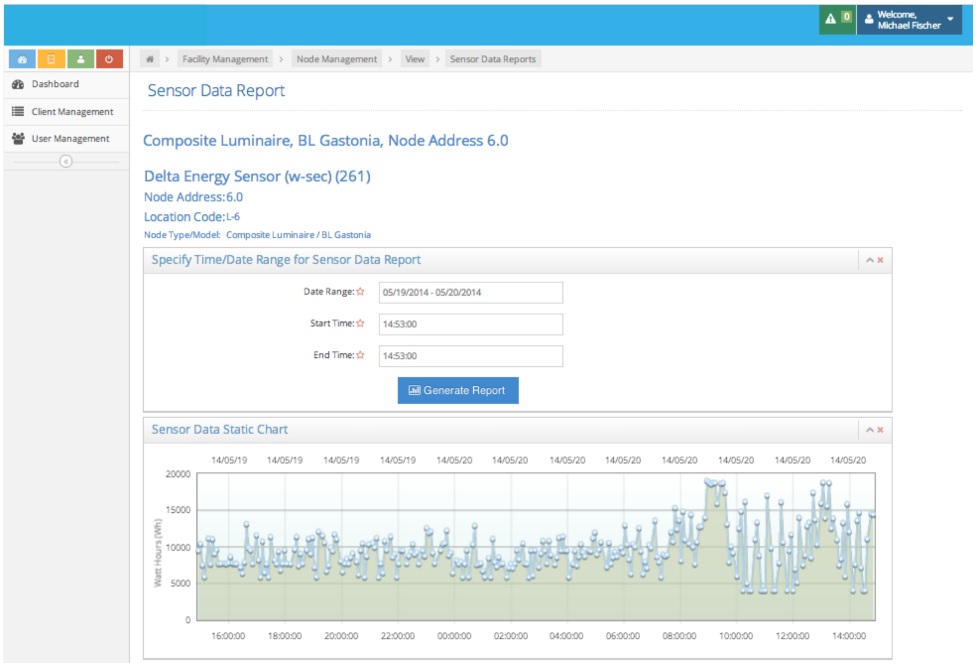

Figure 2.11: A screenshot of a sensor data report view of the web-based MCS GUI. This example shows energy use for a selected luminaire (address 6.0) for a specific queried date and time range.

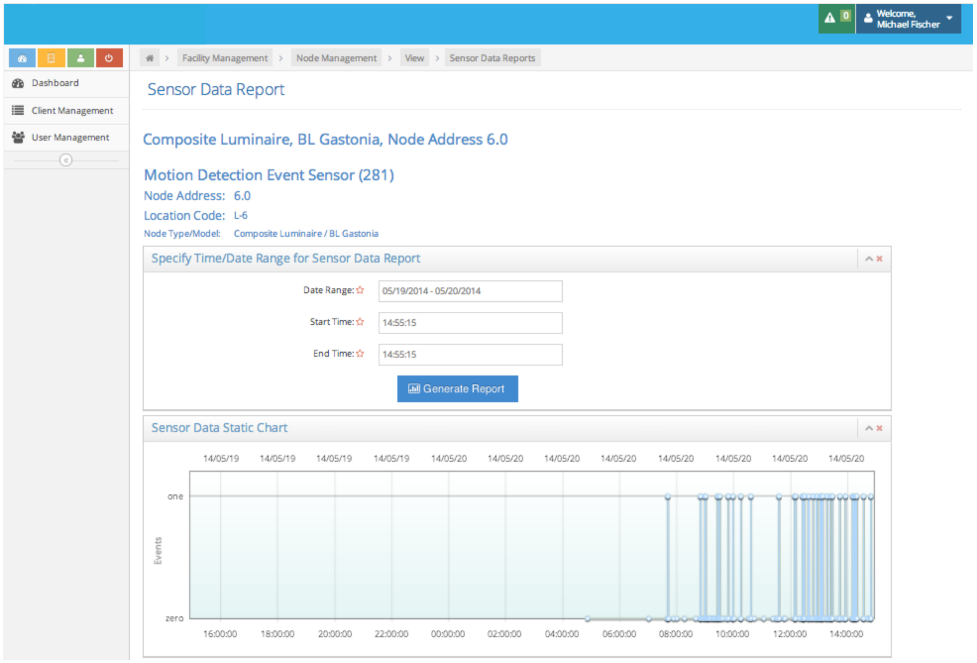

Figure 2.12: A screenshot of a sensor data report view of the web-based MCS GUI. This example shows the state of an integrated passive infrared motion detection sensor for a selected luminaire (address 6.0) for a specific queried date and time range.

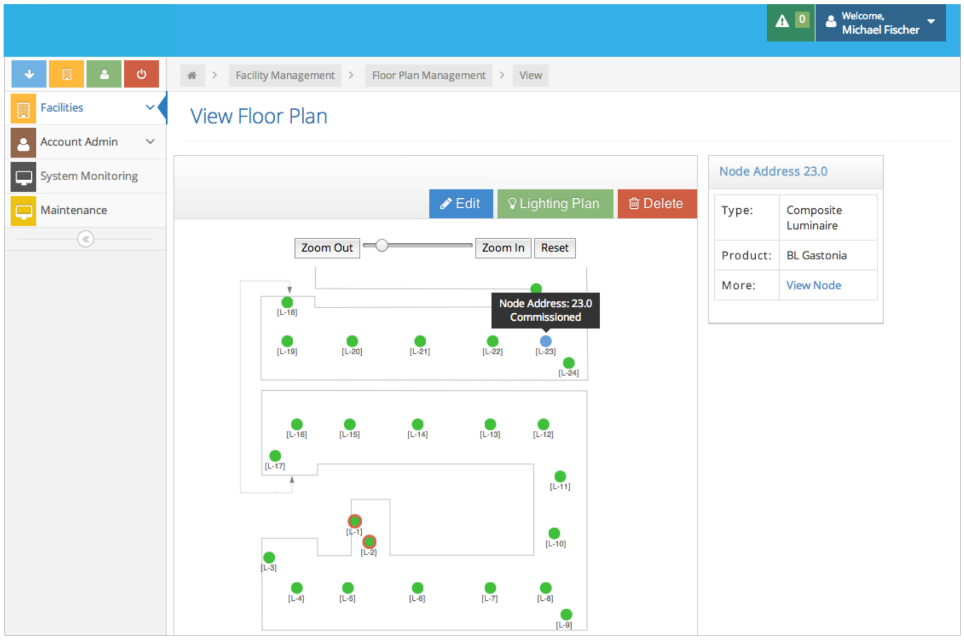

Figure 2.13: A screenshot of a floor plan view of the web-based MCS GUI. This example shows a section of the layout of the underground parking garage pilot facility, indicating luminaire node locations and operating states with coloured circles. Solid green circles indicate an installed and commissioned node that is operating normally, blue indicates a currently selected node, and red outlines indicate nodes detected as having a *disconnected* communication state (i.e. they are currently unreachable).

2.6 Sensor Middleware System (SMS)

The ILS was a technology research and development project that spanned over three years and two main phases of development. As mentioned earlier, the prototype ILS consisted of a WFN of four luminaires managed by one CG, which was polled from the cloud by the pMCS every minute for status, energy use, and occupancy state reports. This prototype proved the key functionalities of the WFN, which was considered the highest risk subsystem in the development of the ILS. It also demonstrated the advantages of cloud-based platforms as a solution to centralized data capture and analysis, as well as rapid infrastructure scaling. The next phase of development called for revision and scale-up of the WFN subsystem. Reporting from the WFN would change to a *push* scheme, among other WFN refinements and feature additions. It would be tested at a pilot size that encompassed a multi-level underground parking facility (52 LED luminaires, replacing the total of 89 original MH HID luminaires). This was accompanied by re-engineering the MCS at an enterprise scale and expanded feature set, as covered in Section 2.5, and the design and development of a new Sensor Middleware System component between WFN and MCS in the ILS hierarchy.

It was understood from the beginning that the scheme of polling the CG via repeated requests for statuses – one request for each luminaire within its wireless network – would not be a scalable solution. The embedded JN5148 SoC controller would become taxed and potentially overwhelmed by the increased burden that would be created by having to track luminaire status and buffering time series data for a network of up to 100 nodes (the estimated limit of wireless nodes that can by managed by one JenNet JN5148-based coordinator). Instead, *push* reporting would ensure that node event reports would be immediately forwarded upward by the CG, avoiding buffering. Buffering, as well as the remaining functional requirements to be implemented for a commercial version of the ILS listed in Table 2.1 – remote firmware update (#15), fault tolerance to ensure no data loss (part of #14) and security (#16) – made sense to be implemented on a SBC residing within the facility with more processing and RAM resources than the CG. Softwares providing these functionalities are well developed on high-level operating systems, many of them being built-in including network protocol and security stacks and libraries. The SMS component was developed on an SBC hosting Linux to act primarily as a message

broker and translator between the MCS and all the facility WFNs, via their CGs. Additionally, the SMS was to offer services to support the management of facility WFNs and sustain the overall QoS at the field level of the ILS. To summarize in high-level terms, the SMS was to be responsible for providing the following services within the ILS:

- brokering and buffering asynchronous field network sensor and status event reports received via LAN from each CG device within the facility that hosts a WFN. Generate related report messages to the MCS for key telemetry data (e.g. energy use and motion events).

- monitoring WFN device operating states and network connectivity states, and generating and sending notifications to the MCS when error states occur,

- processing, translating, and forwarding device configuration commands from the MCS to the WFN,

- receiving firmware file updates from the MCS for WFN nodes and managing the firmware upgrade process in concert with affected CGs, and

- maintaining time synchronization between all ILS subsystems (WFN, SMS, MCS) associated with its given facility.

The SMS was prototyped on a Raspberry Pi (Model B, Rev 2) [**35**] SBC platform developed by the Raspberry Pi Foundation. It was developed with the purpose of teaching basic computer science in schools, but has also become popular in hobby projects and as a rapid prototyping platform. As shown in Figure 2.14, the Raspberry Pi SBC is roughly the size of a deck of cards. It is based upon the Broadcom BCM2835 SoC, which includes an ARM1176JZF-S 700 MHz processor, 512 MB of RAM, VideoCore IV graphics processor unit, one 100 Megabit Ethernet port, two Universal Serial Bus (USB) 2.0 ports, and a Secure Digital (SD) card dock. USB 2.0 is also used as the common bus connecting the Broadcom SoC, Ethernet controller, and SD card dock. The stock Debian-based Linux variant optimized for the Raspberry Pi architecture called Raspbian (Debian Wheezy release) was used for the underlying SMS operating system,

pre-installed on an 8 GB SD card that also provides the local file system. Future revisions of the SMS would likely transition to an industry-grade SBC with resources optimized for network I/O, running a commercial-grade Linux variant.

Figure 2.14: A photo of the Raspberry Pi (Model B, Rev 2) Single Board Computer [**35**].

The detailed requirements, design, and implementation of the SMS software will be covered in Chapter 3.

3 Design and Implementation of the Sensor Middleware System

The physical architecture of the ILS is that of a distributed system of interacting heterogeneous networks. It is by its nature an integration-heavy solution, where the components involved are separated in distance by a few metres to potentially thousands of kilometers, with differing deployment details. Network latency and reliability are not always predictable, especially over long distances and through multitudes of different mediums and network hardware devices (e.g. telephone lines, fiber optics, Ethernet LAN, gateways, routers, switches, and embedded wireless networks) [**55**]. A loosely coupled *Messaging* design pattern was thus seen as the right implementation fit for the ILS – superior to other common integration approaches such as *File Transfer*, *Shared Database*, and *Remote Procedure Invocation* [**55**]. *Middleware* refers to software that enables communication and management of data in distributed applications or systems [**56**]. A *Message Oriented Middleware* (MOM) [**57**] layer would be required in order to support the messaging-based loose coupling of the ILS, and to offer QoS guarantees and the flexibility needed to interface the well-resourced MCS, with scalable IDC infrastructure and variable network latency, with an embedded WFN in each facility with real-time performance requirements. The SMS was purposed with acting as a message broker between the WFN and MCS, as well as offering services to support the management of the WFN nodes (consisting of two node types at this point: CGs and luminaires with embedded sensors).

A *Messaging* design pattern [**55**] was also adopted for the software within the SMS. A number of commercial messaging and middleware software technologies exist, such as Java Message Service [**58**] (part of the Java 2 Enterprise Edition platform) and Microsoft Message Queuing [**59**] (the messaging product built into Windows 2000, Windows XP, and Windows Server 2003-2008). However, it was decided that a custom SMS implementation using open-source technologies on the Linux operating system was preferable. This would increase the likelihood that these technologies would remain royalty-free and maintained on an ongoing basis by the open-source software community. It was also intended that the ILS be extensible in the future to support a variety of possible building subsystems beyond just lighting, and related BAS communication protocols and supervisory functionalities that come with these new domains. The design

of this custom SMS implementation would keep this future flexibility in mind. This chapter will focus on the design and implementation of the first version of the SMS software suite.

3.1 Requirements

The ILS was designed following the *Messaging* design pattern, wherein the SMS sits in the middle of the system hierarchy, and thus the requirements of the SMS pertain mostly to message handling in one form or another. Given this intermediate position in the hierarchy, and that the other ILS subsystems (the WFN and MCS) were designed before the SMS, most of its technical requirements and implementation specifications were elaborated in significant detail apriori. Table 3.1 lists the requirements driving the design of the first release of the SMS.

Req #	Short Name	Description
Interface Requirements		
1	TCP listening	The SMS shall listen via the Ethernet interface for asynchronous TCP/IP datagram messages from CG devices within the facility on a mutually agreed TCP port.
2	HTTP listening	The SMS shall listen via the Ethernet interface for asynchronous HTTP requests from the MCS (for commands), and web browser clients (for SMS Administration Web Application).
3	Secure sockets transactions	The SMS shall support and use Secure Sockets Layer (SSL) encryption and authentication for all transactions with the MCS (as client and server). Note: future versions will add this as a requirement for transactions with the CG as well.
Performance and Reliability Requirements		
4	High availability	The SMS listening services (TCP and HTTP) shall be highly available and non-blocking, employing lightweight and robust multithreaded server technologies.
5	Incoming message buffering	The SMS shall employ a FIFO buffer for all received messages from CG nodes and the MCS to ensure all messages are processed (even if delayed), preserving temporal ordering.
6	Incoming message validation	The SMS shall provide message format and checksum integrity validation before taking any further action in handling the message contents. Malformed messages shall be discarded.
7	Outgoing message delivery guarantee	The SMS shall employ a FIFO buffer for all outgoing messages to the MCS. In the event of a message transmission failure, the message will be returned to the top of the buffer (to preserve temporal ordering) and transmission reattempted repeatedly until successful. Note: this requirement does not apply at this time to messages destined for a CG, due to bandwidth limitations on that device's TCP/IP interface.
8	Node connectivity monitoring and reporting	The SMS shall monitor periodic heartbeat messages emanating from all nodes in the facility WFN (including CG nodes). If a node's heartbeat message is not received over a time interval equal to two periods, a report is to be sent to the MCS indicating a loss of node connectivity. A report shall also be sent to the MCS if a node's heartbeat report resumes after a

		loss of connectivity.
9	Automatic resume SMS after power failure	In the event of power failure / disconnection, upon restoration of power the SMS shall restart and resume normal operation (includes all operating system reboot and services restart steps, LAN connection reestablishment, and restarting of SMS processes, applications, and services daemons).
10	Error logging	The SMS shall log all errors generated at the SMS processes, applications, and services daemon levels. Error logs should be kept on local storage with rolling buffers to limit maximum file size.
Functional Requirements		
11	Energy report forwarding	The SMS shall receive accumulated energy (units of Watt-seconds) reports from any facility CG. These reports originate from individual luminaire nodes and are event-based, but generally arrive at regular intervals. The SMS shall provide message format and transport protocol translation, and forward the modified report to the MCS.
12	Sensor-specific motion detection event forwarding	The SMS shall receive asynchronous single sensor motion detection event reports from any facility CG. These reports originate from any motion detection sensor embedded in or attached to a luminaire node, and include information identifying the sensor type. The SMS shall provide message format and transport protocol translation, and forward the modified report to the MCS.
13	Zone-specific motion detection reporting	The SMS shall receive asynchronous motion detection event reports from any facility CG for logically grouped zones of luminaire nodes (generated by the CG upon receiving one or more single motion sensor detection events within a zone). The SMS shall provide message format and transport protocol translation, and forward the modified report to the MCS.
14	Node state reporting	The SMS shall receive instantaneous operating state reports for individual luminaire nodes from any facility CG. These reports originate from luminaire nodes, and generally arrive from the CG at regular intervals. The SMS shall provide message format and transport protocol translation, and forward one or more reports to the MCS (operating state reports may be filtered by attribute type and split into multiple reports to be forwarded to the MCS, as specified by the MCS data intake API).
15	Node state monitoring database	The SMS shall maintain an internal WFN Node State Database to track network addresses and record the current operational parameters and state attributes of all WFN nodes within its facility (currently this includes luminaire and CG nodes).
16	Node operating parameter adjustment	The SMS shall receive commands from the MCS to modify the value of an operating parameter on a node, and relay the command to the CG that manages the target node.
17	Node operating state attribute query	The SMS shall receive requests from the MCS to query the most recent known value of a given operating state attribute for a node, and return the value to the MCS (retrieved from the SMS's internal WFN Node State Database).
18	Node to zone assignment	The SMS shall receive commands from the MCS to assign a node or group of nodes to a logical zone, provide message parsing and format and transport protocol translation, and forward the modified command to the CG.
19	Scheduled policy to zone assignment	The SMS shall receive commands from the MCS to assign a schedule for a given operating policy (*ocsHigh*, *ocsLow*, *ocsDelay, ocsConfig*) to an existing zone, provide message parsing and format and transport protocol translation, and forward the modified command to the CG.
20	WFN firmware update management	The SMS shall receive updated WFN node firmware files for specific nodes (or groups of nodes) from the MCS and store them locally. The SMS shall then push the new firmware to all affected nodes via the CG.

		The SMS will monitor for successful download completion via node heartbeat/firmware report messages, and forward these reports to the MCS. The SMS shall receive a firmware activation command from the MCS at some point after all affected nodes have completed new firmware download and shall relay the activation command to the CG(s) of all affected nodes to trigger a reboot and switchover to running the new firmware.
21	Field device clocks synchronization	The SMS shall initiate and coordinate daily synchronization of the real time clocks of itself and all WFN nodes within its facility (including CGs) with the Internet Network Time Protocol (NTP) server. Note: the MCS independently synchronizes itself with the NTP server.
Operational Requirements		
22	Remote configuration and administration	The SMS shall be configurable by remote Secure Shell (SSH) login and a password authenticated HTTPS web browser-based SMS Administration Web Application for ILS Administrator users. Configurable options will include all ports the SMS will listen on and URLs it will serve and transmit messages to. Administrative options will include WFN Node State Database inspection and modification (with caution), and SMS processes, tasks, and queues monitoring and configuration.
Implementation Requirements		
23	Low-cost, small physical form factor hardware	The SMS shall be implemented on low-cost hardware capable of supporting high-level operating systems and programming languages, modern third party open source software, and up-to-date and regularly maintained Internet communication protocols and security technologies. The hardware should be implemented in a small physical form factor (i.e. a SBC) for simple and discreet installation in a customer facility (i.e. no special power or ventilation requirements).

Table 3.1: The list of requirements for the Sensor Middleware System.

3.2 Use Case Model

The general use cases that the SMS was to support were derived from the requirements of Table 3.1. These use cases are summarized in the UML context diagram of Figure 3.1. Most of the duties of the SMS pertain to machine-to-machine (M2M) message handling, with the exception of direct interaction by an ILS Administrator user via the SMS Administration Web Application.

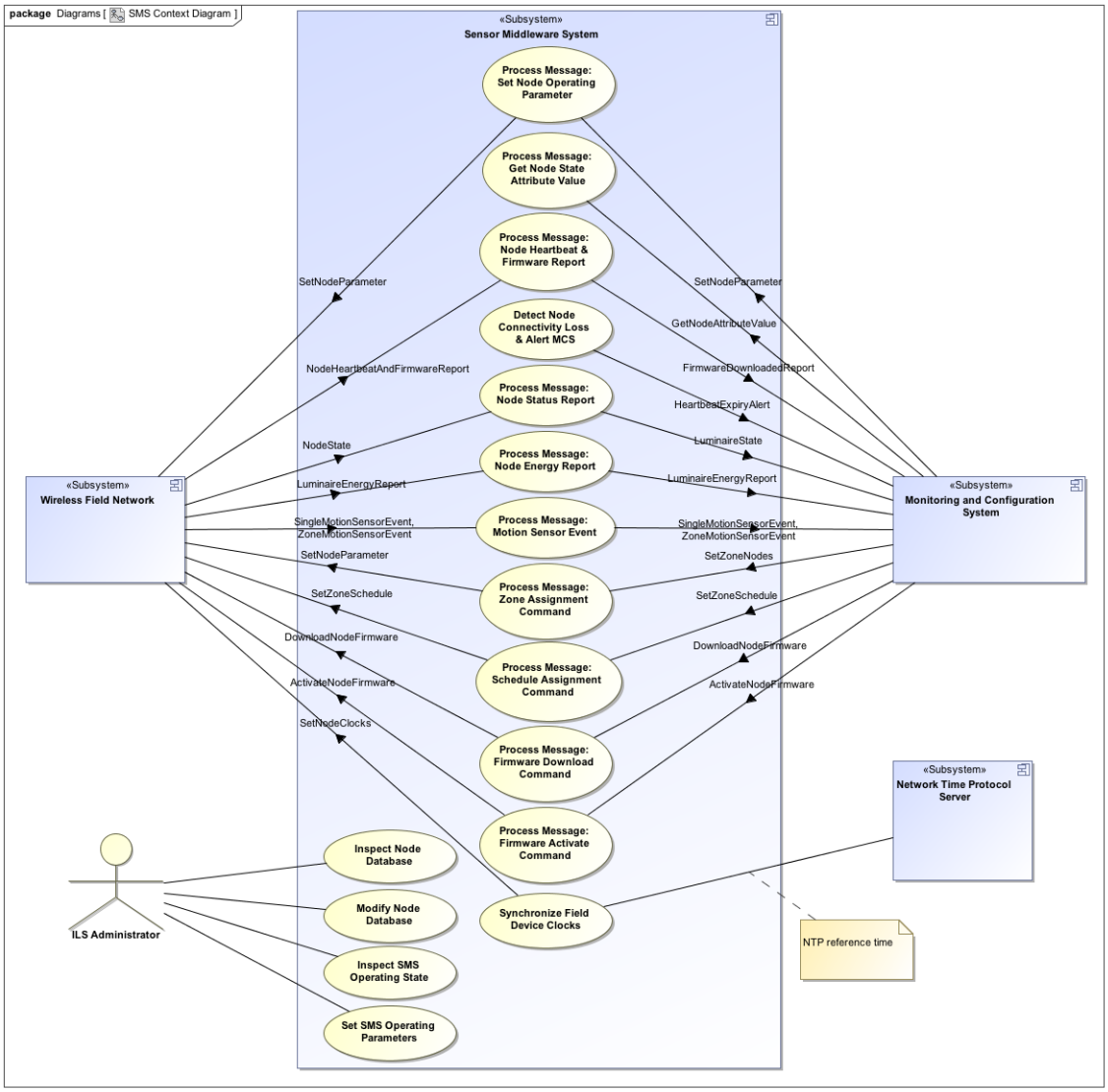

Figure 3.1: UML context diagram illustrating the use cases that the SMS is responsible for supporting.

3.3 Messaging Between ILS Subsystems

The ILS is a distributed system comprised of a number of distinct subsystems. Each subsystem runs independently, but the entire system only functions through the coordination of all subsystems in a loosely coupled way. Messaging enables this loose coupling, allowing subsystems to exchange data or commands across multiple layers of network using an asynchronous "send and forget" approach. Using this approach, the subsystem that sends a message can immediately resume other tasks while the information is transmitted by the messaging system to its destination. Optionally, the sender can be notified later of message delivery through a callback. This tends to make design and implementation more complex. Conversely, asynchronous messages can be

retried up to a specified number of times, or repeatedly until successful delivery, to make communication more reliable [**55**]. The ILS uses a combination of these approaches.

This section will give an overview of messaging interactions within the ILS. Details of message contents will be given in Section 3.4. There are three message-based interaction flows among the subsystems of the ILS, those that originate within the WFN, those that originate at the MCS, and those that originate at the SMS. Messages within the WFN follow the proprietary JenNet protocol. All messages from the WFN CGs and upward are sent over the Transmission Control Protocol / Internet Protocol (TCP/IP). Figure 3.2 illustrates the architecture of the TCP/IP stack. Messages between CGs and the SMS use standard TCP/IP datagrams (a.k.a. TCP datagrams) at the transport layer with custom payloads. TCP datagrams were chosen at this level for their lightweight implementation and reasonable level of message delivery reliability. Messages between the SMS and MCS use Hypertext Transfer Protocol (HTTP) Representational State Transfer (REST) requests at the application layer with JavaScript Object Notation (JSON) object payloads. "RESTful" Application Programming Interfaces (APIs) between systems on the Internet have become ubiquitous and greatly simplify interoperability across heterogeneous systems handling large amounts of data, as well as offering easy interfacing with humans via web-enabled applications.

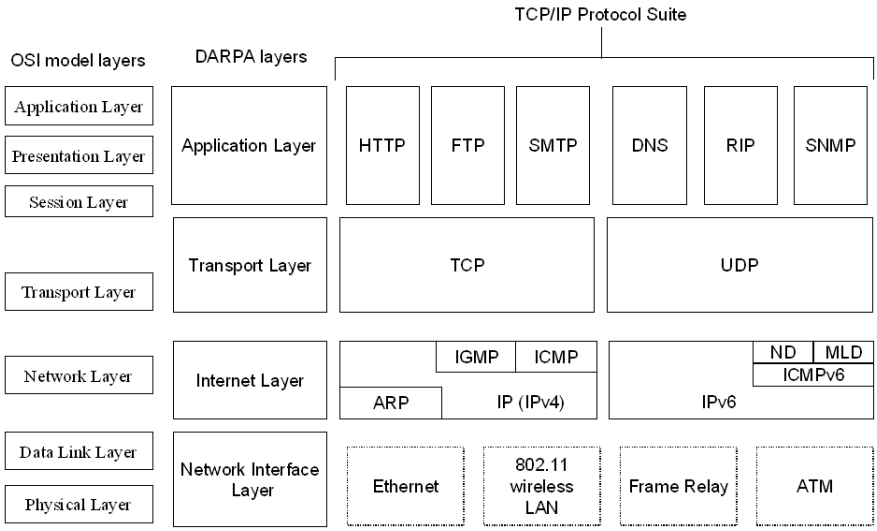

Figure 3.2: Architecture of the Transmission Control Protocol / Internet Protocol (TCP/IP) stack [**60**].

The SMS is responsible for receiving the messages from the various CGs within its facility, as well as from the MCS in the cloud, and plays a message broker role between these layers. The *Message Broker* pattern (the messaging equivalent to the GoF *Mediator* pattern) [**55**] [**61**], depicted in Figure 3.3, is thus employed here, where the purpose is to reduce coupling between subsystems that communicate via messaging. In the case of WFN messaging to the SMS, neither luminaires nor CGs have any knowledge of the ultimate destination address(es) of their messages (the MCS); only the SMS needs this information. Likewise, in the case where the messaging interaction begins at the MCS, the SMS keeps track of the parent IPv4 LAN addresses of the CGs associated with all luminaire nodes, and directs messages accordingly.

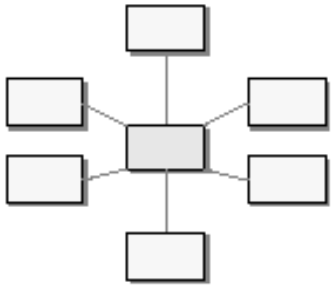

Figure 3.3: The *Message Broker* design pattern [**55**]. A central broker can receive messages from multiple sources, determine the destination, and direct the message to the correct channel so as to reach the destination.

The first message-based flow within the ILS is illustrated in the sequence diagram in Figure 3.4, where a message is originates at a WFN node (either a luminaire or CG). These messages are generally called *event messages* [**55**], and are used to transmit information about events that have occurred within one object that another object needs to know about. To avoid wireless network congestion, event messages from luminaire nodes to CGs are sent asynchronously without end-to-end acknowledgement (i.e. "send and forget"). This trade-off avoids doubling the amount of traffic through the wireless network, and is acceptable given that no events from any single luminaire are critical to the smooth operation of the ILS. In fact, some important luminaire event information (i.e. active firmware version and motion detection state) is repeated in periodic luminaire node state messages, providing some information redundancy across messages.

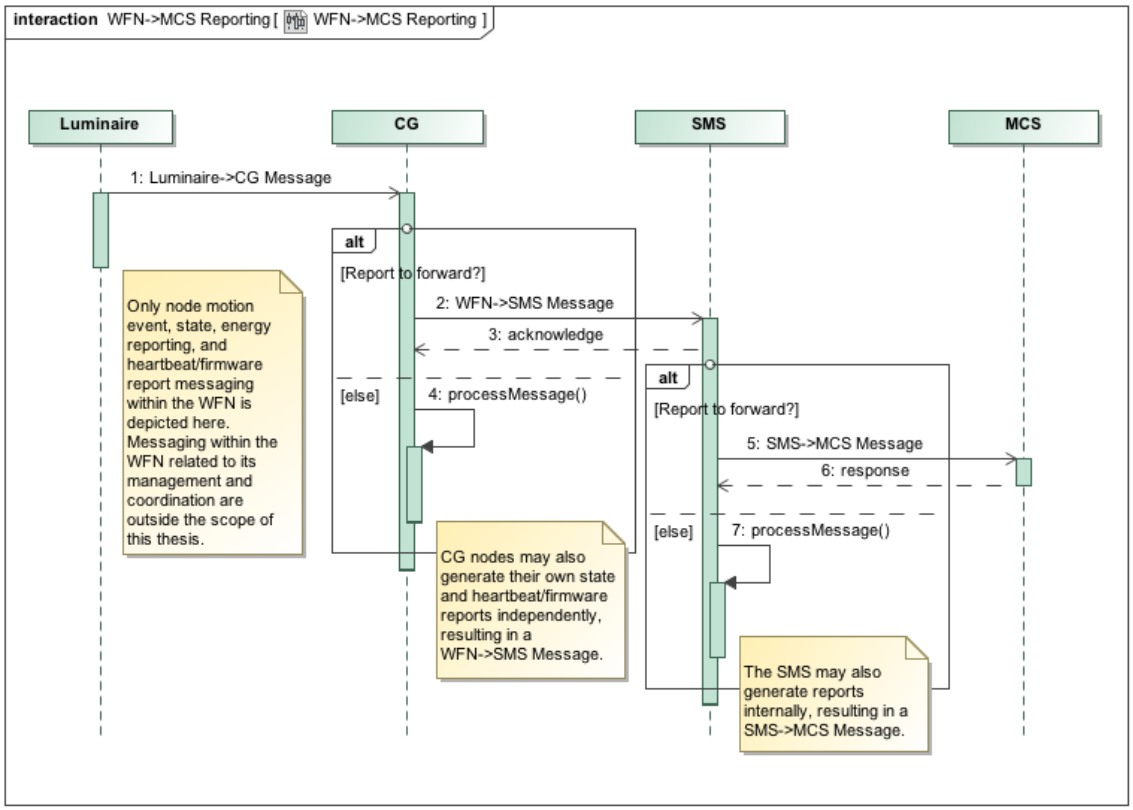

Figure 3.4: UML sequence diagram showing the message-based interaction flow beginning at a luminaire and (optionally) ending at the MCS. A luminaire sends an asynchronous event message to its CG over the wireless network without acknowledgement. Depending on the message type, the CG can then translate and forward it over the LAN to the SMS. Similarly, depending on the message type, the SMS can then translate and forward the message over the Internet to the MCS.

Depending on the message type, messages sent by luminaire nodes to the CG at the top of their wireless network are either forwarded upward to the SMS or processed by the CG. Motion events, node state and energy reports, and heartbeat and firmware version reports are forwarded, whereas messages related to management of the JenNet wireless network (e.g. network joining, recovery, zonal lighting action coordination) are processed within the CG[8]. There are two exceptions to this rule not explicitly shown in Figure 3.4:

- a CG node will generate and send a zonal motion event report to the SMS based upon processing individual luminaire motion events and identifying the zone each luminaire belongs to from an internal table it maintains, along with forwarding the individual luminaire motion event report to the SMS, and

[8] Lower-level details of messaging within and management of the JenNet-based WFN are not covered.

- a CG node will generate and send its own node state and heartbeat/firmware report messages to the SMS.

CGs sending TCP datagram messages to the SMS will wait a specified amount of time for acknowledgement and then retry up to a specified (adjustable) number of times before giving up. It is assumed however, that while operating properly, the private ILS facility LAN infrastructure should never form a bottleneck between CGs and the SMS.

Following from this message broker functionality comes the need to deal with translating message formats, and thus the *Message Translator* pattern (the messaging equivalent to the GoF *Adapter* pattern) [**55**] [**61**] was applied, depicted in Figure 3.5. In the first flow (Figure 3.4), the SMS parses the messages arriving from CGs – the message payload containing one long string of byte codes in a proprietary format – and replaces the byte codes with human-readable key-value pairs, then encapsulates them in a JSON object along with unique identifying information of the facility and SMS. The format of this JSON object is that specified for the MCS data intake system API, thus no additional translation is needed at the MCS upon reception of a message from the SMS. Not explicitly shown in Figure 3.4, but worth noting, is that reports the SMS receives from CG nodes may also be parsed for information to update its SMS internal WFN Node State Database (covered in Section 3.5).

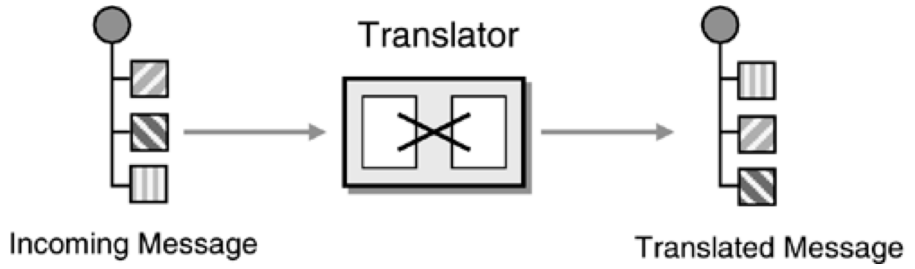

Figure 3.5: The *Translator* pattern [**55**]. A key function of the SMS is to translate incoming messages from one format to another for subsequent transmission to another subsystem. The *Translator* pattern is implemented in the SMS for communications in both directions, from the CG through to the MCS, and vice-versa.

In the second message-based flow, depicted in Figure 3.6, the SMS translates incoming HTTP REST requests from the MCS – the message payload containing a JSON object of key-value pairs – into messages of a proprietary format, each containing a string command code followed by an integer parameter value(s). These messages are sent to

the parent CG(s) of the target luminaire nodes. A CG is not capable of splitting apart a message containing multiple commands destined for multiple nodes, and sometimes these packaged commands will span nodes belonging to more than one CG within a facility (and CGs have no knowledge of each other). Thus, the SMS also takes care of splitting packaged command messages from the MCS into numerous outgoing messages to CG nodes, then issuing them sequentially, separated by constant delays to avoid overflowing the CG's command input.

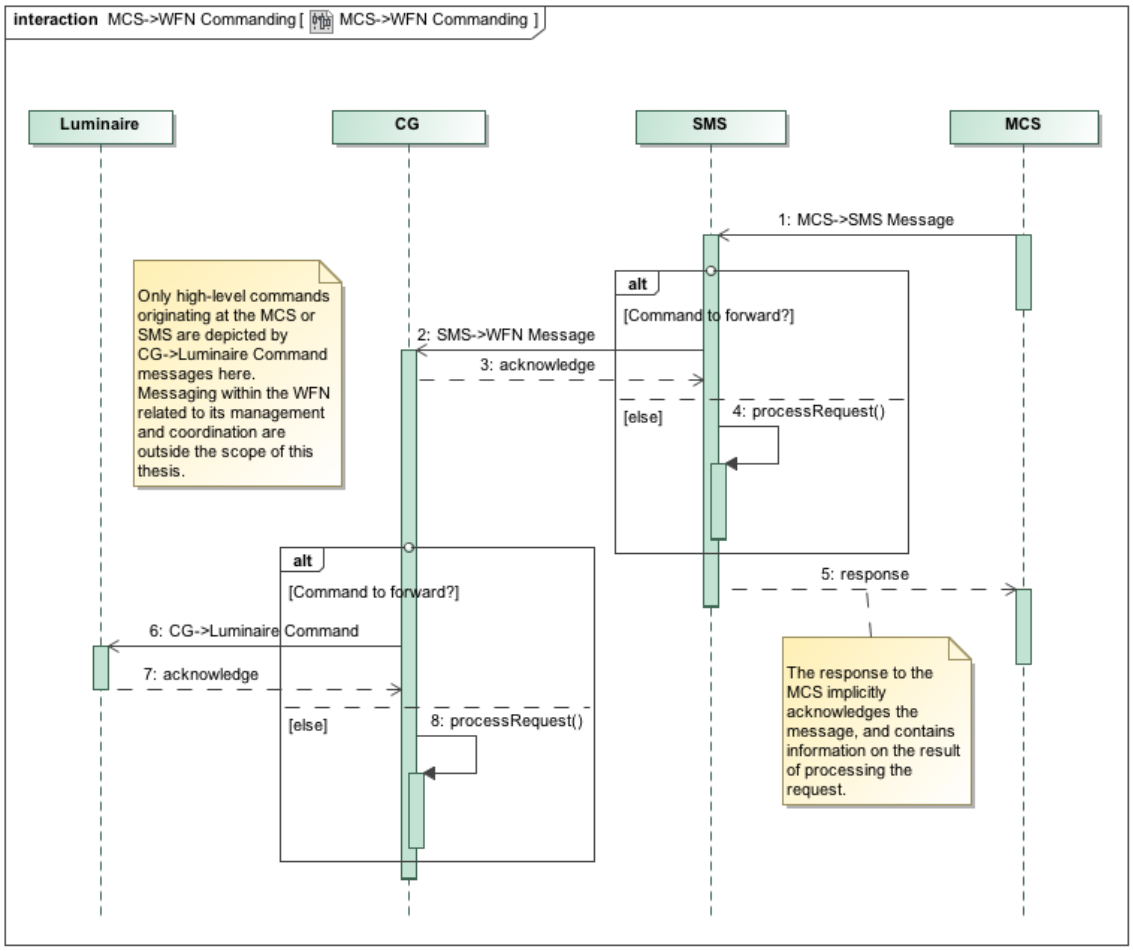

Figure 3.6: UML sequence diagram showing the message-based interaction flow beginning at the MCS and (optionally) ending at a luminaire node. The MCS sends an asynchronous command message to a facility's SMS over the Internet and waits for acknowledgement. Depending on the message type, the SMS can then translate and forward it over the LAN to the CG associated with the target luminaire node (or send to the CG itself). Similarly, depending on the message type, the CG can then translate and forward the message over the wireless network to the luminaire node.

The final message formation pattern applied by the SMS is the *Envelope Wrapper* pattern [**55**], depicted in Figure 3.7, used to repackage messages for transmission

according to the outgoing protocol. When sending messages to facility CGs, a TCP datagram wrapper is used. When sending messages to the MCS, an additional HTTP REST request wrapper is formed around a TCP datagram. Figure 3.8 depicts the layers of data encapsulation employed within TCP/IP. The destination IP address is provided in the IP header, and the destination port is provided in the TCP header. For HTTP, the REST request header is provided in the application header, which includes the type of request (i.e. POST or GET) and a Secure Sockets Layer (SSL) certificate.

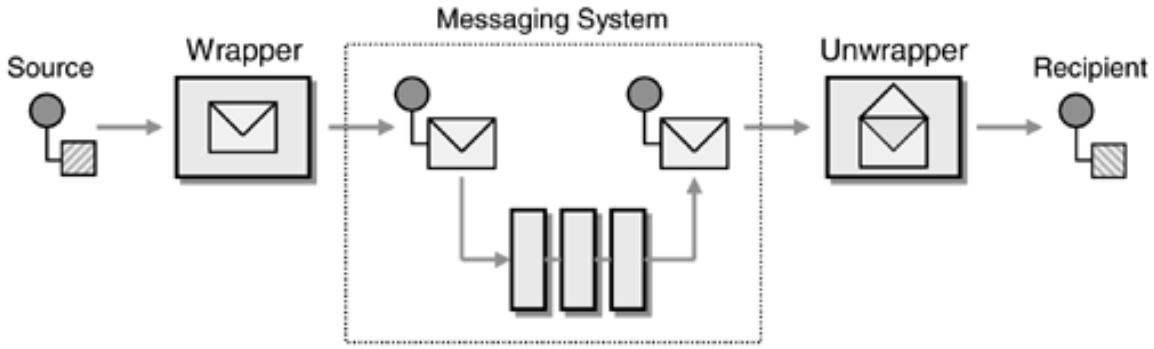

Figure 3.7: The *Envelope Wrapper* pattern [**55**]. It is used to wrap data inside an envelope that is compliant with the messaging system. The destination subsystem unwraps the message upon arrival, revealing the original message contents.

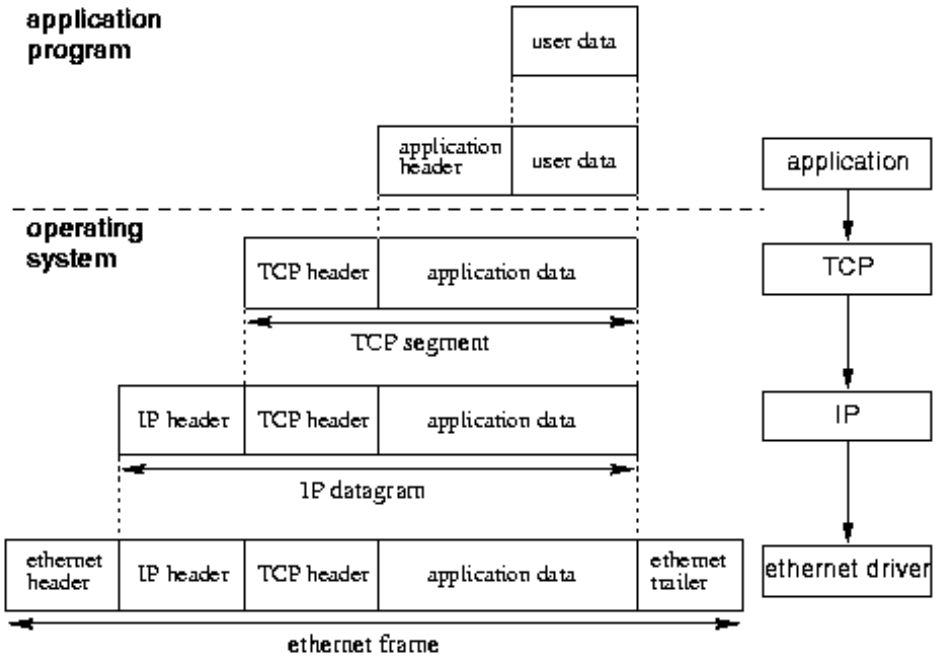

Figure 3.8: Encapsulation of message data in from the link layer to the application layer using TCP/IP [**62**]. Intra-facility messages between the SMS and CGs are wrapped in TCP/IP datagrams (labeled here as the TCP segment). Messages between the SMS and MCS are wrapped in HTTP REST requests at the application layer. The destination IP address is provided in the IP header, and the destination port is provided in the TCP header. For HTTP, the REST request header is provided in the application header.

The requirement of guaranteed delivery (Table 3.1, requirement #7) applies to all messages sent by the SMS to the MCS, thus the *Guaranteed Delivery* pattern, depicted in Figure 3.9, was applied. The essential aspect to this pattern is that the messaging middleware stores the message and retries delivery until successful, in the event that the receiver (or the network itself) is unavailable. In the SMS, a FIFO buffer is used to store messages in the order they were produced. Transmissions are retried repeatedly, and a message is only removed from the buffer once successful transmission to the MCS has been confirmed (by way of the HTTP response code). In the worst case scenario, prolonged network / MCS outage could lead to outgoing messages being lost by way of all or part of the FIFO ring buffer being overwritten (preferable to running out of disk space on the SMS, which would also cause problems).

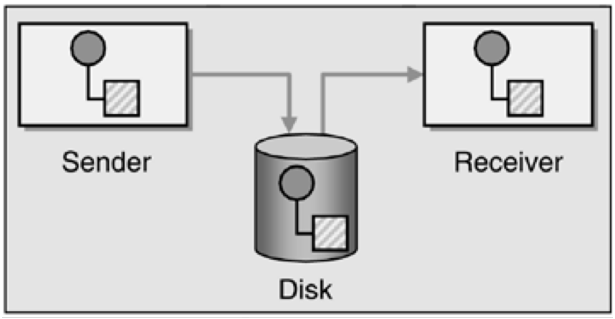

Figure 3.9: The *Guaranteed Delivery* pattern [**55**]. To ensure messages are delivered to their intended destination, the messaging middleware is responsible for storing them on disk and retrying transmission until acknowledged successful by the recipient.

As mentioned, the SMS also plays a facility WFN supervisor role, which is limited at this point to firmware update management, node connectivity monitoring, and time synchronization. Messages to the MCS related to these functionalities are internally generated by the SMS when necessary (to be described in Section 3.6).

3.4 Message Classes

As mentioned, the SMS is responsible for sending and receiving messages using two TCP/IP protocols: TCP datagrams and HTTPS REST requests. TCP datagrams are used for communications via the LAN with all CGs managing WFNs within a facility, and HTTPS REST requests are used for communications with the MCS, as well as remote ILS Administrator users via the SMS Administration Web Application. The

following subsections will describe the classes that represent the information contained in these various message types.

3.4.1 Intra-Facility TCP Datagrams

The class diagram of Figure 3.10 depicts the set of possible TCP datagram messages that the SMS may receive from any CG within its facility. These may be messages originating from luminaire nodes within the WFN or CGs themselves. The following message types are generated by a CG, triggered by events received from individual luminaire nodes in its WFN: *LuminaireStateReport, LuminaireEnergyReport, SingleMotionSensingEvent*, *NodeHeartbeatAndFirmwareReport,* and *FirmwareDownloadedReport*. CG nodes generate the following message types: *NodeHeartbeatAndFirmwareReport, FirmwareDownloadedReport*, and *ZoneMotionSensingEvent*.

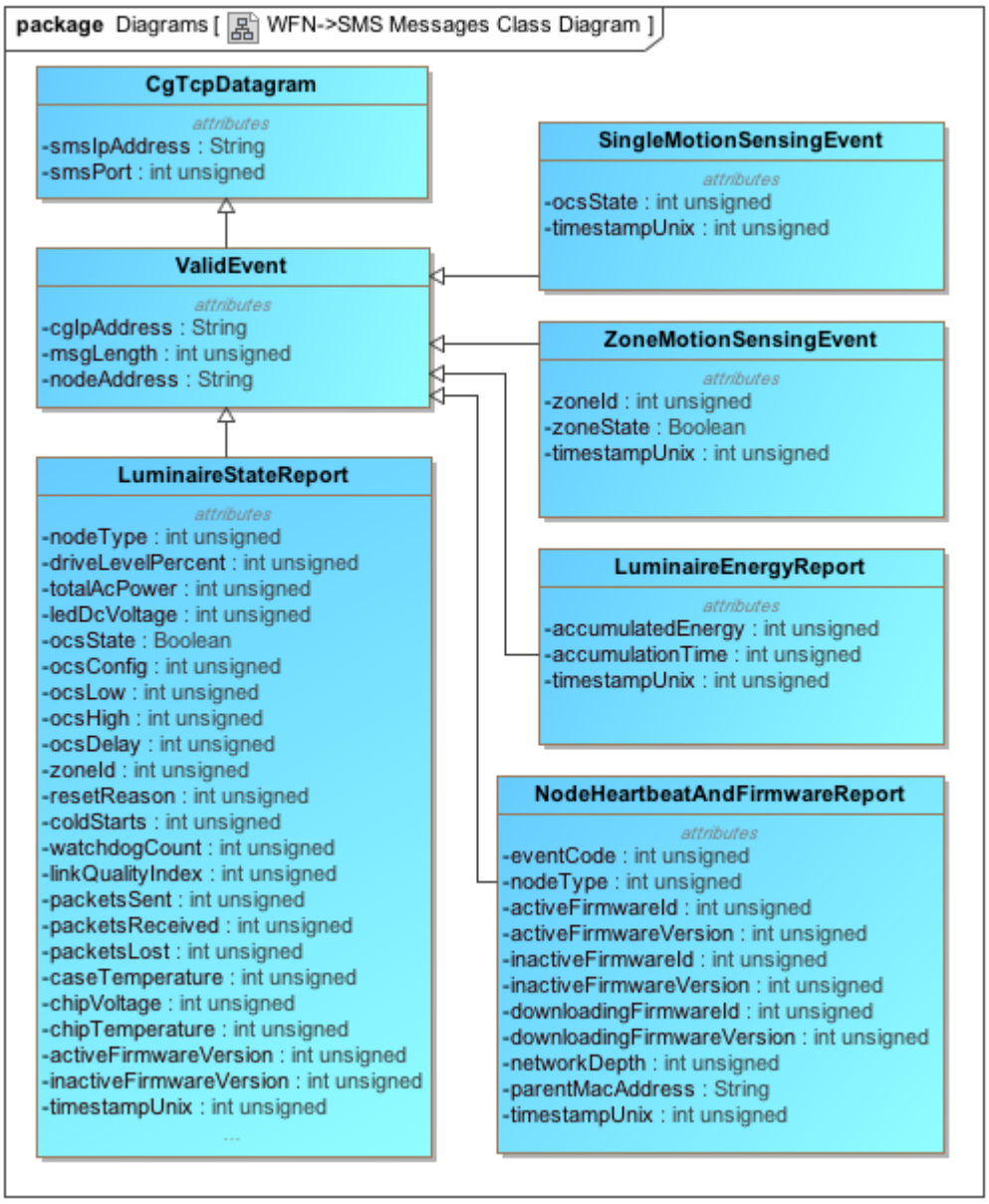

Figure 3.10: UML class diagram describing the data payload of TCP datagram messages sent by facility CGs to the SMS.

The class diagram of Figure 3.11 depicts the TCP datagram messages that may be sent by the SMS to any facility CG. *DownloadNodeFirmware, ActivateNodeFirmware, SetZoneSchedule,* and *SetNodeParameter* represent commands originating at the MCS (by an ILS Administrator) that are handled and translated by the SMS – acting as a message broker – for transmission to the WFN. *SetNodeClocks* is generated within the SMS by a scheduled periodic task (the schedule can be set by the ILS Administrator via the SMS Adminstration Web Application).

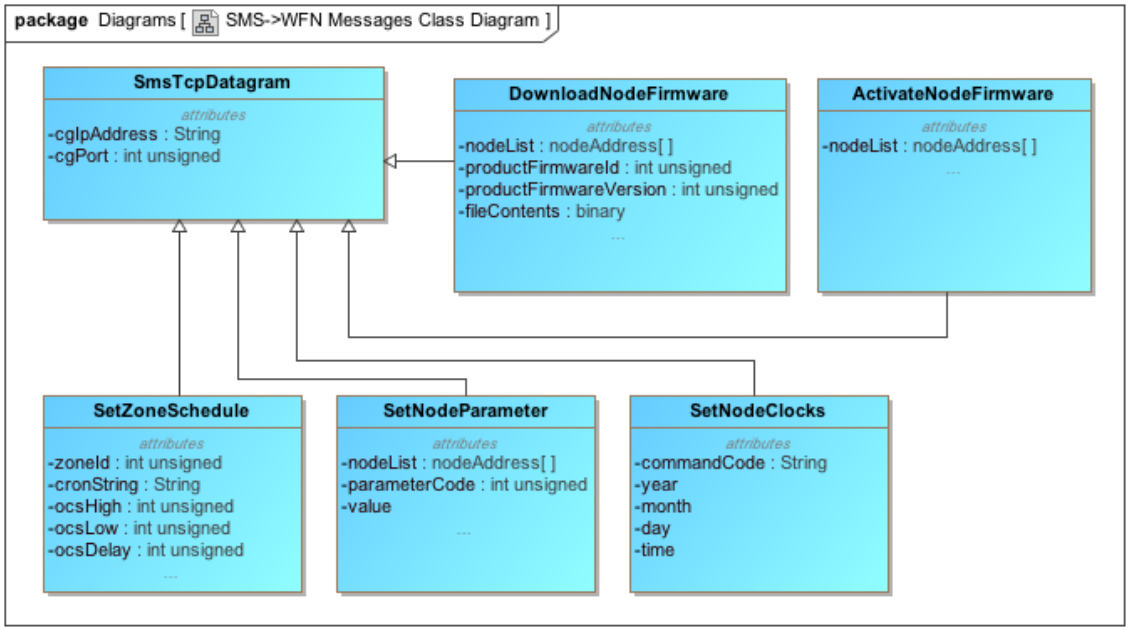

Figure 3.11: UML class diagram describing TCP datagram messages sent by the SMS to the CGs managing WFNs within its facility.

3.4.2 Extra-Facility REST Requests

The class diagram of Figure 3.12 depicts the HTTP REST request messages that may be sent by the SMS to the MCS. Again in its message brokering role, the SMS generates *LuminaireState, LuminaireEnergyReport,* and *SingleMotionSensorEvent,* and *ZoneMotionSensorEvent* messages in response to the reception of related messages from the CG. *ConnectionStateChangeEvent* and *FirmwareDownloadedReport* are derived from comparing incoming *NodeHeartbeatAndFirmware* TCP Datagrams to existing node states in the WFN Node State Database, to be elaborated upon later. All such MCS REST Request Messages contain standard header attributes for the protocol [**63**], enacting a POST transaction using SSL to a pre-determined URL and port hosted by the MCS data intake subsystem.

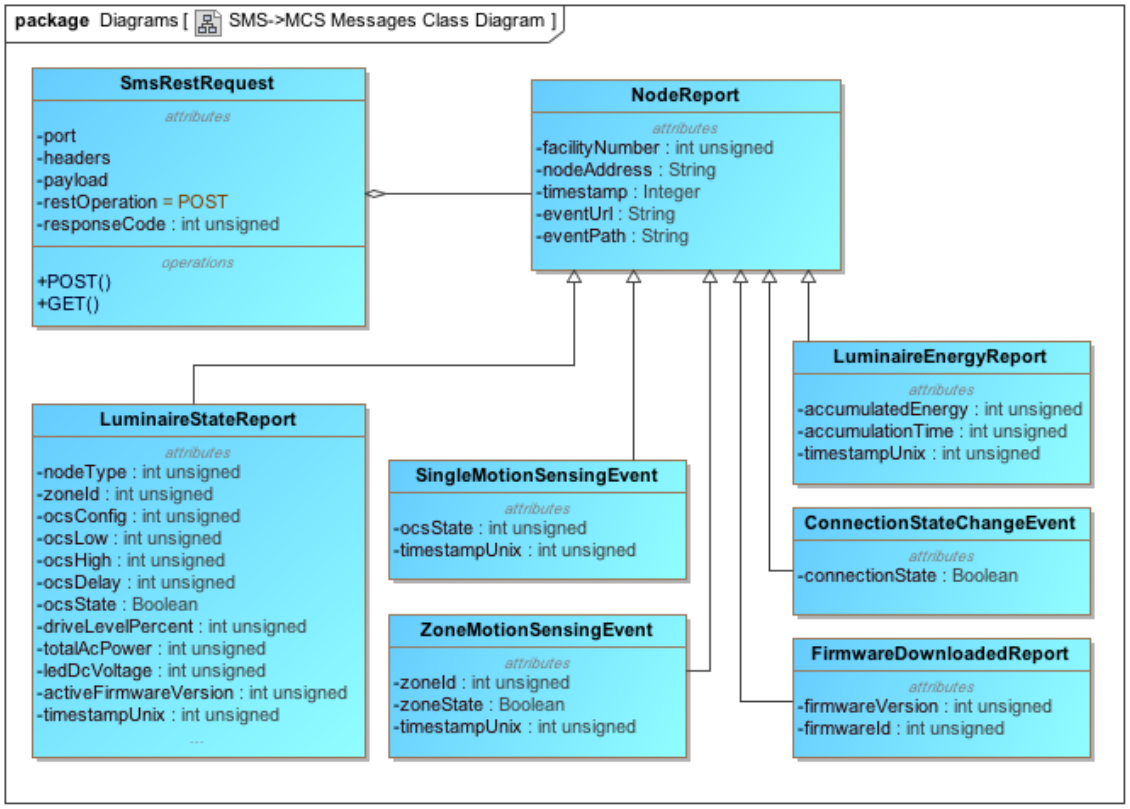

Figure 3.12: UML class diagram describing HTTPS REST request messages sent by the SMS to the MCS.

The class diagram of Figure 3.13 depicts the HTTPS REST request messages that may be received by the SMS from the MCS. The SMS presents a URL and port through a web server for receiving these requests from the MCS.

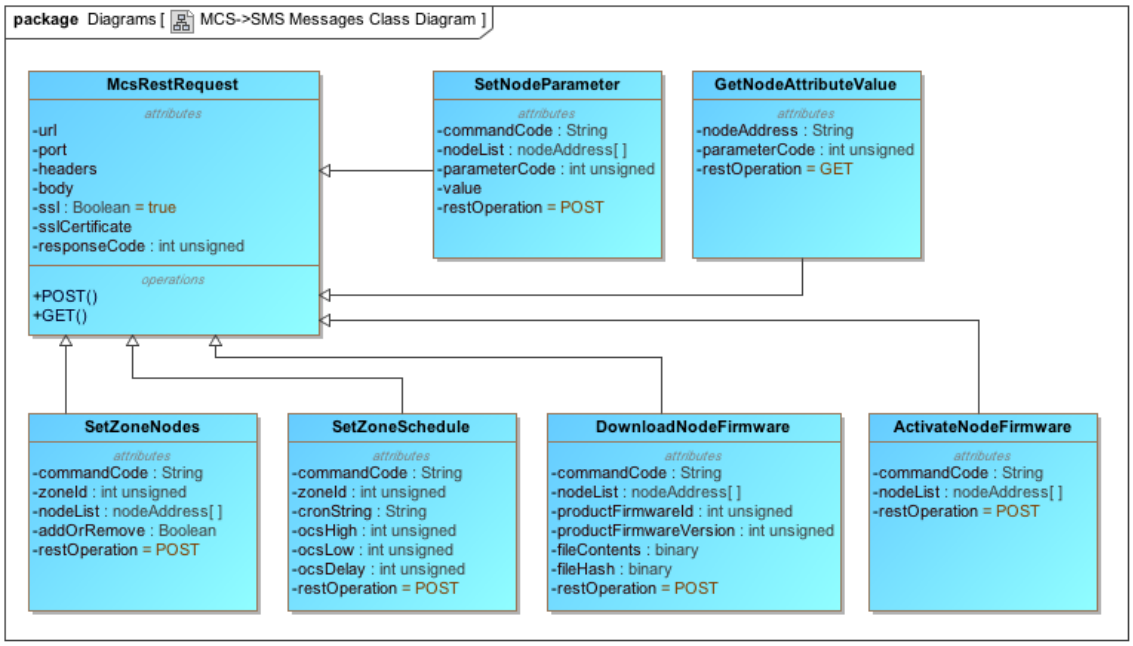

Figure 3.13: UML class diagram describing HTTPS REST request messages received by the SMS from the MCS.

3.5 WFN Node State Data Model

The SMS maintains an internal WFN Node State Database to track the current operating parameters and running state of all WFN nodes within its facility. Implemented in SQLite [**64**], it enables the SMS to provide automated network connectivity monitoring of WFN nodes (luminaires and CGs) and low latency responses to node state or operating parameter queries by the MCS. It also allows administrators to conduct WFN diagnostics via the SMS Administration Web Application, inspecting the last known operating states of facility nodes without burdening the WFN with bulk node status queries[9] (which may not even be possible in some failure modes).

The WFN node state data model is depicted in the class diagram of Figure 3.14. *NodeState* is the base class of the data model, allowing for a variety of possible wireless node types to be included in the future by class inheritance. The CG belongs to this base class, and at present time, *LuminaireState* is the only supported additional node type state class. The *FirmwareFile* class is used to encapsulate a firmware binary file downloaded from the MCS for a node and kept in local storage. Given that many (or all) nodes within

[9] This functionality is for low-level debugging of the WFN only and must be used with caution, and is not included among functionalities the SMS must support.

a facility will share the same firmware version, and to save disk space, there may be many nodes associated with the same *FirmwareFile* instance.

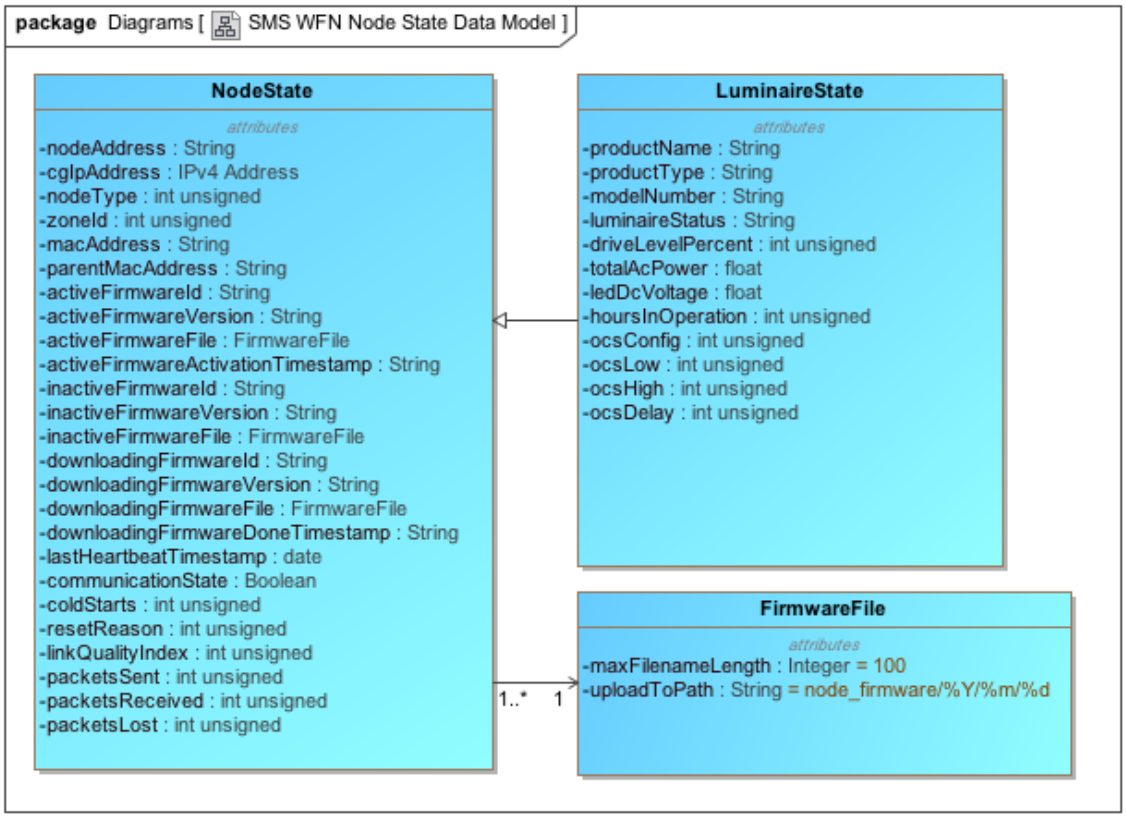

Figure 3.14: UML class diagram describing the data model of the WFN Node State Database implemented by the SMS.

3.6 Software Implementation

The SMS was largely implemented using the high level Python scripting language [**65**]. Python is employed for development across many domains, offering a wide breadth of packages/libraries allowing for rapid prototyping and test. A number of popular and continuously maintained open source applications across the domain of web- and message-based systems have been developed in Python. It was chosen for developing the SMS because of the versatility these factors together offer, allowing for rapid development of customized solutions, as well as its typically good run-time performance. The SMS largely separates the software implementations of the message brokering and user-facing web services (i.e. the SMS Administration Web Application), the common thread between them being the WFN Node State Database.

3.6.1 Software Packages and Components

Figure 3.15 gives the software package diagram showing the various software layers[10] of the SMS: Domain, Application, Presentation, and Technical Services. All of the packages used were either developed in Python and/or offer a Python API.

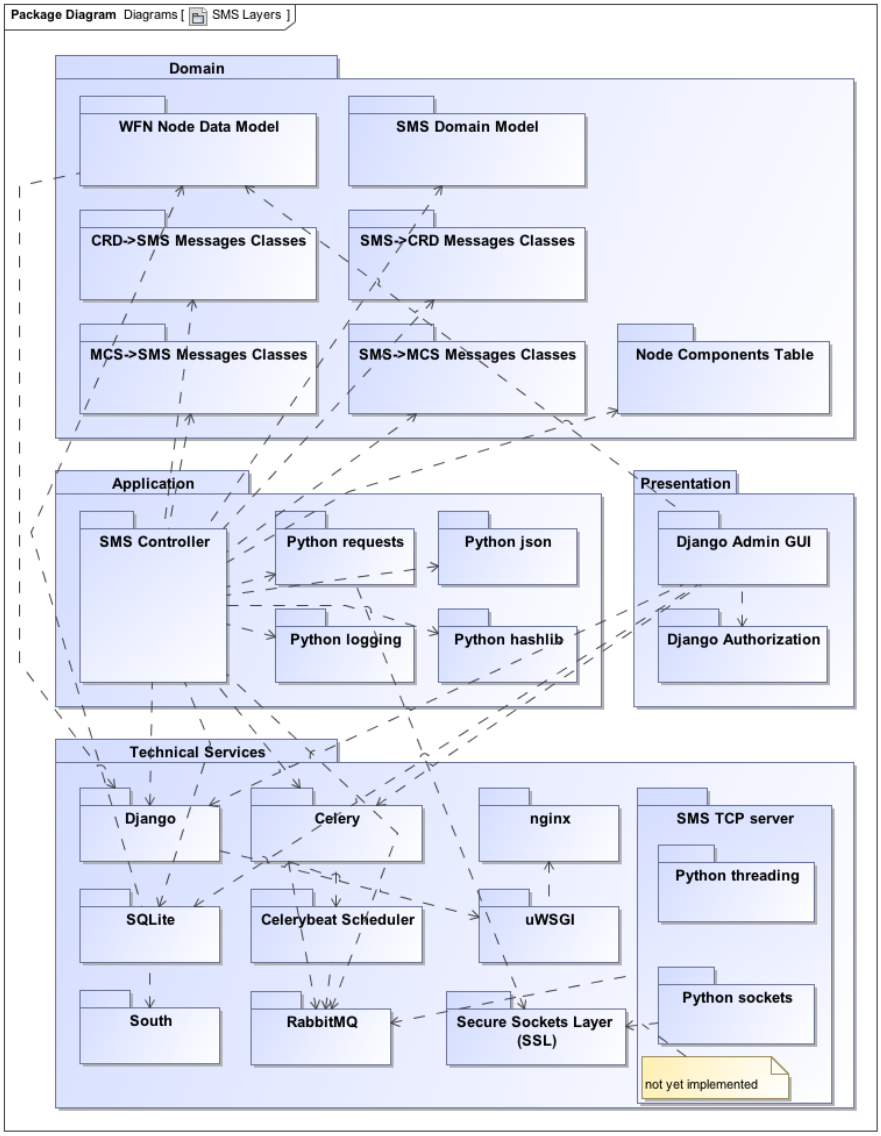

Figure 3.15: Software package diagram showing the various software layers of the SMS: Domain, Application, Presentation, and Technical Services. All of the packages used were either developed in Python and/or offer a Python API.

[10] Excluding the Debian-based Linux operating system and standard installed services.

At the Domain layer, custom Python classes were created to support the various message types described in Section 3.4. The code snippet of Figure 3.16 shows two of the classes implemented by the SMS to encapsulate incoming messages from the CG.

```
class ValidEventObj(object):
    "Valid event messages result in objects being created and populated with parsed message data."
    def __init__(self, unix_timestamp, crd_ip, rx_msg_len):
        self.node_address = {}
        self.msg_timestamp_unix = unix_timestamp
        self.crd_ip = crd_ip
        self.rx_msg_len = rx_msg_len

class EvtOcsObj(ValidEventObj):
    "OCS (motion/occupancy) event message object. This is instantiated and populated upon \
    receiving a valid TCP SingleMotionSensingEvent message."
    def __init__(self, unix_timestamp, crd_ip, rx_msg_len):
        ValidEventObj.__init__(self, unix_timestamp, crd_ip, rx_msg_len)
    self.node_address = {}
    self.ocs_state = {}
```

Figure 3.16: Implementation of two of the WFN->SMS Messages classes shown in Figure 3.10 (*ValidEventObject* and *SingleMotionSensingEvent*, respectively).

At the Application layer, the SMS Controller is comprised of a number of scripts encapsulated as worker tasks written in Python that tie together the message-based functionality of the SMS, employing the following Python libraries:

- *requests* – HTTP(S) REST request messaging client [**66**]. Used to form and send outgoing REST requests from the SMS to the MCS. The code snippet of Figure 3.17 gives an example of a typical usage of *requests*.

```
# Using requests for HTTPS encrypted connections to a remote server:
    full_url = str(self.url) + str(self.path)
    if operation == 'POST':
        r = requests.post(url = full_url, data = payload, headers = self.headers, cert =
('/path/to/certificate/file.crt', '/path/to/key/file.key'), verify = False)
    if operation == 'GET':
        r = requests.get(url = full_url, cert = ('/path/to/certificate/file.crt',
'/path/to/key/file.key'), verify = False)
    returned_data = r.json()
    request_status_code = returned_data['request_status']
```

Figure 3.17: Sample code showing usage of the Python HTTP *requests* module, with SSL certificates and no 3rd party authority verification.

- *json* – JavaScript Object Notation (JSON) lightweight data interchange format [**67**]. JSON objects are human-readable nestable key/value data objects very similar in format to existing data types in a number of programming languages (including the Python dictionary type), and are easy to parse, and thus are an ideal format for data interchange. JSON objects are used to store message

payloads for incoming and outgoing HTTP REST requests, as well as payload data parsed from incoming TCP datagrams, and are used internally for passing data between SMS functions and worker tasks. The code snippet of Figure 3.18 shows two samples of JSON object templates of the payloads of a) an outgoing *LuminaireEnergyReport* message to the MCS, and b) an incoming *DownloadNodeFirmware* message from the MCS.

```
# JSON object template of a SMS->MCS event report.
# The cp_code keys represent the attributes to be reported
{"fa":<facility_number>
,"no":<node_address>
,"dt":[{"cp":<cp_code1>,"val":<val_1>}
      ,{"cp":<cp_code2>,"val":<val_2>}
      ,{"cp":<cp_coden>,"val":<val_n>}]
,"ts":<UTC UNIX timestamp in seconds>
}

# JSON object template of a MCS->SMS DownloadNodeFirmware message.
{"node_list":
    [{"node_address":"<node_address>"}
    ,{"node_address":<node_address1>}
    ,{"node_address":<node_address2>}
    ,{"node_address":<node_addressn>}]
,"product_firmware_version":
    {"version":<version>
    ,"file_name":<file_name>
    ,"content":<content>
    ,"md5hash":<md5hash>
}}
```

Figure 3.18: Two samples of JSON object templates that form the payloads of an outgoing *LuminaireEnergyReport* message to the MCS, and an incoming *DownloadNodeFirmware* message from the MCS.

- *hashlib* – Interface to many secure cryptographic hash and message digest algorithms [**68**]. The commonly used MD5 hash algorithm is employed to verify the integrity of node firmware binary files received from the MCS (the hash generated at the MCS is transmitted in the *md5hash* member of the second JSON object template in Figure 3.18).

- *logging* – Event logging service for applications, libraries, and system services [**69**]. Used to log events across custom-developed and third-party processes running within the SMS. Multiple levels of verbosity with customizable filters, handlers to direct log entries to particular files, and log text formatters make it a very flexible and useful tool (from high-bandwidth debug logging during development to lower-bandwidth event logging in the production version), and

the number of log files and their sizes can be limited (i.e. a rolling buffer) to ensure the host system does not run out of disk space.

The Presentation layer is comprised of a built-in Admin GUI and Authorization plugin that optionally installs with the Django package [**53**] (to be described shortly). This is used to implement the SMS Administration Web Application, configured to enable an authorized ILS Administrator to check the status of task queues and worker processes, set SMS operating parameters, and view / modify (with caution) node attributes in the WFN Node State Database. Figure 3.19 gives a screenshot of a view provided by the SMS Administration Web Application.

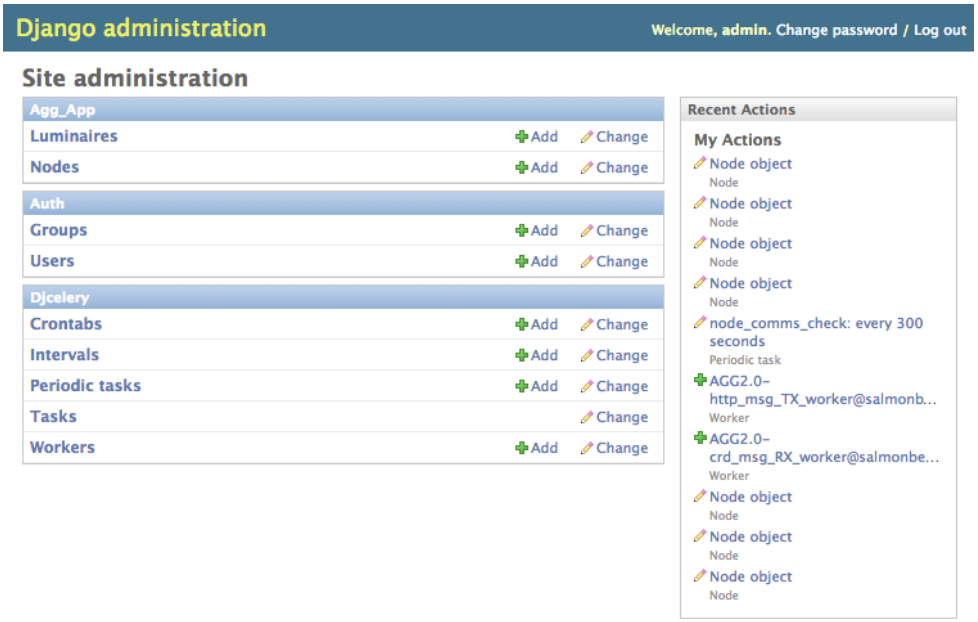

Figure 3.19: A screenshot of the main page of the SMS Administration Web Application, implemented using Django's built-in Admin GUI and Authorization plugins [**53**]. It allows for easy inspection and adjustment of WFN Node State Database entries, and periodic and event-driven Celery task and worker process status and configurations.

The Technical Services layer provides a number of services that either run in the background during normal SMS operation, or are invoked as needed, to support the Application and Presentation layers:

- Django [**53**] – A web application framework loosely following the Model-View-Controller (MVC) paradigm. Django was chosen because the set of available features supported many of the SMS use cases and requirements.

These included high stability under large traffic loads (spikes of over 50,000 hits per second) and the ability to scale resources as needed, user/machine authentication, data model and database access management, and an option to include a production-ready and customizable web application administration interface. Django was designed to allow for quick development of web applications, and its Python API makes integration with other Python-based applications, system services, and modules (libraries) easy. A custom SMS application was built that used the Django framework to manage the WFN Node State Data Model, as well as to provide the URL-based linking of incoming HTTP(S) requests to worker processes to handle them, via an internal message broker service and a task queuing service (Celery). As mentioned, the SMS Administration Web Application is hosted using Django's built-in Admin GUI and Authorization plugin.

- SQLite [**64**] – A lightweight database implementing the WFN Node State Database as defined by the Django data model. Django provides an object-relational mapper and data model description and query syntax in Python that is agnostic to the underlying database technology. SQLite was chosen for its small code footprint, efficient use of memory, disk space and bandwidth, and high reliability and lack of database configuration or administration requirements. Django also supports MySQL [**70**], PostgreSQL [**71**], and Oracle [**72**] database backends. The code snippet of Figure 3.20 shows part of the *Node* class of the WFN Node State Data Model implemented in SQLite, as declared by Django's data model description syntax.

```
class Node(models.Model):

    node_address = models.CharField(max_length=25, blank=True)
    ipv4 = models.GenericIPAddressField(protocol='IPv4', blank=True, null=True)
    mac_address = models.CharField(max_length=25, blank=True)
...
    # Wireless node type -- 0=coordinator/1=router/2=end device
    node_type = models.PositiveIntegerField(blank=True, null=True)
    zone_id = models.IntegerField(blank=True, null=True)
...
    # Timestamp of last heartbeat/firmware report from this node
    last_heartbeat_timestamp = models.DateTimeField(blank=True, null=True)
...
    # Current node state -- Not Communicating = 0, Communicating = 1
    node_state = models.IntegerField(blank=True, null=True)
```

Figure 3.20: Code snippet, using Django's data model description syntax, of the *Node* class of the WFN Node State Data Model.

- South [**73**] – A database schema migration tool. It is used to migrate the WFN Node Database whenever the WFN Node Data Model is changed in a quick and automated way, maintaining the integrity of existing data. South supports migration for all the database backends supported by Django, thus providing a vehicle with which to change to another backend in the future if required.

- Celery [**74**] – A distributed task queuing service. Celery's focus is on task queuing for real-time processing, but also supports task scheduling (either periodic or single shot) via the Celerybeat Scheduler module. Dedicated worker processes run concurrently and monitor queues for messages requesting processing of new tasks. In the SMS, there are four dedicated task queues, one for each of four workers. Shown in Figure 3.21, the task queues are *TCPMessageInQueue*, *TCPMessageOutQueue*, *HTTPMessageInQueue*, *HTTPMessageOutQueue*, and the *PeriodicTaskQueue*. The associated workers, respectively, are *TCPMessageProcessorWorker*, *TCPMessageSenderWorker*, *HTTPMessageProcessorWorker*, *HTTPMessageSenderWorker*, and *ScheduledProcessingWorker*.

- RabbitMQ [**75**] – RabbitMQ is an application message broker service that is an open source implementation of the Advanced Message Queuing Protocol (AMQP) [**76**]. It is designed to allow tradeoffs between reliability and performance, with functionalities including message persistence, delivery acknowledgement, and high availability. Celery depends upon a message

transport service to send and receive messages to and from its queues; in the SMS, this service is provided by RabbitMQ. RabbitMQ provides exchanges of various types which messages are routed through before being delivered to queues. In the SMS, three main "channels" are implemented with RabbitMQ providing the message routing between higher-level subsystems and Celery queues and workers. The first two channels (depicted in Figure 3.21) support the flows of execution that implement the main real-time messaging functionalities of the SMS (MCS→WFN messaging and WFN→MCS messaging). The third channel supports periodic task dispatch and message generation originating within the SMS. RabbitMQ offers the option to persist all undelivered messages on disk, even if the service crashes and is restarted[11]; this was used to implement the Guaranteed Delivery pattern at the interfaces between subsystems / worker processes within the SMS (represented by the input "lollipop" and output "socket" symbols in Figure 3.21).

- nginx [**77**] – An open source high-performance web server with simple configuration and low resource requirements. Nginx has an event-driven (asynchronous) architecture for handling requests, rather than using threads, which means the amount of memory it can use under load is limited and predictable, important factors given the low-resourced SBC hardware and Linux operating system the SMS was implemented on. The nginx web server is represented by the HTTP Server subsystem in Figure 3.21. It directs all traffic arriving on the *operationsHttpPort* and *smsAdminHttpPort* (Figure 2.3) to Django via the uWSGI layer.

- uWSGI [**78**] – A Python Web Server Gateway Interface. In the SMS, all HTTP requests for non-static content pass through the uWSGI middleware, which provides a two-way interface between the nginx web server and the Django web application framework. A maximum number of concurrent uWSGI processes

[11] This has an upper limit on the number of messages that can be held in persistent memory; for the SMS this was implemented as a ring buffer of size 25 MB.

that can be spawned and run in parallel is one configuration option used to tune HTTP request handling performance.

- Secure Sockets Layer (SSL) [**79**] – A widely used cryptographic protocol for secure communications over computer networks.

- A lightweight multi-threaded Python TCP server, implemented using the *threading* and *sockets* libraries. It receives TCP/IP messages from all CG devices within the same LAN in the facility as the SMS. It is represented by the *TCP Server* subsystem in Figure 3.21.

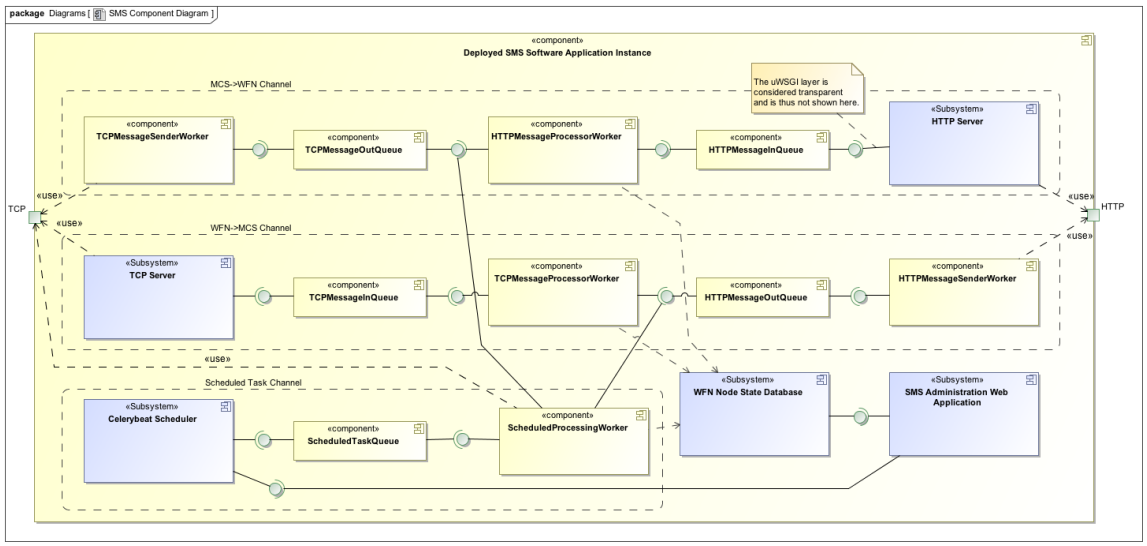

Figure 3.21: UML component diagram showing the main application layer components of the SMS for a single minimal deployment. Execution flows are grouped into logically parallel channels, one each for MCS→WFN and WFN→MCS communications processing, and one for handling scheduled tasks.

3.6.2 Tasks and Worker Processes

Execution of Celery worker processes is triggered by messages appearing at the head of any of the FIFO task queues shown in Figure 3.21. These messages can originate at the HTTP Server, TCP Server, or Celerybeat Scheduler, and are routed to the queues (with guaranteed delivery via message persistence) by RabbitMQ. It is unknown to the caller which worker will consume a given message; this is specified by telling worker processes which queue(s) to listen on when spawned at system start-up, e.g.

```
$ python manage.py celery -A sms_django worker -Q http_msg_tx_q --
hostname=http_msg_sender_worker@%h --loglevel=INFO --uid www-data --gid www-data
```

The code snippet of Figure 3.22 shows sample syntax of the Celery task `send_msg_to_mcs()` execution being requested by way of placing a message on the queue `http_msg_tx_q` (*HTTPMessageOutQueue*).

```
# If the event is to be reported to the MCS...
    if checksum_validated == True and report_to_MCS == True:
    # Put message in http_msg_tx_q for sending to MCS by task send_msg_to_mcs
    send_msg_to_mcs.apply_async((system_timestamp,
                                 dest_url,
                                 dest_port,
                                 dest_path,
                                 headers,
                                 rest_operation,
                                 json.dumps(payload)),
                                 queue = 'http_msg_tx_q')
```

Figure 3.22: Execution of the task `send_msg_to_mcs()` is requested via the Celery `apply_async()`call, which puts a message on the `http_msg_tx_q` (for an *HTTPMessageSenderWorker* process to consume).

The channels of Figure 3.21 are implemented by way of the caller specifying the target queue as well as the type of task that should be executed by the worker consuming the message from the queue. It is possible to have multiple concurrent worker processes listening and consuming from the same queue and executing the same or different task type(s). For example, multiple *TCPMessageProcessorWorker* instances could be created to increase throughput on the WFN→MCS channel, if system resources allow. The number of concurrent listening workers was limited to one per queue during development and testing of the SMS, thus giving a total of 5 possible concurrently running worker processes (performance tuning will be discussed in Chapter 4). Likewise, different sources – even tasks executing on other workers – can put tasks into the same queue. For instance, the *TCPMessageOutQueue* can receive tasks from a number of possible tasks being executed by the *HTTPMessageProcessorWorker*, or from the `system_time_sync()` task being executed by the *ScheduledProcessingWorker*. Table 3.2 gives the possible Celery task to worker process mapping for this release of the SMS. What follows is a grouping of these tasks by channel, illustrating the individual function of each, and how they work together to implement the channels.

Celery Worker Process	Listening to Queue	Assignable Tasks
TCPMessageProcessorWorker	*TCPMessageInQueue*	`process_tcp_message()`
TCPMessageSenderWorker	*TCPMessageOutQueue*	`send_command_to_cg()`

HTTPMessageProcessorWorker	*HTTPMessageInQueue*	`set_node_operating_parameter()`
		`assign_nodes_to_zone()`
		`assign_schedule_to_zone()`
		`commit_node_firmware()`
		`activate_node_firmware()`

HTTPMessageSenderWorker	*HTTPMessageOutQueue*	`send_msg_to_mcs()`

ScheduledProcessingWorker		`missed_heartbeat_check()`
		`system_time_sync()`
		`push_firmware_to_cg()`

Table 3.2: Mapping of tasks that can be assigned to Celery worker processes for execution, and the queues the worker processes listen on to receive the execution requests and input parameters.

WFN→MCS Channel

CgTcpDatagrams received from facility CGs by the multithreaded TCP Server are immediately stripped of headers and their hex string payloads are passed as input parameters with a `process_tcp_msg()` task put into the *TCPMessageInQueue*.

`process_tcp_message()`

During normal operation, *TCPMessageProcessorWorker* is the most active of all the worker processes, executing the `process_tcp_msg()` task on every node event report that is passed upward by the CG of every facility WFN. A description of execution flow of `process_tcp_msg()`is given by the activity diagram of Figure 3.23. Beginning at the top left, the task receives the hex string payload from the *CgTcpDatagram*. The beginning of the payload is parsed, length and checksum validation are performed, and the message type is identified by parsing out the subsequent *eventCode* in the string. The remainder of the message attributes – with fixed positions and character lengths within the string depending on the message type – are then parsed and processed. This remaining execution flow of `process_tcp_msg()` will vary depending on the *eventCode*, and sometimes, the last recorded state of the originating node within the WFN Node State Database as well. The code snippet of Figure 3.22 is the implementation of the last action of the activity diagram shown in Figure 3.23 before termination, putting the `send_msg_to_mcs()` task on the *HTTPMessageOutQueue* with an *SmsRestRequest* object payload.

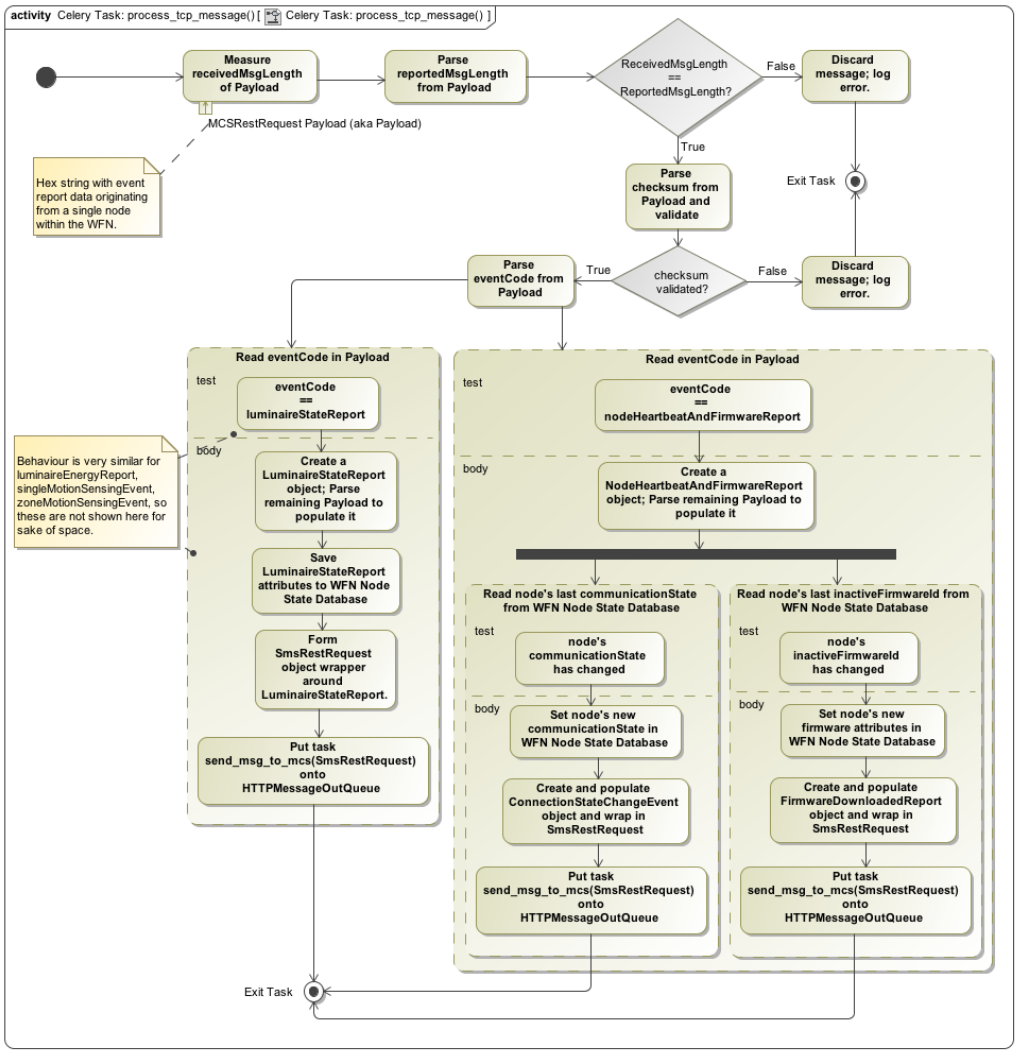

Figure 3.23: UML activity diagram describing the execution algorithm of the `process_tcp_msg()` task, which is consumed from the *TCPMessageInQueue* and carried out by a *TCPMessageProcessorWorker* process.

`send_msg_to_mcs()`

The `send_msg_to_mcs()` task consumes *SmsRestRequest* objects with their JSON payloads specific to the report type and content from the *HTTPMessageOutQueue* and uses the Python `requests` module to carry out the HTTPS REST transaction – encrypting the payload and attaching the SSL certificate – with the MCS. If the MCS is unavailable (either an internal error or network connectivity issue), the HTTP request will retry up to a specified `max_retries` (default is 5); if it ultimately fails, `send_msg_to_mcs()` will abort itself and be returned to the top of the *HTTPMessageOutQueue* (preserving temporal ordering of messages). This will result in the task being executed again immediately, and repeatedly until successful delivery to the

MCS. This implements the Guaranteed Delivery pattern (and requirement) on the SMS→MCS interface, ensuring that no important facility reporting data is lost (i.e energy use), subject to the maximum number of tasks that can be put into the *HTTPMessageOutQueue* before running out of disk space.[12]

MCS→WFN Channel

REST requests received from the MCS via the HTTP Server are first handled by Django's URL parser and view mechanism, which completes the REST transaction (receiving the payload and returning an HTTP response code to the MCS). The only GET request implemented at this time is *GetNodeAttributeValue*. In this case Django's view mechanism handles the request directly by querying the WFN Node State Database for the last known attribute value for the node in question, and returns the value along with the HTTP response.

For POST requests, based upon the URL, any one of the following tasks is put into the *HTTPMessageInQueue* with the unwrapped *McsRestRequest* object from the payload as an input parameter: `set_node_operating_parameter()`, `assign_nodes_to_zone()`, `assign_schedule_to_zone()`, `commit_node_firmware()`, or `activate_node_firmware()`.

All these tasks, with the exception of `commit_node_firmware()`, have similar behaviours that will not be described in detail here. Essentially, each is responsible for reforming the received *McsRestRequest* object into a proprietary position-based *command:value1,value2,…valueN* message format expected by the CG. This is followed by putting the `send_command_to_cg()` task into the *TCPMessageOutQueue* with the CG command string as an input parameter.

`send_command_to_cg()`

The task `send_command_to_cg()` is run on the *TCPMessageSenderWorker* and can be invoked by either a task running on the *HTTPMessageProcessorWorker* (any of `set_node_operating_parameter()`, `assign_nodes_to_zone()`,

[12] Alternatively, a Time-to-Live (TTL) attribute can be set for tasks on queues, so that they will be dropped after sitting on a queue beyond this expiry time.

`assign_schedule_to_zone()`, or `activate_node_firmware()`) or `system_time_sync()` running on the *ScheduledProcessingWorker*. It carries out all simple command-related TCP message transactions to facility CGs using the Python *sockets* module.

`commit_node_firmware()`

The `commit_node_firmware()`task takes the binary firmware file provided to it within the received *McsRestRequest.DownloadNodeFirmware* payload, performs an integrity check with the provided *fileHash*, queries the WFN Node State Database to return all nodes given in the *nodeList* parameter, saves the binary file to disk and points each node's *inactiveFirmwareFile* using the filename and path as saved (which is automatically timestamped with a YYYY-mm-dd filename suffix). The firmware file then resides on disk for the `push_firmware_to_cg()` task to be invoked later and begin the update of firmware on the target nodes.

Scheduled Task Channel

The Scheduled Task Channel is governed by the Celerybeat Scheduler, which acts similarly to a UNIX CRON service, entering tasks into the *ScheduledTaskQueue* at their scheduled execution times. Task schedules can be created and adjusted via the SMS Administrator Web Application.

`missed_heartbeat_check()`

The `missed_heartbeat_check()` task runs on a periodic schedule, specified by default to run every 5 minutes – the heartbeat check interval. Upon invocation, it reads the current system time and performs a lookup of all nodes in the WFN Node State Database. Then, for each node, it checks if the current time minus the node's existing *lastHeartbeatTimestamp* in the database is greater than two heartbeat check intervals. If so, the node's *communicationState* attribute is set to `False` (and saved back to the database) and a JSON object payload is constructed and wrapped in an *SmsRestRequest* object. Finally, a `send_msg_to_mcs`(*SmsRestRequest*) task is sent to the *HTTPMessageOutQueue*, to prompt the HTTPMessageSenderWorker to send an alert message to the MCS. Note, `missed_heartbeat_check()` can only set a node's *communicationState* attribute to `False` in the WFN Node State Database; as shown in

Figure 3.23, `process_tcp_msg()` is responsible for (re)setting *communicationState* to `True` upon reception of *NodeHeartbeatAndFirmwareReport* messages from the WFN.

`system_time_sync()`

The `system_time_sync()` task is scheduled to run once per day, at a low traffic time. The SMS is running the Network Time Protocol (NTP) daemon at all times, thus generally keeping its system time well synchronized with NTP time. `system_time_sync()`captures the current SMS date and time and puts a *SetNodeClocks* command for each facility CG into the *TCPMessageOutQueue* for delivery by the `send_command_to_cg()` task.

`push_firmware_to_cg()`

The `push_firmware_to_cg()` task is invoked by the Celerybeat Scheduler to execute just once, as configured by an ILS Administrator user via the SMS Administration Web Application. Used only as frequently as important firmware updates are issued, this would generally follow a `commit_node_firmware()`task being carried out, but would be scheduled to happen during a typically low WFN activity time (i.e. middle of the night when no occupants are likely around), to minimize the chances of creating network congestion and/or noticeable lag in luminaire responses to occupancy events. `push_firmware_to_cg()` directly uses the TCP port to initiate and carry out a binary file transfer to the CG using the Xmodem [**80**] protocol. The binary file to be transferred is retrieved from the entries in the WFN Node State Database for the affected node(s). Multiple nodes in a WFN will generally share the same firmware; the list of node addresses are provided to the CG as well during this transfer.

4 Testing and Performance

The ILS was successful in demonstrating functional integration of all its selected and developed hardware and software technologies, with very significant energy savings over the legacy lighting systems along with a notable improvement in illumination quality. The SMS subsystem also proved capable of keeping up with the demand of servicing incoming network traffic from the single facility CG managing 52 WFN luminaire nodes. Figure 4.1 depicts the physical layout of the pilot ILS field installation.

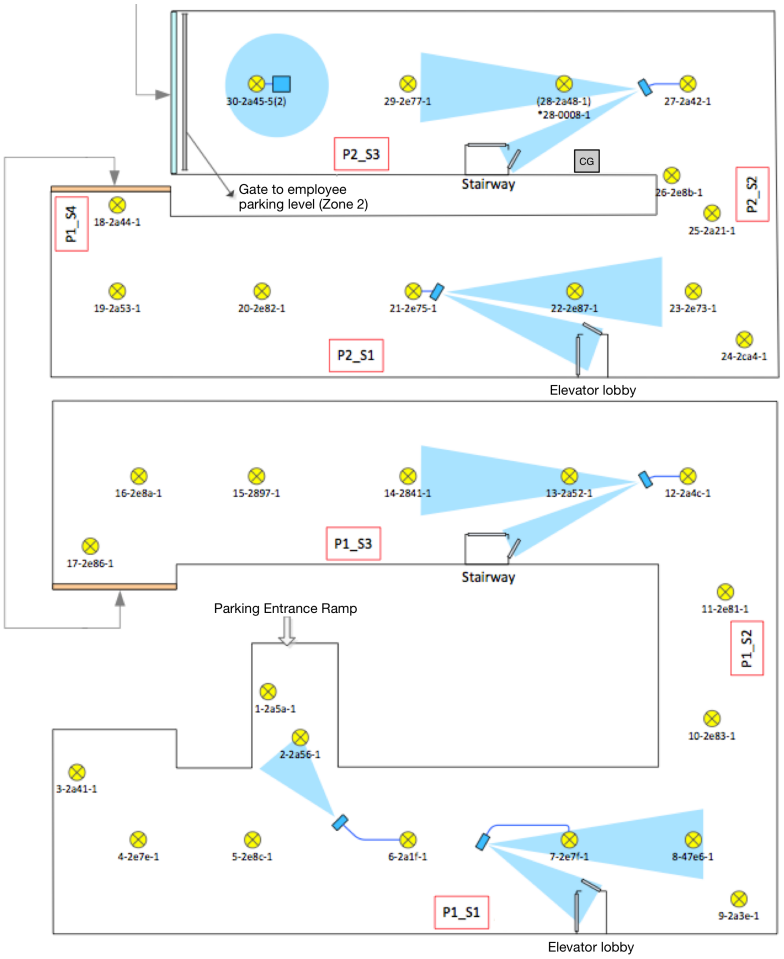

Figure 4.1: The physical layout of the ILS pilot scale installation. Two of the three levels of the spiral configuration underground parking facility are shown, with luminaires represented by yellow circles (each with an embedded PIR motion sensor facing downward). Lighting zones are indicated by the dashed suffix of each alphanumeric text code below each luminaire. Microwave motion sensors are represented by blue rectangles, their effective sensing arcs and ranges shown in blue shading. These are positioned to detect pedestrians as they enter the parking area from stairwells and elevator lobbies, outside of luminaire PIR motion sensor coverage, as well as moving vehicles that are cold (i.e. recently started, and thus not having a significant infrared heat signature). The single Coordinator-Gateway is represented by the grey box labeled "CG", located where a LAN connection was made available. The SMS is not shown, being located in the facility server room on another floor (H. Davis 2014).

4.1 ILS Performance

Figure 4.2 shows a screenshot of the Summary view of the web-based pMCS GUI, summarizing the key historical performance and activity telemetry of the ILS at the demonstration scale. As indicated in this screenshot, the prototype WFN consisted of one CG and four LED luminaires [**81**] (run at 112 W maximum output, 13 W minimum output) demonstrated energy savings of over 90% in a typical week of operation over the twelve 140 W MH HID luminaires it replaced. Figure 4.3 shows a screenshot of the Light Fixture Information view of the GUI (allowing a user to drill down to query historical activity, current operating parameter values, and fixed attributes of specific luminaires), which illustrates the time spent at *ocsLow* luminaire output level due to intelligent control.

Figure 4.4 shows a screenshot of the Energy Summary view of the web-based enterprise MCS GUI for the pilot scale ILS installation of 52 LED luminaires running at an average 82 W load, replacing all existing 89 MH HID luminaires.

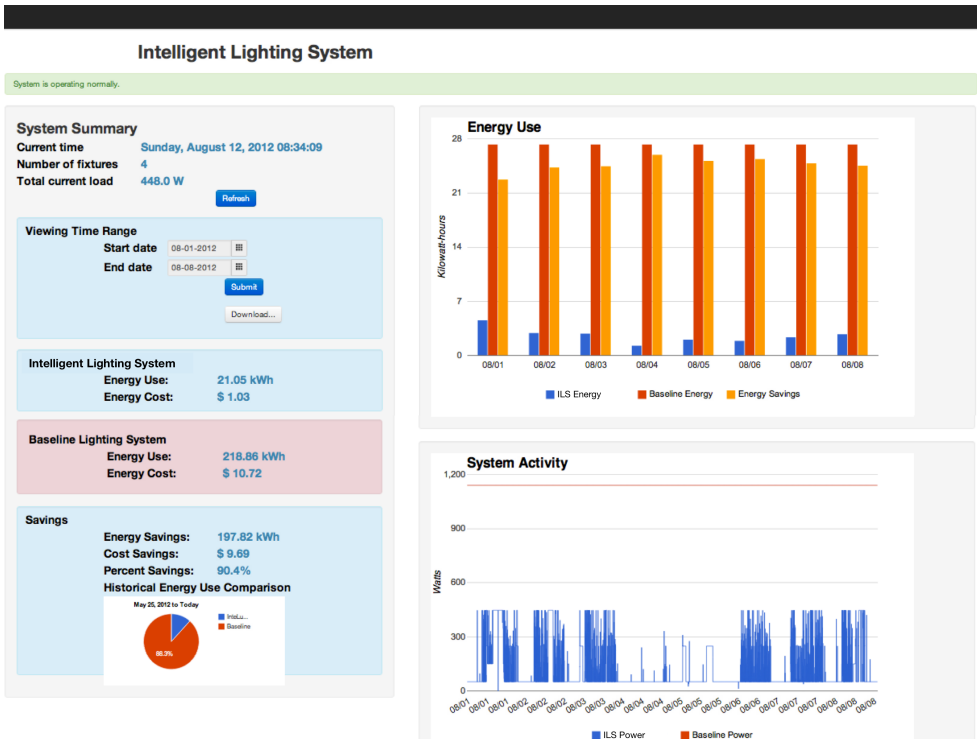

Figure 4.2: A screenshot of the summary view of the web-based GUI of the Prototype MCS, summarizing the key historical performance and activity metrics of the ILS. The System Activity pane shows aggregate luminaire load as it varies according to motion detection events (and scheduled operating policies).

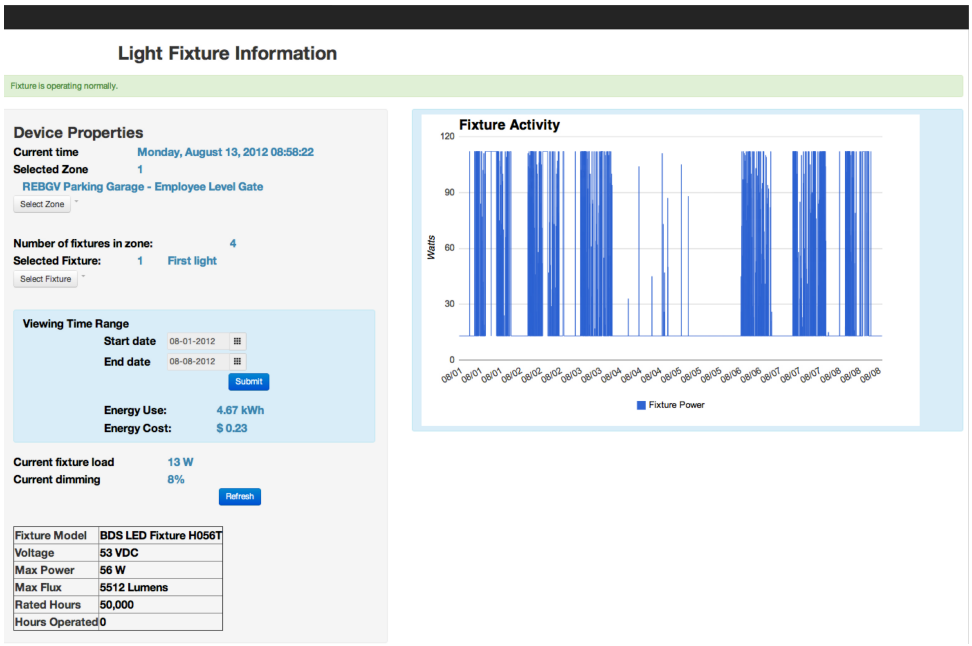

Figure 4.3: A screenshot of the fixture (luminaire) view of the web-based GUI of the pMCS, allowing the user to drill down to query historical activity, current operating state values, and fixed attributes of specific luminaires in the WFN. Activity for a single luminaire is shown, its load in Watts fluctuating with motion detection events.

Figure 4.4: A screenshot of the summary view of the web-based MCS GUI for the pilot ILS facility. The view defaults to present both the past 7 days of daily energy use and the past 24 hours of hourly energy use.

As summarized in Table 4.1, at the pilot scale illumination was improved by the higher luminous efficacy and spectrum coverage of the LED luminaires, while energy

savings related to the LED luminaire retrofit alone was approximately 65%. An additional 35% energy savings was attributed to the intelligent dimming of the ILS.

Legacy Lighting Installation		
Number of MH HID luminaires	89	
Measured load per luminaire (W)	90 - 198 (from random sample measurements of 9 luminaires)	
Average load per luminaire (W)	140	
Minimum illumination (FC)	0.1 (Code requires a minimum of 1 FC for emergency lighting)	
Maximum illumination (FC)	10.5	
Energy Use (kWh)	~2,100 per week	~109,150 per year
Intelligent Lighting System Pilot Installation		
Number of LED luminaires	52	
Rated load per luminaire (W)	90	
Average load per luminaire (W)	83	
Minimum illumination (FC)	2	
Maximum illumination (FC)	15	
Energy Use, LEDs only (kWh)	~727 per week	~37,808 per year
Savings (%)	~65%	
Energy Use, LEDs + ILS (kWh)	~200 per week	~10,400 per year
Savings (%)	90 - 95%	

Table 4.1: Performance summary of the LED lighting retrofit at the pilot demonstration site, as compared to the legacy lighting installation for LED retrofit only (~65% energy savings), and LED retrofit with ILS control (~90% energy savings).

4.2 SMS Performance

The SMS met its performance requirements, demonstrated by its ability to keep up with the processing of steadily arriving TCP traffic, at the pilot scale setup of 52 WFN luminaires managed by one CG, where TCP datagram arrival frequency generally did not exceed 10 per second. This was running with one worker process per task queue on both the WFN→MCS and MCS→WFN channels depicted in Figure 3.21. Task queues were monitored during testing; the number of pending tasks on the queues fluctuated (their rate of consumption being influenced by factors such as OS multitasking and system resource contention) but worker processes were always able to reduce the number of queue entries back to zero within 30 seconds. This delay in clearing out queues was considered acceptable, as data updates at the MCS were not required to be close to real-time (receiving events within a minute or two of their occurrence was considered adequate). The rate of arrival of TCP datagrams averaged 3 per second and peaked around 10 per second, and the SMS was able to keep up with performing its duties at this rate.

The SMS software stack was designed to give the capability of performance tuning. The main tunable parameter is the number of available Celery workers for handling

higher traffic loads. Specifically, it is expected that additional instances of *TCPMessageProcessorWorker* would be required as the number of WFN nodes in a facility grows[13]. This would be subject to the system resource limits of the host SBC, which laboratory testing confirmed. The Raspberry Pi SBC used as a SMS prototype platform was deemed to be inadequate as a SMS host for larger commercial-grade installations, as it began to fall behind at sustained TCP datagram arrival rates greater than 10 per second. If the rate that tasks arrive on the queues consistently exceeds the rate that they can be consumed by available workers, the situation is not sustainable. Increasing the number of available *TCPMessageProcessorWorker* instances did reduce the rate at which the SMS fell behind, but system RAM would become exhausted after 9 such spawned instances (all other worker types being single instances). Some gains could be made from tuning memory and processor usage of certain system processes, and/or stripping down the number of (unneeded) running services on the Linux operating system, but ultimately, other aspects of the architecture of the Raspberry Pi SBC would be the major limiting factors: 1) the 10/100 Ethernet controller communicates with the Broadcom SoC over the same USB 2.0 bus that the SD card drive is accessed from, creating ongoing bus contention (and related overhead) and thus reduced overall throughput, and 2) SD flash file systems have very slow read and write access compared to standard hard drives and solid-state drives. These design/cost tradeoffs in the Raspberry Pi are well known, and can lead to serious bottlenecks in pseudo real-time applications such as this [**82**].

It should be mentioned that it is also possible to tune the frequency of the periodic event reporting from WFN nodes and relieve the burden on the SMS (and indeed the entire WFN). For instance, *LuminaireEnergyReports* could be sent every 10 minutes instead of every 5 minutes, with the trade-off of lower granularity (but less) time series data collected at the MCS.

---

[13] Conversely, the frequency of HTTP requests on the SMS from either the MCS or the ILS Administrator Web Application (most commonly configuration updates or queries) is very low, and is thus considered a negligible factor.

5 Conclusions and Future Work

The Intelligent Lighting System development project detailed in this thesis aimed to provide a reliable, responsive, user-friendly, and highly efficient commercial lighting solution through the integration of a suite of key emerging technologies – high brightness white Light Emitting Diodes, low-power wireless networks, Single Board Computers, and cloud computing. The ILS was envisaged as a highly distributed system in order to be able to combine the unique advantages of these technologies. It was by its nature a messaging-based system of interacting and uniquely addressable systems, fitting well into the Internet of Things paradigm.

The ILS was successful in demonstrating functional integration of all the selected and custom developed hardware and software technologies. Its performance was characterized by a 90% or greater energy savings over the legacy lighting systems replaced in the underground parking garage setting, exceeding the savings of simply retrofitting with LED luminaires without intelligent control by 35%. This result was shown at both the demonstration scale, with 4 LED luminaires replacing 12 MH HID luminaires, and at a pilot scale with 52 LED luminaires replacing 89 MH HID luminaires. This replacement at an approximate ratio of three 90 W LED luminaires for every five 140 W MH HID luminaires also resulted in improved illuminance in the space when users were present, characterized by higher brightness and fuller spectrum light.

The Sensor Middleware System was a custom subsystem development that grew out of the need for scaling up the ILS with larger commercial lighting installations. It would provide the reliable two-way link between the facility Wireless Field Networks and the cloud-based Monitoring and Configuration System, as well as add new facility supervisory functionalities to improve overall ILS reliability and quality of service. It was designed and developed using proven open-source software packages and libraries – mostly Python-based – with its own software scalability in mind to support higher throughput. Message throughput and guaranteed system telemetry data delivery requirements drove the internal SMS software design to be message-based as well. To this end, it employed a number of common messaging-based design patterns including *Message Broker*, *Translator*, *Envelope Wrapper*, and *Guaranteed Delivery*.

The SMS was tested at the pilot scale described, with one Wireless Field Network containing all 52 luminaire nodes managed by one Coordinator-Gateway device. It demonstrated complete capability of handling incoming messages under all load conditions at this scale, running on the Linux-based Raspberry Pi Single Board Computer prototyping platform. Deploying the same SMS software suite on better resourced hardware platforms promises to enable this solution to scale very well for facility intelligent lighting installations consisting of dozens of Coordinator-Gateways, and potentially thousands of wireless luminaires.

Future work for the ILS would probably see further integration of the facility hardware to bring down overall system costs and complexity. The first step would likely be to combine the Sensor Middleware System and Coordinator-Gateway hardware platforms into one component consisting of a SBC host device and a wireless USB dongle with a NXP JN5148 32-bit System-on-Chip. The continually dropping cost of SBCs, and the availability of off-the-shelf JN5148 wireless dongles, indicates that this will be a lower cost solution than designing and manufacturing a custom circuit board and enclosure for standalone CG devices as in the current iteration. This would also mean removing a layer of hierarchy in the ILS, which would simplify data transmission between the WFN and the MCS. The SMS software design would require only minor modifications to accommodate this change.

Bibliography

[1] Natural Resources Canada, "Energy Use Data Handbook 1990 to 2009," Office of Energy Efficiency, ISSN 1715-3174, 2012.

[2] US Department of Energy, "2011 Buildings Energy Data Book," Building Technologies Program Energy Efficiency and Renewable Energy, Pacific Northwest National Laboratory, 2012.

[3] Government of Canada, "National Inventory Report 1990 to 2011: Greenhouse Gas Sources and Sinks in Canada," Environment Canada, Annual (Part 3) ISSN: 1910-7064, 2013.

[4] Illuminating Engineering Society. (2014) Illuminating Engineering Society. [Online]. http://www.ies.org/

[5] US Environmental Protection Agency. (2004) Lighting Technologies: A Guide to Energy-Efficient Illumination. [Online]. https://www.energystar.gov/ia/partners/promotions/change_light/downloads/Fact%20Sheet_Lighting%20Technologies.pdf

[6] US Department of Energy. (2013, August) High-Intensity Discharge Lighting Basics. [Online]. http://energy.gov/eere/energybasics/articles/high-intensity-discharge-lighting-basics

[7] US Department of Energy, "Solid-State Lighting Technology Fact Sheet," Energy Efficiency & Renewable Energy - Building Technologies Program, Pacific Northwest National Laboratory, 2013 PNNL-SA-94206, March.

[8] US Department of Energy. (2012, July) LED Lighting. [Online]. http://energy.gov/energysaver/articles/led-lighting

[9] US Department of Energy, "Lifetime of White LEDs," Energy Efficiency and Renewable Energy - Building Technologies Program, Pacific Northwest National Laboratory, PNNL-SA-50957 , 2009.

[10] Navigant Consulting, Inc. (2014) Smart Street Lighting - LEDs, Communications Equipment, and Network Management Software for Roadway and Highway Lighting: Global Market Analysis and Forecasts. [Online]. http://www.navigantresearch.com/research/smart-street-lighting

[11] Navigant Consulting, Inc. (2014) Outdoor and Parking Lighting Systems High-Pressure Sodium, Metal Halide, LED, Induction, and Fluorescent Lighting and Lighting Controls: Global Market Analysis and Forecasts. [Online]. http://www.navigantresearch.com/research/outdoor-and-parking-lighting-systems

[12] Navigant Consulting, Inc. (2013) High-Bay Lighting Energy Efficient Lighting and Lighting Controls for Warehouse, Industrial, Sporting, Retail, and Transportation Facilities. [Online]. http://www.navigantresearch.com/research/high-bay-lighting

[13] Navigant Consulting, Inc. (2014) Residential Energy Efficient Lighting and Lighting Controls. [Online]. http://www.navigantresearch.com/research/residential-energy-efficient-lighting-and-lighting-controls

[14] W Kastner, G Neugschwandtner, S Soucek, and HM Newmann, "Communication systems for building automation and control," in *Proceedings of the IEEE*, vol. 93, 2005, pp. 1178-1203.

[15] T Salsbury, "A survey of control technologies in the building automation industry," in *16th IFAC World Congress*, 2005.

[16] American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). BACnet standard. [Online]. www.BACnet.org

[17] Modbus Organization, Inc. (2015) Modbus Specifications and Implementation Guides. [Online]. http://www.modbus.org/specs.php

[18] LonMark International. LonMark Standard. [Online]. www.lonmark.org

[19] Echelon Corporation. (1999) Introduction to the LonWorks System. [Online]. http://downloads.echelon.com/support/documentation/manuals/general/078-0183-01A.pdf

[20] KNX Association. KNX Specifications. [Online]. http://www.knx.org/knx-en/knx/technology/specifications/index.php

[21] J Beutel, M Dyer, L Meier, M Ringwald, and L Thiele, "Next-generation deployment support for sensor networks," *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004), ACM Press, New York*, pp. 291–292, 2004.

[22] J.A Gutiérrez, "On the use of IEEE Std. 802.15. 4 to enable wireless sensor networks in building automation," *International Journal of Wireless Information Networks*, vol. 14, no. 4, pp. 295-301, 2007.

[23] E Pramsten, D Roberthson, J Eriksson, N Finne, and T Voigt, "Integrating Building Automation Systems and Wireless Sensor Networks," 2007.

[24] W Vandenberghe et al., "A system architecture for wireless building automation," *Proc. of IST Mobile & Wireless Communications Summit*, 2006.

[25] Navigant Consulting, Inc. (2013) Wireless Control Systems for Smart Buildings. [Online]. http://www.navigantresearch.com/research/wireless-control-systems-for-smart-buildings

[26] ZigBee Standards Organization. (2008, January) ZigBee Specification. [Online]. http://www.zigbee.org/non-menu-pages/zigbee-specification-download/

[27] NXP Laboratories. (2010, October) Jennic Wireless Microcontrollers. [Online]. http://www.jennic.com/support/user_guides/jn-ug-3041_jennet_user_guide

[28] Z-Wave Alliance. (2012, May) Z-Wave Alliance Standard. [Online]. www.z-wavealliance.org/

[29] EnOcean GmbH. EnOcean Wireless Standard. [Online]. www.enocean.com/en/enocean-wireless-standard/

[30] Bluetooth Technology Special Interest Group. (2010, June) Adopted Specifications. [Online]. https://www.bluetooth.org/en-us/specification/adopted-specifications

[31] C Perera, A Zaslavsky, P Christen, and D Georgakopoulos, "Context Aware Computing for The Internet of Things: A Survey," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 414-454, May 2013.

[32] Kevin Ashton. (2009, June) RFID Journal.

[33] V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle J. Höller, *From*

Machine-To-Machine to the Internet of Things, 9780124076846th ed.: Elsevier, 2014.

[34] Olivier Monnier. (2013, October) A smarter grid with the Internet of Things. [Online]. http://www.ti.com/lit/ml/slyb214/slyb214.pdf

[35] Raspberry Pi Foundation. (2015) Model B. [Online]. http://www.raspberrypi.org/products/model-b/

[36] Arduino. (2015) Arduino. [Online]. http://www.arduino.cc/

[37] BeagleBoard.org Foundation. (2014, November) BeagleBoard.org. [Online]. beagleboard.org/beagleboard

[38] DMP Electronics Inc. (2009) eBox-3300 & 3310 User's Manual. [Online]. http://www.compactpc.com.tw/ebox-3300.htm

[39] Advantech Co., Ltd. (2015) Energy Data Concentrator BEMG-4220. [Online]. http://www.advantech.eu/products/ENERGY_DATA_CONCENTRATORS/BEMG-4220/

[40] Infinite Automation Systems, Inc. (2013) The MangoES. [Online]. http://infiniteautomation.com/index.php/hardware

[41] A Mohammed. (2009, March) A history of cloud computing. [Online]. http://www.computerweekly.com/feature/A-history-of-cloud-computing

[42] Amazon Web Services, Inc. (2015) Amazon Web Services. [Online]. https://aws.amazon.com/

[43] Amazon Web Services, Inc. (2015) Amazon EC2. [Online]. https://aws.amazon.com/ec2/

[44] M Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," UC Berkeley Reliable Adaptive Distributed Systems Laboratory, Technical Report UCB/EECS-2009-28, 2009.

[45] Google, Inc. (2014, December) Google Cloud Platform. [Online]. https://cloud.google.com/appengine/docs

[46] Chantal Polsonetti. (2014, July) Automation World. [Online]. http://www.automationworld.com/cloud-computing/know-difference-between-iot-and-m2m

[47] NXP Laboratories. (2006, October) Jennic Wireless Microcontrollers. [Online]. http://www.jennic.com/support/support_files/jn-ug-3024_ieee_802154_wireless_networks

[48] NXP Laboratories. (2010, November) Jennic Wireless Microcontrollers. [Online]. http://www.jennic.com/support/datasheets/jn5148_module_datasheet

[49] NXP Laboratories. (2010, March) Jennic Support. [Online]. http://www.jennic.com/support/user_guides/

[50] ARM Ltd. (2014) Cortex-M3 Processor. [Online]. http://www.arm.com/products/processors/cortex-m/cortex-m3.php

[51] Beyond Limitations. (2012) Winged Gastonia Arch Area Flood Light. [Online]. http://www.bl-lights.com/view-led-product-details/239/10/8

[52] Microchip Technology Inc. (2014) PIC18F25K20 - 8-bit PIC Microcontroller.

[Online].
http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en026333

[53] Django Software Foundation. (2015) Django Documentation. [Online]. https://docs.djangoproject.com/en/1.7/

[54] The Graphite Project. (2012) Graphite Overview. [Online]. http://graphite.readthedocs.org/en/latest/overview.html

[55] G Hohpe and B Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, 1st ed.: Addison-Wesley Professional, 2003.

[56] S. Krakowiak. (2004) What's Middleware? [Online]. http://middleware.objectweb.org/

[57] Wikipedia.org. (2015) Message-Oriented Middleware (MOM). [Online]. https://en.wikipedia.org/wiki/Message-oriented_middleware

[58] Sun Microsystems. (2013, May) Java Message Service (JMS) 2.0 Specification. [Online]. http://www.oracle.com/technetwork/java/jms/index.html

[59] Microsoft Corporation. (2008, January) MSMQ Service. [Online]. https://technet.microsoft.com/en-us/library/cc773676(v=ws.10).aspx

[60] Microsoft Corporation. (2015) TCP/IP Protocol Architecture. [Online]. https://technet.microsoft.com/en-ca/library/cc958821.aspx

[61] E Gamma, R Helm, R Johnson, and J Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, 1st ed.: Addison-Wesley Professional, 1994.

[62] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, 1st ed.: Addison-Wesley Professional, 1993.

[63] Wikipedia.org. (2015) Representational State Transfer (REST). [Online]. https://en.wikipedia.org/wiki/Representational_state_transfer

[64] SQLite Project. SQLite Home Page. [Online]. https://sqlite.org/

[65] Python Software Foundation. (2015) Welcome to Python. [Online]. https://www.python.org/

[66] Kenneth Reitz. (2015) Requests - HTTP for Humans. [Online]. http://docs.python-requests.org/en/latest/

[67] EMCA International, "The JSON Data Interchange Format," Standard 2013. [Online]. http://json.org/

[68] Python Software Foundation. (2015, January) hashlib — Secure hashes and message digests. [Online]. https://docs.python.org/2/library/hashlib.html

[69] Python Software Foundation. (2015, January) logging - Logging facility for Python. [Online]. https://docs.python.org/2/library/logging.html

[70] MySQL.com. (2015) MySQL Reference Manual. [Online]. https://dev.mysql.com/doc/

[71] The PostgreSQL Global Development Group. (2015) PostgreSQL - The World's Most Advanced Open Source Database. [Online]. http://www.postgresql.org/

[72] Oracle. (2014) Introduction to the Oracle Database. [Online]. http://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm

[73] Andrew Godwin. (2010) About South. [Online].

http://south.readthedocs.org/en/latest/about.html

[74] Ask Solem & Contributors. (2014) Celery - Distributed Task Queue. [Online]. http://celery.readthedocs.org/en/latest/index.html

[75] Pivotal Software, Inc. (2015) RabbitMQ - Messaging that just works. [Online]. http://www.rabbitmq.com/

[76] OASIS. (2015) AMQP. [Online]. http://www.amqp.org/

[77] Nginx.org. (2014, February) Nginx Community. [Online]. http://wiki.nginx.org/Main

[78] uWSGI. (2014) The uWSGI project. [Online]. https://uwsgi-docs.readthedocs.org/en/latest/index.html

[79] Wikipedia.org. (2015, February) Transport Layer Security. [Online]. https://en.wikipedia.org/wiki/Transport_Layer_Security

[80] Wikipedia.org. (2015, January) XMODEM. [Online]. https://en.wikipedia.org/wiki/XMODEM

[81] PHC Northwest, Inc. (2014, November) PHC LED Lighting - High Bay Lighting.

[82] Raspbian.org. [Online]. http://www.raspbian.org/RaspberryPiSDPerformance