Distributed Energy Monitoring and Control Application for Intelligent Buildings

by

Mark Kevitt

A dissertation submitted to the University of Dublin, in partial fulfilment of the requirements for the degree of Master of Science in Computer Science

University of Dublin

2010

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Mark Kevitt

05/09/2010

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Mark Kevitt

05/09/2010

Abstract

Modern society has seen significant advances in the software industry, of particular interest in this thesis are advances in distributed computing with a focus on software to improve energy conservation in buildings. Computing offers opportunities to further improve the intelligence of a building, to enhance the quality of its occupancy and also to save on energy consumption. The design of intelligence in buildings present challenges to software engineering, more specifically with the difficulty of developing building system monitoring, management systems and their interoperability.

This thesis is focused at the design, development and evaluation of a distributed energy monitoring and control application (DEMAC) purposefully for use in intelligent building systems focusing on energy consumption reduction. A primary aim of this thesis is to develop the application to be scalable for use in large buildings and also for multiple geographically located buildings. A secondary aim of this thesis is to develop an application which will facilitate the interoperability of smart building systems to enable different network intelligent building systems to inter communicate.

Current research in this area is explored to get in tune with recent work that other researchers have pursued and to build on the strengths of their contributions in this field. Research is also conducted in the current technological trends in distributed software systems for generic software development and also in the field of intelligent building monitoring and control applications. This research is evaluated with the intention of deciding on the best approach to pursue with the development of the application.

Intelligent (or smart) building systems is examined with an over view of one system which is used in this application design and execution. The application requirements, architecture design, implementation and evaluation are explored to give an overview of each element and the merits to their respective design. Where one component follows another, the reasons for their selection are given and why the succeeding component logically follows its predecessor. The critical evaluation of the application is given with respect to its requirements and what benefits and shortcomings it has. Lastly any further work is outlined to assist with continuity of research in this area.

Acknowledgements

I would like express my gratitude for the continued support and understanding of my wife Caroline and also to my newly born daughter Cara who both give me inspiration and vigour for the pursuit of my dreams.

Contents

Table of Tables

Table of Figures

2 Introduction

This thesis is concerned with the design and development of a distributed energy monitoring and control (DEMAC) application for use with multiple networked buildings and the intelligent systems contained therein. The purpose of the application is threefold, firstly to facilitate the aggregation and dissemination of energy usage data of the buildings, secondly to facilitate a more informed control of the buildings' facilities to improve the quality of experience for the buildings' occupants and thirdly to further improve the buildings' intelligence through the improvement of automation system interoperability.

Research into intelligent building systems and current distributed computing technology is explored to ensure that this thesis builds upon existing research in this field and that the most suitable technology is employed for the application design.

2.1 Energy Consumption and monitoring

According to the United Nations Environment Programme for Sustainable Buildings and Climate Control(UNEP) – Sustainable Buildings and Climate Change- "The building sector contributes up to 30% of global annual green house gas emissions and consumes up to 40% of all energy"[28]. The improvement of Intelligence in buildings has the potential to save a considerable amount of energy and money for their owners. There are several ways in which to advocate the improvements in the use of energy in buildings, such as switching off lights, heating & cooling while zones within a building is unoccupied, the roles of the builders owners, facilities management and its occupants can each contribute to its efficient use. A system which informs each of these parties about energy use and which also learns and adapts to its environment has the potential to make energy efficiency change throughout the lifecycle of the building [5].

2.2 Intelligent Buildings

Buildings are constructed to create a refuge from the external environment in order to improve the comfort and wellbeing of people. In commercial buildings the productivity, morale and satisfaction of workers and customers alike is being optimised. The building must perform these objectives while minimising its energy consumption, minimising operating cost and extending its lifetime. Intelligent buildings provide 'qualities that create a productive and efficient environment such as functionality, security and safety; thermal, acoustic air-quality and visual comfort' [35].

There are certain degrees of intelligent buildings, a high degree of intellect in a building include the permission of complex systems to integrate and communicate and lessen the amount of human management intervention. A lower degree of intellect in a building would not permit the interaction and communication of systems but have many systems which operate independently. An example of high degree Intelligence in buildings includes the necessity of the emergency system overriding the security system in life threatening situations. On September 12 2001 when terrorists crashed a plane in to the Pentagon in the US the intelligent integrated building system facilitated the shut-down of ventilation fans to prevent smoke movement throughout sections of the burning building and also to starve the fire of oxygen [33]. The data acts as a basis on which a controlling program makes decisions and changes to the systems with a goal of optimising energy consumption and building operation costs. Other intelligence includes the aggregation of energy consumption.

In the case of large corporations with many geo-located buildings and a mobile workforce there is the need for a distributed network of intelligent buildings. Not all buildings and systems have the level of sophistication of the Pentagon. Remote access for facility engineers is a necessity; each engineer may be responsible for multiple buildings. The benefits for facility engineers with remote management access include a reduction of their need to travel to individual buildings. The creation of an intelligent building system with autonomous intellect which learns from usage trends and which has embedded system interoperability has the potential to improve on energy efficiency and aforementioned building qualities.

This thesis will give some background to intelligent building systems in the State of the Art chapter before examining recent developments in standards which aid the interoperability of intelligent building systems. The technologies which are used to develop applications and systems for distributed software are also explored with some examples of their use.

In the Existing Networked Energy Management Research chapter recent research on systems and applications which were developed purposefully to aid interoperability of smart building systems are examined with their benefits and contributions to this topic. These benefits will then be used as a foundation to the design of DEMAC.

The Application Solution Design chapter will outline the requirements for DEMAC and discuss the design from a high level. The chapter will also descend through subsequent lower level designs through to the component level architecture. Opposing design solutions are examined and their merits are compared against the requirements.

Opposing design solutions are discussed and the reason for the final DEMAC architecture explored in detail in the Implementation chapter. The main components which comprise the application are discussed with an overview of their responsibilities. This chapter also outlines the actual application from the server and client perspectives.

This thesis will assess the application under the headings of feasibility, performance and scalability in the evaluation chapter with conclusions and further work taking the thesis to its closure.

3 State of the Art

3.1 Description of an Intelligent Building System

In this chapter a brief description of what components contribute to an intelligence buildings system will be given. In addition the current technological trends in intelligent buildings are outlined. The research efforts of others for remote user management and monitoring of smart buildings are evaluated critically. The differing protocols which smart building systems / device communications are also investigated. The feasibility of these protocols and some recent development in distributed computing, namely Web Services and related technical architecture is examined to identify the benefits and merits of their use in DEMACs solution architecture.

Intelligent buildings are an aggregation of smart systems. A smart system incorporates sets of actuator devices and environmental sensors which when used together are programmed to respond to stimuli. Examples of building control systems include: heating, ventilation and air conditioning (HVAC), elevators, laboratory equipment, life/safety systems, access control, intruder detection, A/V event management, CCTV monitoring, and many others [23].

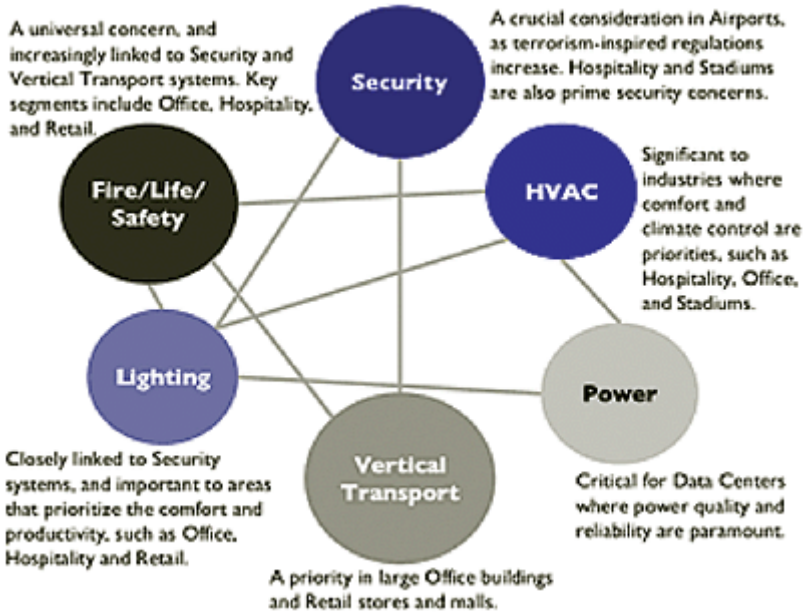

Figure 3-1Intelligent Building Systems

(www.greenintelligentbuildings.com, 2005)

There are discrete systems which operate in a closed circuit, e.g. an elevator system responds to buttons to 'call' a lift and to move to a designated floor. Legacy devices would be electronic but simulated and controlled by electrical operation, e.g. lift call buttons. More modern devices enabled by sensor and actuator technology can be 'soft' in their operation and have more logic to their operation. Intelligent buildings offer more soft interoperable systems which provide greater integrated building facility management. Over the last decade more open standards and protocols have emerged which improve facility management and the interoperation of different manufactured systems and devices.

Historically in building systems, different protocols existed for each different system, the motivation behind each protocol differed and no single protocol offered a multiple system integration solution. Domestic automation protocols were intended for the house with only a few devices and offered little use in the commercial market.

There were some commercial protocols with good characteristics which were feasible for their adoption as a system integration protocol. The proliferation and acceptance of international standards has assisted with the extension and adoption of such legacy protocols and for their use in interoperable systems. The growth of software comprehension by the populace and the desire for smarter integrated systems has fuelled the demand for integrated systems further. In the home, entertainment systems now interoperate with other domestic media devices and the Internet. Further advances are being pursued with white goods interoperation and internetworking.

In the commercial arena the focus was on issues such as security system integration with HVAC systems and other building specific systems. Only recently with the convergence of domestic, commercial and other communicating computing devices on packet switched networks has the desire for complete device interoperability been so strong. Energy consumption has been increasing in line with device proliferation; the need for energy monitoring has also been on the increase.

The blur of both domestic and commercial device roles and the widespread use of the Internet at home and in work has unified the protocol requirements of both domestic and commercial domains.

3.2 Interoperability Protocols for Intelligent Buildings

Over the past few decades commercial device protocols for Heating ventilation and control (HVAC) and other building systems have evolved slowly and mostly been manufacturer driven. Owners of buildings and multiple building systems have demanded interoperability in their buildings. This demand has pushed manufacturers to be more open in their use of protocols and as a result open standards (see 2.2.1 and 2.2.2) came about. In the home devices sector, customers also demanded interoperability of their home devices (e.g. media players and TV's). As a result open standards have evolved which to allow devices to interoperate, e.g. UpNP (see 3.2.3 UpNP).

3.2.1 BACNet

Building Automation and Control network (BACNet) is the common term used to describe a non-proprietary open protocol designed for building automation and control systems. It was specified for building systems including heating ventilation and air conditioning (HVAC) lighting systems and fire systems. BACNet is an ANSVASHRAE Standard 135-1995, adopted and supported by the American National Standards Institute (ANSI) and the American Society of Heating Refrigeration and Air-conditioning Engineers (ASHRAE) [12]. The purpose for this standard was to facilitate the interoperability of systems in the building control industry. BACNet is an object oriented protocol with the devices to be controlled represented by objects and properties. BACNet fits in to the transport layer of the OSI stack. BACNet has 23 virtual objects that represent as much building functionality as is typically needed [33].

3.2.2 LonTalk

LonTalk is an open communication standards for BAS developed by Echelon [33], it originated in the transportation and utilities industry but is currently in widespread use in BAS. The standard is adopted on the LonWorks platform and functions by use of a dedicated Neuron Chip which interfaces with control devices and communicates across multiple network layer protocols (of note - excluding RS485) which are controlled and monitored via an iLon 100 web server.

Both BACNet and LonTalk are considered the leading industrial BAS protocols for use in interoperable and multi vendor systems. BACNet has an advantage over LonTalk since it communicates over RS485 and has a protocol embedded prioritising mechanism to simplify engineering BAS solutions for competing control messages [33].LonTalk implements Networked Variables (NV) which are on a par with BACNet objects.

3.2.3 UPnP

On the smaller scale of open communication standards for smart buildings is UPnP. UPnP was originally designed for small networks such as those in domestic buildings by the UPnP forum which are made up of industry leaders in consumer electronics. Its architecture sits on the application layer of the networking stack. For each UPnP device, interoperability functions are available which aid intercommunication. Functions such as IP addressing, device and service discovery, description, control, events and presentation are included. UPnP relies on Simple Device Discovery Protocol (SDDP) and Simple Object Access Protocol (SOAP) to interact with devices as objects and to discover objects. The User Datagram Protocol (UDP) is used for discovery, multicast messages are sent across the network to discover and advertise devices and services. The message format uses XML. The UPnP forum has devised Device Control Protocols (DCP) for home automation (HVAC, lighting) and Audio \ Visual (media servers) to name a few. Unfortunately UPnP was not designed for the Internet and the discovery service sends multicast messages which are not suitable for large networks [16]. There are means to circumvent this, one such as the implementation of an OSGi gateway [11]. UPnP on its own fails to offer a solution for building automation control

on a large scale. The developments of web services which wrap UPnP offer a possible solution. Work is under way for Device Profile for Web Services (DPWS) or UPnP 2.0 to offer a scalable solution. OSGi is often used in centralised, embedded equipment, it can allow UPnP to be a scalable solution in BAS but it requires considerable expertise [36] which would rule this as a cost effective solution.

The UPnP architecture and its underlying protocols lend towards internetworking but with a reduction in multicasts and some access control over discoveries and device access. An Internet Gateway Device (IDG) which maintains a list of local networked devices and an UPnP bridge to focus on the control of local devices would allow for the connection of a LAN to a WAN [15].

3.2.4 Zigbee

ZigBee is a wireless networking standard which builds upon the IEEE 802.15.4 standard for low cost, low rate for personal area networks. ZigBee can be implemented in mesh networks and the IEEE specification focuses on the lower physical and data link layer. The ZigBee Alliance is somewhat similar to that of UPnP in that it is an association of companies working together to develop standards (and products) for sensor networks for domestic applications [3]. The contrast between ZigBee and UPnP is that ZigBee is focused on much smaller devices provides a multi-hop network and supports limited mobility of nodes [32].

The ZigBee platform has a network layer and an application layer. The network layer (NWK) is in charge of organising and providing routing over a multi-hop network and the Application Layer (APL) intends to provide a framework for distributed application development and communication. The ZigBee Device Objects (ZDO) which form a part of the APL offer services to discover other ZDO's and to build a network of ZigBee devices [3].

3.3 Ubiquitous/Pervasive Computing Middleware

Ubiquitous or pervasive computing describes an environment where computing is simply everywhere and that it is so embedded that it is indistinguishable from the environment itself. The difficulty with developing this concept into a reality is that usability and interoperability are still hurdles to overcome [21]. To move towards realising this goal, research has been performed into on developing middleware to surmount such hurdles. One such middleware which facilitates Publish / Subscribe paradigms is Distributed Asynchronous Collections (DACs) [8]; DACs aims to allow remote objects to acquire data and can improve interoperability and usability. Information which is necessary is propagated to appropriate components in the network. Another peer to peer middleware infrastructure hosts the objects and their physical devices provides efficient request routing, deterministic object location and load balancing [21].

As described in [4] 'Pervasive computing systems are being deployed in a rapidly increasing number of areas, including building automation, assisted living, pervasive computing systems are standing at the crossroads of several domains (e.g., distributed systems, multimedia, and embedded systems)'. Intelligent buildings contain systems which individually are responsible for automatically controlling a portion of the environment, as previously stated, interoperability is the challenge with facilitating a more intelligent building, perhaps context aware.

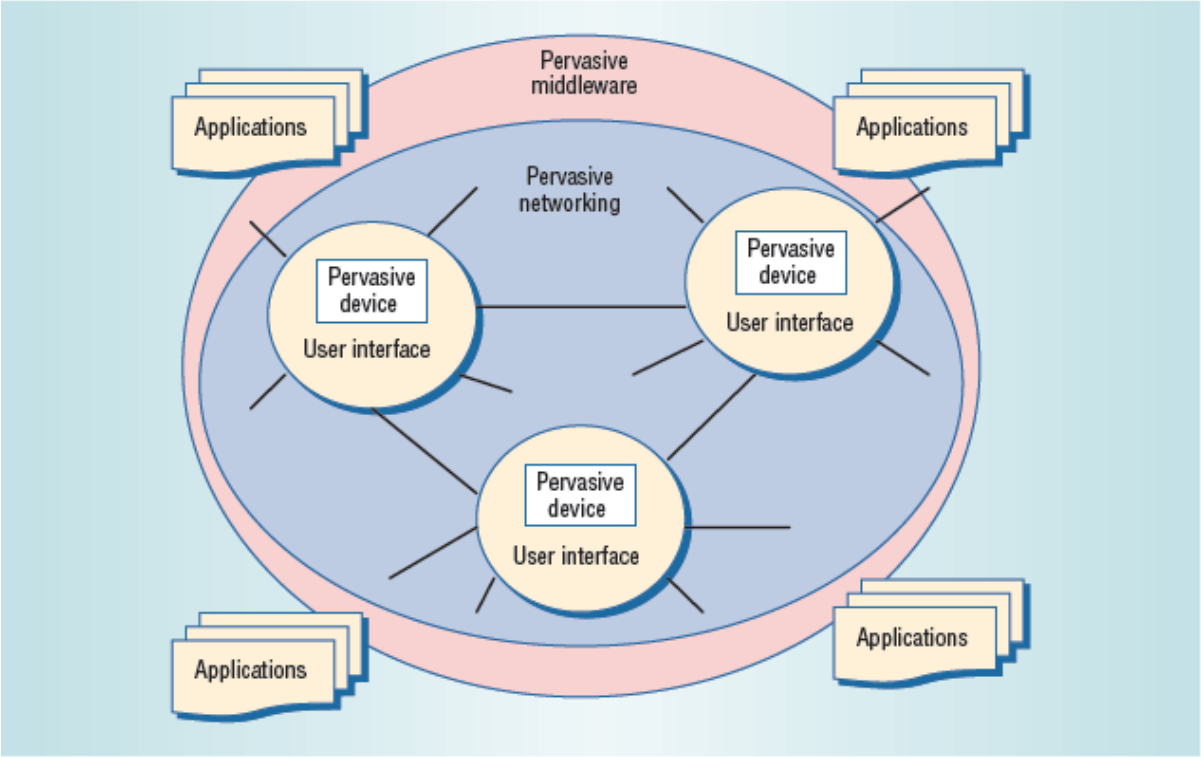

Figure 3-2 Pervasive Middleware [30]

The design of middleware which can improve the seamless integration of pervasive devices would lead to an enhanced intelligent building system. 'Like distributed computing and mobile computing, pervasive computing requires a middleware "shell" to interface between the networking kernel and the end-user applications running on pervasive devices'[30]. Support for the inclusion of the occupants of the building in such a system via multiple interfaces would yield improvements towards the buildings use. The capture of data on energy consumption by facility engineers could be used in a campaign with an aid to advocating a reduction in energy consumption by both facility engineers and other building occupants.

Wong et al. describes building qualities exhibited in modern buildings in three layers of systems.

1. The first layer deals with the provision of normal and emergency operation.
2. The second layer is performed by the building automated systems (BAS), communication management system (CMS) and office automation (OA) system.

3. The third layer contains subsystems including heating, ventilation and air-conditioning (HVAC), lighting system, fire protection system, vertical transport system (elevators), security system and communication system.

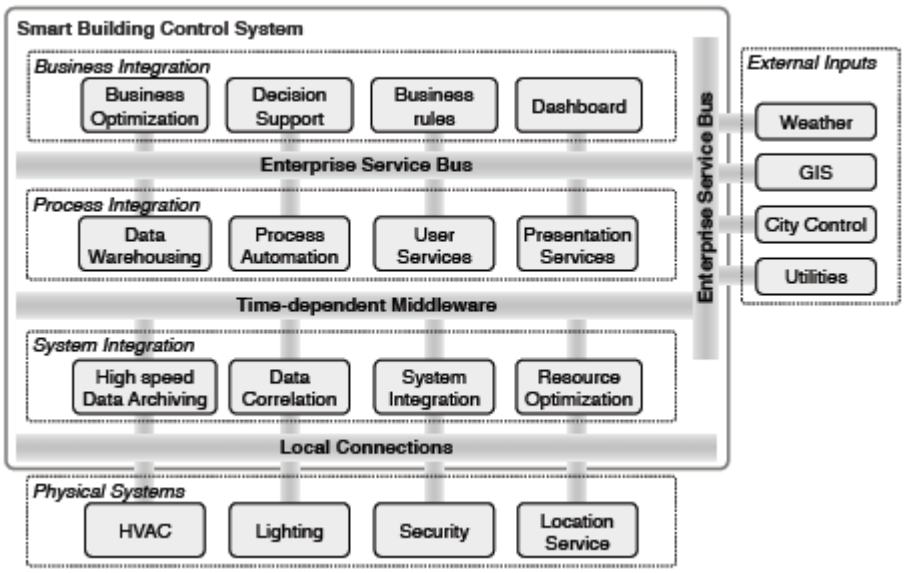

Figure 3-3 Smart Building Control Systems [35]

The Enterprise Bus technology in Figure 3-3 Smart Building Control Systems [35] offers a layered approach for the integration of systems, external inputs and an application layer for users. The use of a layered architecture offers simplicity in design and development and one that offer an interoperable solution for a BAS. To balance this solution architecture, on one side rests the accumulated weights of an Enterprise Bus with interoperability and simplicity, but on the other side lies the security risks (should the communication be open on the bus) and the difficulties of scaling this solution towards multiple building controlling and monitoring. What is needed is an architecture that is very scalable, simple in its design and development and that also offers interoperability, multiple protocol compliance and that is secure.

The Enterprise Bus offers a layered network which caters for building systems to inter communicate at a local level. This concept if extended to include a network that is suitable for wide area or internetworking could extend interoperability in a smart building system for use in a scalable solution.

3.4 Service Oriented Architecture

Service Oriented Architecture (SOA) is a component based approach to offer distributed computing services for the consolidation of a system across a network [20]. The components are loosely coupled, autonomous and reusable to improve heterogeneity and dynamicity. There are three parties and principles to SOA. There is a registry, a provider and a requestor. The provider publishes its services to the registry and allows the service to be invoked by a requestor. The requestor must first find the services from the registry and then bind the service from the provider for use. SOA offers the benefits of networking and distributed computing with a formal description for each service. The services are autonomous as well as interoperable and provide re-usable functions via a technically standardized Interface [31].

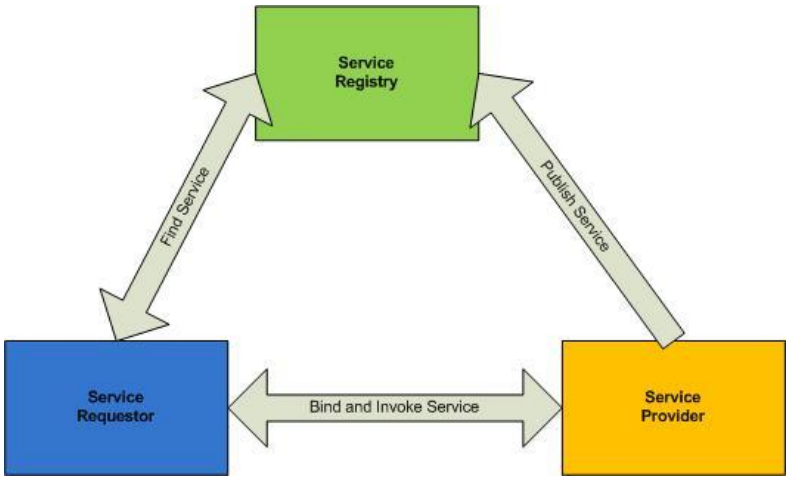

Figure 3-4 SOA Architecture

SOA also offers difficulties with regards to the misuse and unavailability of services. The services require at a minimum the governance of their use and availability, 'SOA Governance deals with the high reliability of services status, record how many services are available to serve on the platform and make sure all of the services operating within an acceptable service level' [25]. If web services are open on the Internet then their governance is the responsibility of all three parties, in this thesis the focus is on the use of a Wide Area Network (WAN) so governance remains the responsibility of the building maintainers and the primary focus is on its security.

3.5 SOA and Web Services Technologies

3.5.1 SOA

Service Oriented Architecture (SOA) is a component based approach to offer distributed computing services for the consolidation of a system across a network [20]. Web Services are one implementation of an SOA which offer a programming integration platform for networked services. Web services at a minimum integrate distributed open standard technologies such as Extensible Mark-up Language (XML), Simple Object Message Protocol (SOAP), Web Service Description Language (WSDL), Universal Description Discovery and Integration (UDDI) in an interoperable fashion to create systems. SOA and web services are the intended architecture for this application and the security of the design is the focus of this paper.

3.5.2 Web Services

'A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-readable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards'[34].

Web Services are one implementation of an SOA which offer a programming integration platform for networked services. Web services at a minimum integrate distributed open standard technologies such as Extensible Mark-up Language (XML), Simple Object Message Protocol (SOAP), Web Service Description Language (WSDL), Universal Description Discovery and Integration (UDDI) in an interoperable fashion to create systems. Fundamentally services are published in a WSDL and discovered by UDDI and data is documented in XML and passed over the network using SOAP via HTTP over the network [19].

3.5.3 XML

XML is a textual specification to describe data artefacts in a machine and human readable manner. XML is also defined in a standard which each XML document must conform to. W3C's XML schema definition (XSD) provides a standard language for defining the document structure and the XML structures' data types [7]. The benefits for using XML are that native data artefacts can be marshalled into XML and transported across networks to an end point and then un-marshalled and reused at the other end. Each XML document contains a sequence of elements and attributes which represent and describe the data artefacts. It was designed for interoperability and ease of use which have proven successful.

3.5.4 SOAP

SOAP is a mechanism which allows the exchange of textual and abstract data which is serialised in a format such as XML. SOAP is also a binding framework that allows XML to be transmitted over transport protocols such as HTTP [10]. SOAP is analogous to a completed traditional banking form which is sealed in an envelope and posted via the post network. The form is completed by a user, sealed in an envelope which has an address written on it and transported to another person via the post service. SOAP is referred to as the transportation of messages, each of which contains an envelope a header and a body. The body of a SOAP message is similar that of the completed banking form, the form is of a standard that can be read by a bank employee, similar to XML. The SOAP header contains information relating to the message content and instructs the server how to deal with the message. The address and addressee in the letter contains information relating to who and where to send the letter.

3.5.5 WSDL

WSDL is a natural progression of Interface Definition Languages (IDL) specifically for web services. IDL originated in interoperable middleware such as CORBA. A complete Web Service Definition Language specification contains two essential parts. The first part describes the web service in abstract terms. The second provides specific protocol details to facilitate the transmission of messages to between nodes [10].

In the abstract part a ‘portType’ is describes the method details and a ‘message’ defines a set of parameters in the method. The parameter ‘types’ are defined as XSD types or user defined types in a derived XSD. In the concrete part the ‘binding’ of the methods relate to the protocol for implementation to a specific end points or ‘port’ for transmission. The SOAP protocol is one commonly used for transmission in web services [7]. Lastly the ‘service’ is a collection of ports for multiple WS end points.

3.5.6 Web Services Engine

Web services are published for a client requestor to invoke and to bind transmitted XML to local code; a Web Services engine provides these tasks. Apache Axis2 is one such WS engine and is the second generation this Web Services middleware engine which was designed to meet with the explosive growth of WS worldwide. The architecture of Axis2 was designed to be component based for extensibility and with improvements over its predecessor to improve functionality in line with new developments in WS standards, e.g. WSDL 2.0 [26].

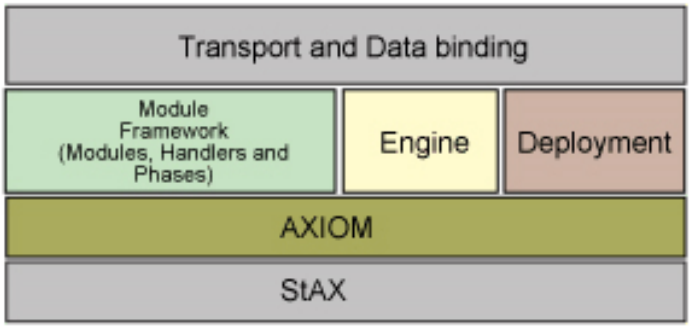

Figure 3-5 Axis2 Architecture [6]

The Axis2 component layers are listed below in Figure 3-5 Axis2 Architecture [6] At the lowest layer is the Streaming API for XML (stAX) which is a 'pull XML Parser', this component pulls XML elements that are needed for parsing and is controlled by the Axis Object Model (AXIOM) layer above. The combination of the pull parsing between the two layers reduces memory usage and provides performance improvements [6]. The extensibility of the engine includes module extensions such as WS-Security, WS-Reliable Messaging and WS-Transactions based web services. The deployment component allows WS to be deployed without restarting the engine. At the top layer is the binding component which binds the XML to native code. Axis2 has closer alliance to Java.

3.5.7 Web Service Containers

A web service engine will execute web services but it does not support the level of complexity of a distributed network of intelligent building web services interoperating with each other. Web services originally were stateless and message exchange orientated. There was no mechanism for security or persistence. An application which would require web services and such attributes would need to be designed with the structure and capabilities of a dedicated application server. For such an application, Java Enterprise Application Servers (J2EE) can host the WS engine; examples of application servers include Tomcat, JBoss or Web Logic amongst others.

The J2EE Application Servers are component based and manage components which are deployed to the server. In the case of Axis2, a web archive is created (with file extension .war) which is then deployed to the enterprise server. This web application is managed with the benefits of a dedicated application server. Extensions to web services in terms of state and security are discussed further.

3.5.8 Web Service Resource Framework

By default web services do not offer a sense of state, web Services by default are stateless transactions. However, applications and systems require a means to observe the state of an object, especially in the building and device monitoring applications addressed in this dissertation. Data which is used by objects in systems may need to reside in a database for tasteful transactions. The WS-Resources framework offers a mechanism to facilitate state in WS. WS-Addressing is part of the WS-Resources framework and is used to define the relationship between Web Services and resources with state. WS-Addressing is used to reference 'endpoints' and putting their use into context. The three components of the endpoint reference are:

Address element

ReferenceProperties element

PortType element

The three elements are used to identify URL of a WS (address) and the interface to the WS (PortType), the 'Reference Property' can relate to a Database or other external entity that facilitates the mechanism of retaining state [10].

WS-Notification is part of the WS-Resource framework and it defines a set of standard message exchanges that improves the interoperability of web services. WS-Notification specifically defines how WS which require desired information exchanges to interoperate. There are four elements to the WS-Notification family:

Publish / Subscribe Notification for Web Services

WSBaseNotification

WS-Topics

WS-BrokeredNotification

The principle of WS-Notifications is build around situation awareness, if there is a situation and an entity desires to be informed about it, the entity subscribes to be kept informed of

situation messages. The entity that produces the situation messages is referred to as the ‘producer’ and the entity which desires to be kept informed is the ‘consumer’. In order for the consumer to receive messages from the producer, it must do so using a ‘subscribe’ operation. The WS-Notifications paradigm can be used be used in conjunction with WS-Resources such that consumers can be informed when reference properties values change. Thus interoperability and stateful web services based systems are designed. Publish / Subscribe Notification can be taken a step further with a marriage of WS and self discovery middleware to construct interoperable application services [9].

3.6 Web Service Security

Web Services are designed to be available on networks and more often than not the Internet. In the case of Intelligent Buildings and the use of web services over wide area networks, a certain degree of security needs to be implemented to prevent unauthorised system access to and undesirable modifications to buildings. Global standards exist in providing online security, and further standards exist in the provision of standards for web services security. The use of policies which define what standards are to be used in conjunction with web services are growing in popularity and offer improvements over one off derivatives for defined policies. 'Security, reliable messaging, privacy, and transactions are the immediate underlying efforts to define Web services policies. For security-related assertions, the new Organization for the Advancement of Structured Information Standards (Oasis) Secure Exchange (WS-SX) technical committee is using the WSSecurityPolicyproposal, which addresses a domain that spans standards under both the WS-SX technical committee and the Oasis Web Services Security (WSS) technical committee' [1]. In the context of intelligent buildings across multiple sites, there is the need for a distributed security mechanism to allow existing security standards and schemes interoperate. 'The purpose of the OASIS WS-SX TC is to define extensions to OASIS Web Services Security to enable trusted SOAP message exchanges involving multiple message exchanges and to define security policies that govern the formats and tokens of such messages'[1]. Simple web server authentication can be adopted as a security policy but on its own, it is not substantial enough to offer system integrity across an intelligent building system.

The advent of intelligent buildings present challenges in software engineering, more specifically with the difficulty of interoperability between distributed systems and sub-systems. This thesis relates to the security concerns during the design and development of a distributed energy monitoring and control (DEMAC) application for use with networked buildings and the intelligent systems contained therein. In the cases of large corporations with many geo-located buildings and a mobile workforce there is the need for a distributed network of intelligent buildings.

3.7 Summary

This chapter focused on what constitutes an intelligent building and listed examples of sub systems found in intelligent buildings. Some different protocols which are found in the industry were examined and the reasons for their being. The concept of pervasive middleware was explored and how pervasive devices can contribute to a smart building. The Enterprise Bus principles were examined to determine the benefits for smart building systems. Finally SOA and related technologies were discussed with their design and core components discussed for interoperability and distribution benefits.

In the next chapter SOA and other technical solutions are examined for the purpose of leading to the design of DEMAC.

4 Existing Networked Energy Management Research

SOA purports to offer a simple scalable secure solution to BAS interoperability. The existing BAS protocols require some method of integration with SOA and Web Services. In order to validate the approach to take in a design of DEMAC, existing research approaches that utilise the current protocols are evaluated and their merits and faults taken into consideration.

SOA and web technologies offer one solution to bridging interoperability between different networks of devices with different protocols. There are other solutions that can be adopted. The use of gateways between networks is another possibility. Some research in the field of intelligent building systems and interoperability is explored further to ascertain the best way forward for DEMAC.

4.1 Dogmotic OSGi Gateway (DOG 2.0)

The DOG 2.0 was devised by the Elite research team the Politecnico De Torino University in Italy. The DOG 2.0 framework is a proven solution in the home automation arena and facilitating interoperability between intelligent building systems. At its core is an OSGi gateway implementation to assist with interfacing and managing connected building devices to the system. Further to the list of modules is a building network bridging system (e.g. KNX and Biticino) which bridges between different network protocols. Protocols such as BACNet and UpNP can be bridged in this manner. The DOG 2.0 framework also comes with Ontology support DOG ONT. DOG ONT uses semantic web as a foundation which enables the reasoning and comprehension of devices and their functional capabilities. Rules engines such as Drools can be used in addition to DOG 2.0 for intelligent capabilities in relation to autonomous building control and monitoring.

DOG 2.0 has its own API for registering devices (with the device manager) and messaging system for registered devices to emit and receive messages. The device message exchange operation is similar to the Publish / Subscribe pattern where subscribed devices receive

messages from the publishers. Drivers are configured for operation with DOG 2.0, such as Biticino and KNX for routing messages between different network protocols.

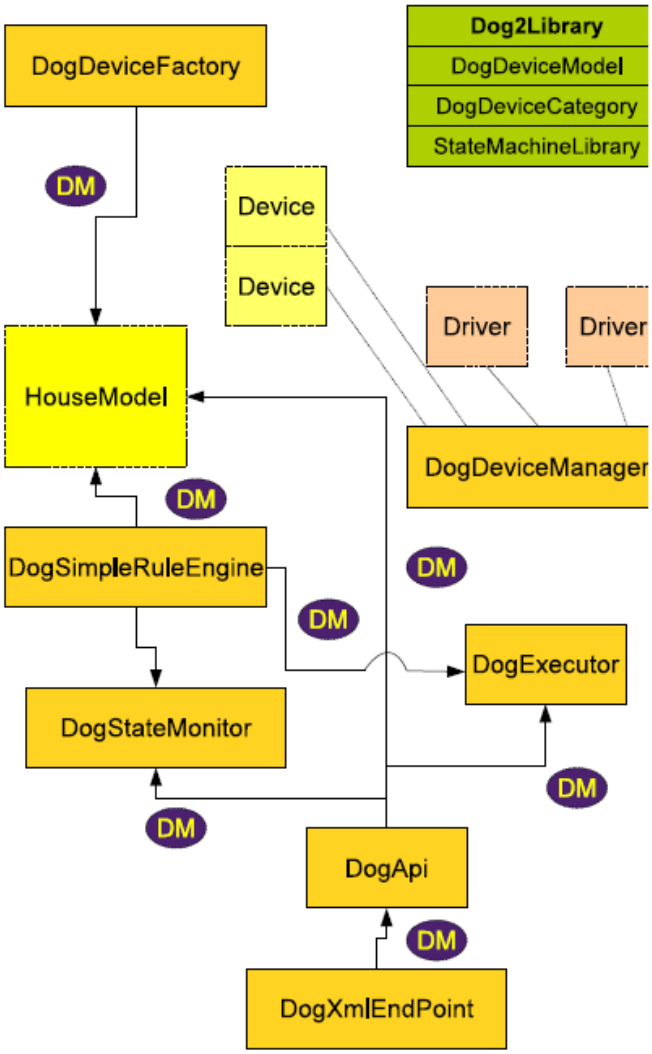

Figure 4-1 Dog 2.0 Framework Architecture

4.1.1 The UPnP and ZigBee gateway

Kim et al. created an UPnP – ZigBee gateway (UZG) which from reading his work was based on the UPnP IDG mentioned in the UpNP section, see 2.2.3. The modification to the IDG offers internetworking solution architecture between ZigBee devices and UPnP devices [32]. At the heart of the UZG is an UPnP proxy manager and a ZigBee Application Object Manager and a ZigBee Device Manager.

This solution architecture resides at the data link layer and while it offers seamless integration between the two standards, it does not offer a high level scalable solution for interoperability on a large scale distributed network.

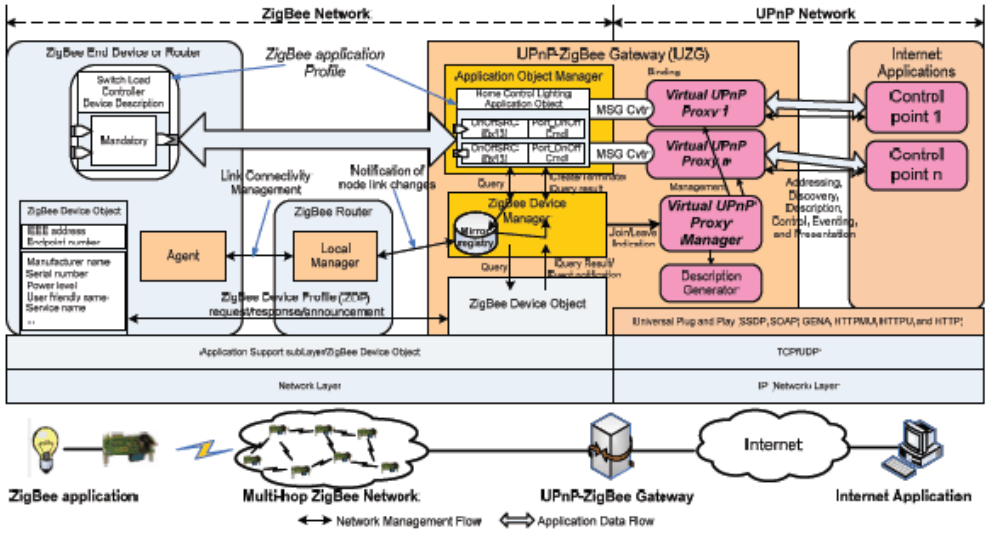

Figure 4-2 2.6 UZG [32]

4.1.2 SOA, Web Services & Ajax

There have been quite recent developments on creating internet facing BMS solutions; different paths have been taken each one depending on the end user and the requirements of the solution.

The approach taken by Xiao et al. [2] was to adopt a Service Oriented Architecture (SOA) to develop web services facilitated with Asynchronous JavaScript (Ajax). This solution incorporates three-tier architecture. The base tier of the building automated system (BAS) includes the most pervasive industry protocols e.g. BACNet and LonTalk, these protocols are responsible for facilitating the communication of the controllers with the end devices which they control. The next tier wraps the BAS protocols within web services. The web services publish the functionality of the BAS via a Web Service Description Language (WSDL) and accompanying Service Broker. At the uppermost tier sits the web application. It integrates the web services offered by the BAS with the Internet. This approach allows web applications to control and monitor the BAS from across the internet.

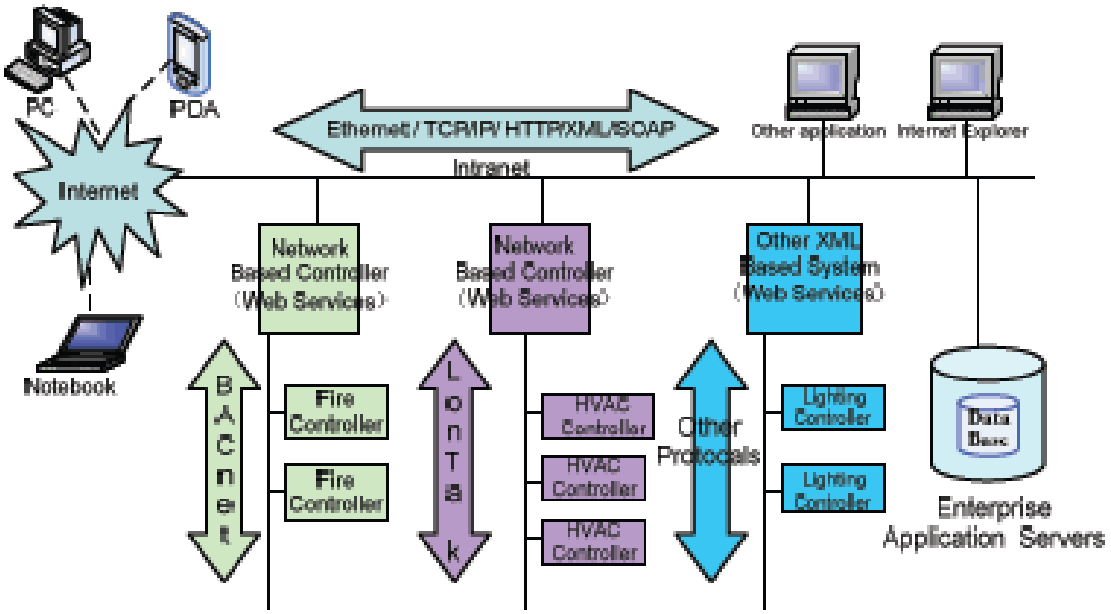

Figure 4-3 SOA web services & Ajax solution [2]

The significance of this approach is that a SOA is used successfully for networked BMS / BAS. The incorporation of Ajax in the web application with interfaces to web services improves the network traffic throughput; it does this by reducing the overhead of web server requests as it requests desired information from the web server rather than the entire web page. This reduces the number of sizeable web page refreshes. Of merit for this approach is that the technology of incorporating the BAS protocols with SOA technology has been proven for its suitability for the BAS industry. This approach however does not articulate the suitability of web services for use with multiple buildings. The use of industry standards for a multiple building scalable solution is also not considered.

Other three tier architecture systems for multiple building automated systems have been successfully developed; the Intelligent Energy Management Network (IEMN) [13] incorporates a familiar architecture with web services with building automated system (BAS) protocols (consult Figure 4-4 IEMN [13]). This system builds upon familiar BAS but integrates the building management system with the facility management system. Huang et al. comment on the fact that there is a 'lack of efficient intelligent service architecture for remote monitoring and control'. A broader view of BAS is taken and the system develops the SOA architecture into a distributed system. There are three application tiers for this system, at the base tier is the Control and Management Station (CMS) which is not dissimilar to Xiao et al., of more significance is the addition of an Area Control and Management Centre (ACMC). This system takes in to consideration security of access and also future extensions.

The IEMN utilises BACNet exclusively and creates surrogate objects which allow intercommunication between remote devices. Only Object Access Services and Remote Device Management are utilised. The Object Access Services creates objects which represent individual devices on the BMS. Remote Device Management provides a tool for maintenance and troubleshooting by dynamically locating devices, it operates similarly to a discovery service. This implementation utilises the BACNet object and service model. The IEMN system collects energy and control data from the objects for each CMS and provides this data and access to the upper tier ACMC. This layering and data aggregation provides real time information on energy consumption and facilitates the control of the networked buildings.

Energy data aggregation from the CMS to the ACMC is a significant step towards smart energy management and control, but the system lends towards a centralised server (ACMC) which is a bottleneck and a single point of failure. The surrogate object model for BACNet is a novel approach to data aggregation which is of merit in energy monitoring and device control.

4.1.3 Wireless Solutions

The BAS solutions thus far mentioned have focused on wired building network solutions. There are existing buildings which need extensions to its BAS but when a wired network extension is not feasible, 'wireless mesh technology allows designers to build electronic networks without ripping apart buildings' [17]. The addition of wireless sensors and devices can allow a solution. One approach which extends BACNet into the wireless arena is a network extension that integrates a wireless solution ZigBee [24] into an existing the wired BACNet network infrastructure. In this approach the application support sub-layer for ZigBee is adopted as one of the data link layer options for BACNet. This experimental approach was simulated and test results from Microsoft's Windows application Visual Test Shell (VTS) for BACNet indicated favourable transactions for BAS implementation. Subsequent to this approach, BACNet's Wireless Networking-Working Group (WN-WG) began work on a specification for BACNet over ZigBee [24]. At the time of writing this thesis the specification is not complete but a wired BACNet solution is a possible extension into developing a wireless mesh solution.

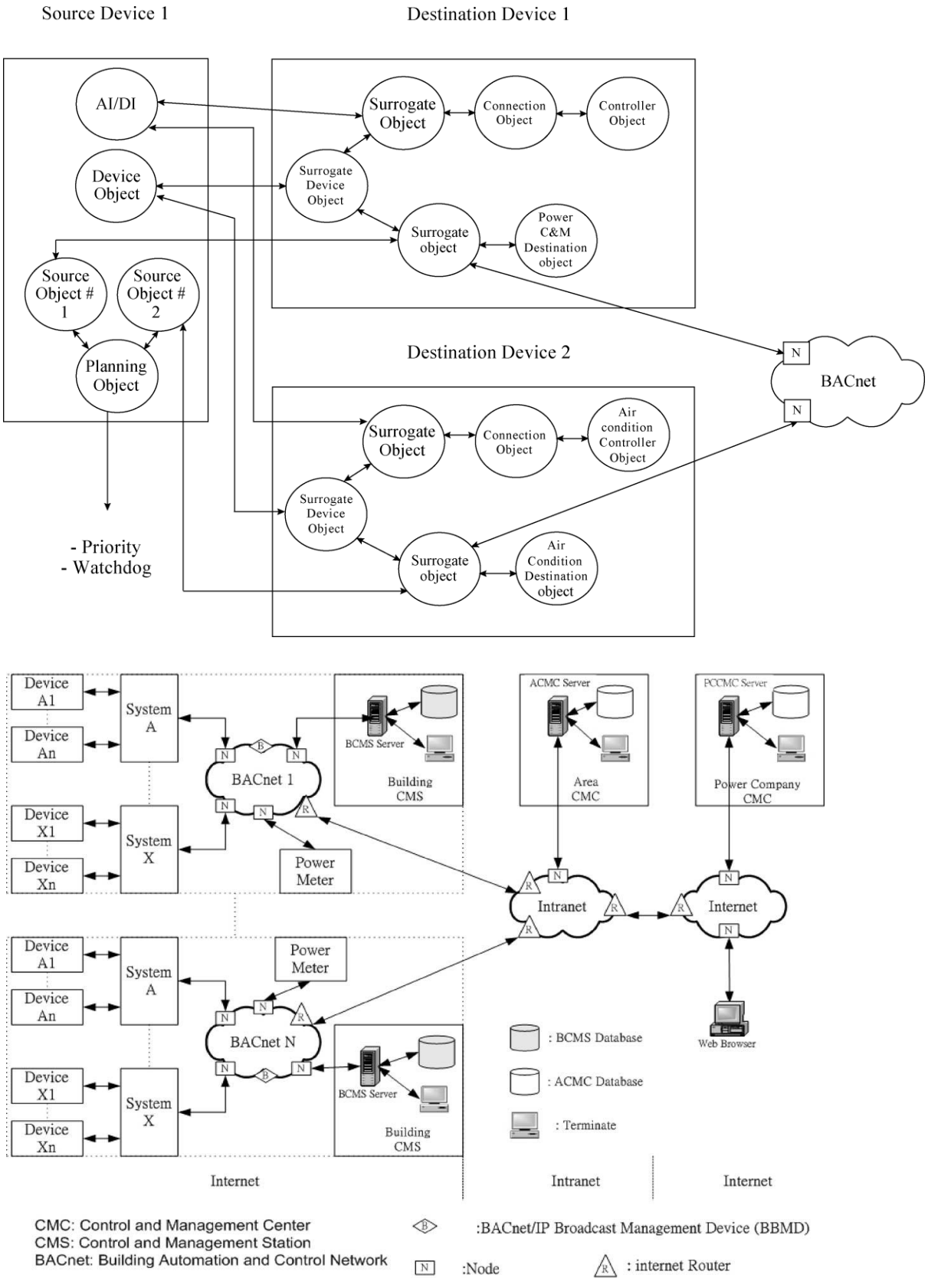

Figure 4-5 IEMN [13]

4.1.4 Heterogeneous SOA solution

Leguay et al. [18] developed a multi-level service SOA solution which incorporates wireless sensor networks (WSN), DPWS and Web Services. The WSN solution lies at the lowest level in the solution architecture stack. At this level a gateway was used which allowed DPWS clients access the ZigBee / WSN [18]. The mid level was where the DPWS resided and at the upper level lay the Web Services. The benefits to this approach allowed dynamic device addition to the network and also for different computational and capacity devices to share the same network with services and data. This solution utilised the aforementioned Publish / Subscribe paradigm. This paradigm managed the interoperability of data exchange between different 'interested' devices. A device published data which it was able to furnish, e.g. temperature information; a different device which was interested in acquiring this data then subscribed to it. If a different device did not subscribe then the data was no longer published. This approach reduces network traffic. The important benefit with this approach allows for small devices with limited electrical and computational power to disseminate information across a large network. This solution was focused on the integration of small devices into the larger network via a gateway; however gateways do not offer a scalable solution for multiple intelligent buildings. It is a solution for WSN and standalone buildings.

4.1.5 Middleware for heterogeneous subsystems interoperability

Perumal et al. proposed an event- condition-action (ECA) interoperation schema which uses SOAP as an interface for Intelligent Building middleware. The use of SOAP is a proposed solution to legacy systems interoperation and also for facilitating heterogeneous system communication. Specifically the concept of device events being traced and acted upon by other systems is a concept to improve further system integration. In this proposed solution the middleware was envisaged to 'provide agreement for integration and act as service mediator between diversified subsystems, therefore there must be rules and procedures that could enable subsystems to work together in a federated manner' [27]. To paraphrase Perumal et al. the rules were designed to allow the subsystems intercommunicate to bridge the heterogeneous systems. This proposed system was not fully developed and deployed but used for experimental analysis.

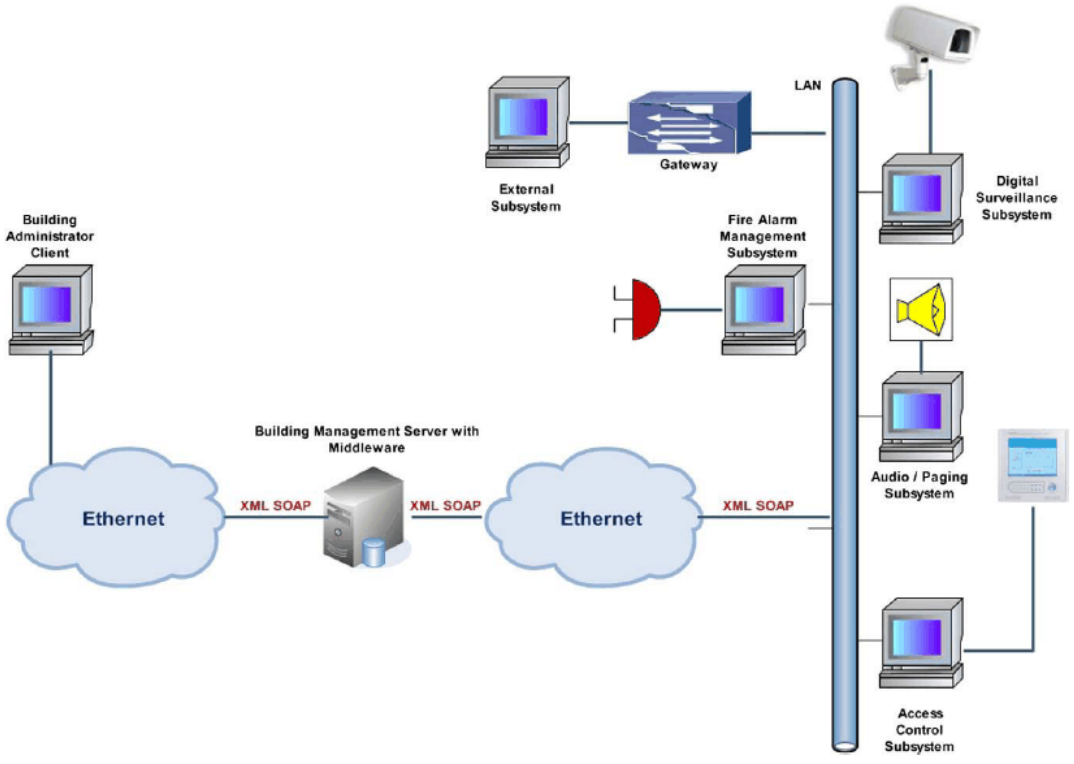

Figure 4-6 Proposed ECA solutions [27]

4.1.6 Open standards for Building Information Exchange

In 2006 OASIS released their open standards for building information exchange (oBIX), and it is an industry-wide initiative to define XML- and Web Services-based mechanisms to Building Control Systems [23]. 'oBIX provides a flexible object model to describe the data and operations available on the server. In oBIX everything is an object: objects are also used to describe data types (classes) and operations (method signatures). The flexibility of oBIX is based on the possibility to custom define any kind of object.' [22]. The use of oBIX allows existing protocols to become interoperable with other protocols using existing open technologies such as XML and SOAP. oBIX has native data formats to represent low level data types but is also extensible as it supports custom specific data models. Since oBIX is open source and community driven there tools already developed which can be used to further the development and use of oBIX in intelligent building automation.

4.1.7 Summary

This chapter is focused on evaluating researched solutions for interoperability in intelligent building systems. Initially an overview of the DOG 2.0 framework which is an OSGi based system is given. The Publish / Subscribe pattern is discussed in relation to DOG 2.0 and its suitability for multiple devices and scalability. Following DOG is an UPnP and Zigbee system which is then evaluated with Web Services as the technology solution. Web Services is also explored in a three-tier SOA solution which has AJAX for efficient client side communication with the server.

IEMN is then examined as a device object focused interoperable system with the BACNet protocol at its core. Wireless solutions are also examined and solutions which also utilise the Publish \ Subscribe pattern. An Event Condition Action system is explored before a final examination of another potential object focused solution oBIX.

With regards to an application for distributed energy monitoring and control (DEMAC) these different solutions offer many technologies, patterns and solution designs. Two opposing solutions for interoperability that stand out in terms of interoperability, scalability are gateways and Web Services. The three tier SOA design offers a scalable solution where multiple networked devices are interoperable. The use of objects in the listed solutions is limited to a single network protocol, but they do offer low level interoperability. oBIX is an open object standard which offers interoperability for multiple network protocols but which has not had many implementations from research for this thesis.

The Event Condition Action pattern wasn't implemented as a solution and doesn't appear to be scalable to large buildings or multiple building. The Publish \ Subscribe pattern has been implemented successfully on two system solutions.

The result of analysing different researched solutions offers two potential avenues of pursuit for a design for DEMAC. SOA and a three tiered architecture incorporating Web Services and related technologies is one potential solution and an OSGi based gateway solution with a Publish \ Subscribe pattern yields another. These design solutions are examined in the design chapter.

5 Application Solution Design

5.1 Requirements of DEMAC

To reiterate the recent and relentless focus on global warming, it has left an imprint on society to reduce carbon foot prints and to be more energy conscious. DEMAC is required to assist with an individual's reduction in energy consumption in their respective building and to improve their daily activities while occupying the building. DEMAC is also required to facilitate the interoperation of pervasive devices in the current building system and to also facilitate the connection of any computing device by extension if necessary seamlessly into the buildings systems.

The niche market for which DEMAC is targeting is that of corporations which require a distributed (smart building system) energy monitoring and control application for enhancing the building environment and energy consumption within the buildings owned by the corporation. The target corporations which this application is designed would not have an existing application which can perform these functions and have enough buildings to warrant the investment in this application. Ideally any user could make changes to the environment, but considering the impact that these changes could have on the building environment and devices therein, it would be best suited to those with training or qualifications in HVAC and related technologies. In order to cater for both experts and novices, a zone based system would allow this to be feasible. If time permits during the application development, a security system could ensure that only users with certain privileges can have access to certain zones, thereby preventing unauthorised changes to devices and systems that they have not relevant authority to access.

While both these target users are ideally customers, the priority customer and user is the building facility engineer who is professionally responsible for the upkeep and maintenance including security of the corporations' buildings physical infrastructure and environmental conditions. For this reason the application will be designed with this individual as the priority requirement driver and other 'novice' users as lower in the priority list.

Prior to writing this thesis, I have had over five years experience in the intelligent building systems industry. During this time I have worked in a research and development department where I gained exposure to experienced Irish facility engineers and their desires for software system functionality. From this experience I drafted the requirements of the DEMAC application; they are listed below in table 4-1:

1.	Highly available, 24*7 access and up time.
2.	Support the concept of building zones (floors, areas and rooms etc)
3.	Secure access to these zones
4.	Interoperability with existing building systems and protocols
5.	Automatic discovery of intelligent building devices
6.	Data aggregation of specified energy consumption and or device usage / changes
7.	Graphical and data display of historical energy consumption per zone
8.	Support heterogeneous networks
9.	Extensible to new devices, buildings and networks
10.	Hierarchical composition of buildings and contents for ease of support
11.	Platform independence
12.	Highly Usable by technical competent individuals
13.	Monitor Control distributed buildings remotely over the Internet from web enabled devices
14.	Real time – monitoring and control of any building
15.	Scalability of application for multiple different geographically located buildings & large buildings of 20+ stories
16.	Ease of development within 3 month time frame

Table 5-1 DEMAC Requirements

5.2 Solution Architecture Design

5.2.1 High Level Solution Architecture Design

Complex systems will develop at a faster pace if they are composed from more simple systems [14]. A hierarchical system is composed of more simple systems and having a different levels, each with defined responsibilities, offers a scalable solution which can be implemented in a short time frame quite well. The solution architecture is discussed in term of three levels of abstraction. The first layer is at the highest level of abstraction (see figure 4.1) and the system is represented by integrating different intelligent building systems from the Internet with layers of security protection between each building and the internet. Each building will be accessible via HTTP protocols, GET requests and responses. Each building system will either have a set of web services for its subsystem or other xml based messaging system. The WS-Security standard will be implemented to provide user authentication and message integrity during operation.

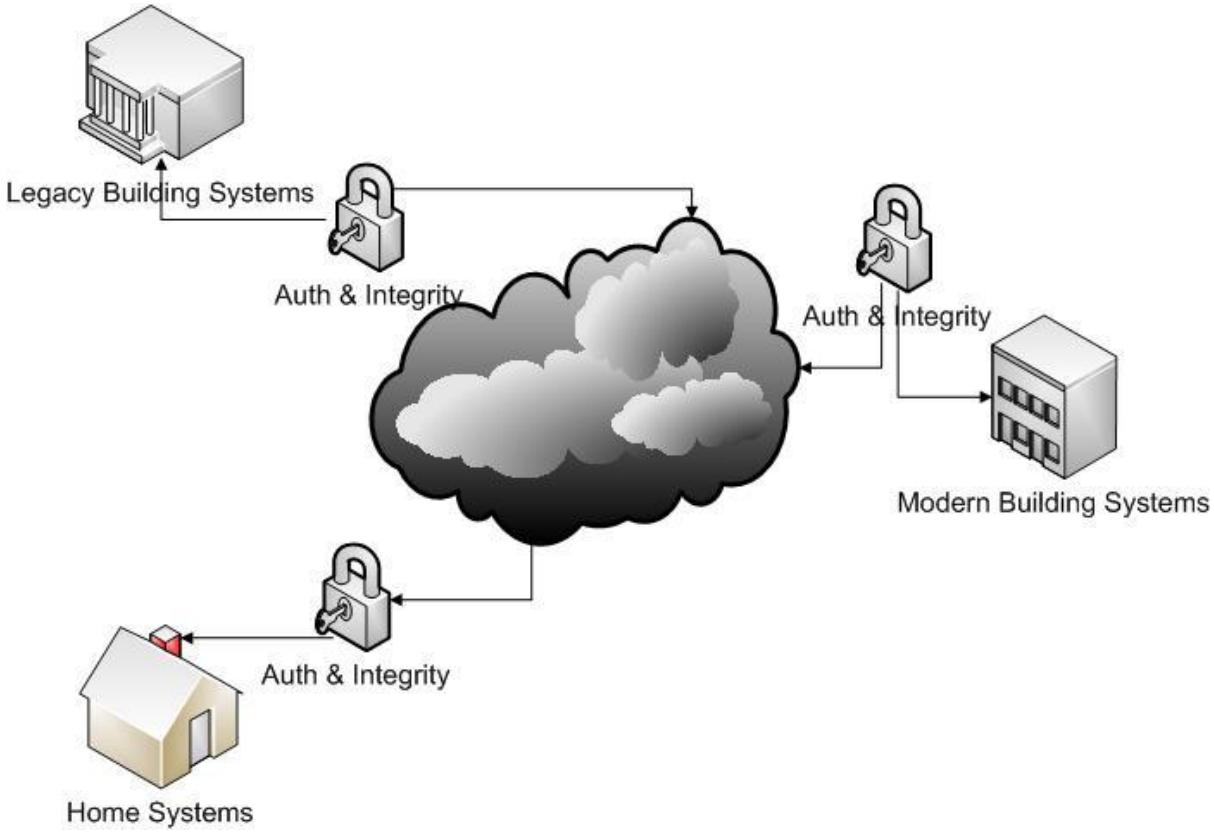

Figure 5-1Distributed Heterogeneous Secure Intelligent Building Application Solutions (mobile & fixed client interaction)

The second level of abstraction is at a subsystem level in a building. The building is an aggregation of zones. The zones represent a user defined space within the building. This zone based concept is to improve shared ownership of space with other users so that groups of small amounts of users take ownership of their zone to suit their needs and are also held accountable for energy usage. Each zone has its subsystem of lighting or HVAC subsystem, and access to these subsystems is via web services. The zones ideally would be representative of the building structure. Within a building there may be several floors, each can be representative of a zones. For larger buildings, further breakdown of zones may be to specific areas which are located on a particular floor, e.g. reception area, particular offices etc. The systems for a particular zone and the devices that constitute the system in that zone would form a local zone which will be controlled and monitored via the application. Changes that users make to the system which impacts these zones will be recorded in a database for historical record and report generation purposes. The data contained therein can then be used for energy consumption is aggregated into a data store for historical analysis purposes (see figure 4.2).

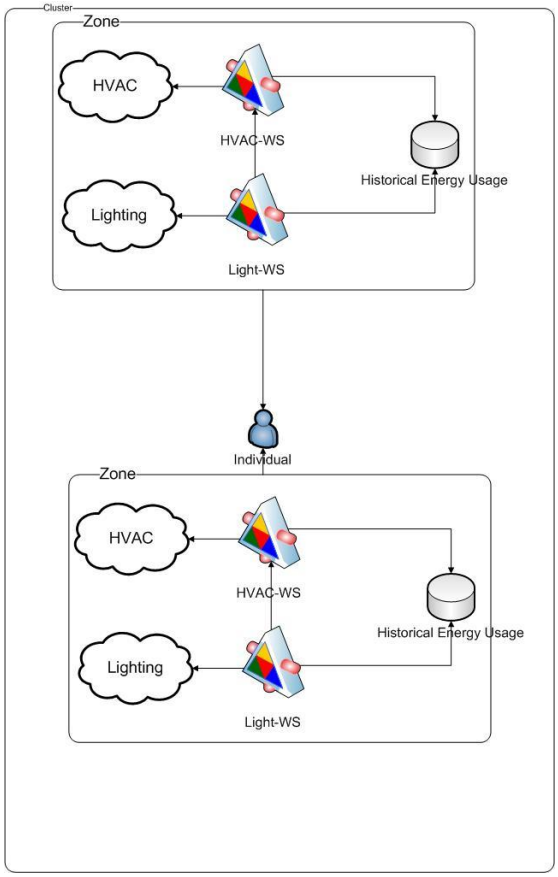

Figure 5-2 Zone based web services and data aggregation.

Access to a zone and its control and energy data is ultimately viewed from the user's browser client who connects to a web application server. Information pertaining to a zone is retrieved from the system from interrogating the web services from the application server layer. This interoperability is abstracted away from a user. An idea of this use case is depicted below (while not a strict UML Use Case, it gives an outline of the user and system actions). Messages are passed between the web services of different systems to be presented to the user in real-time. The application server deals with the logic of the request and executed the particular web service and DB requests.

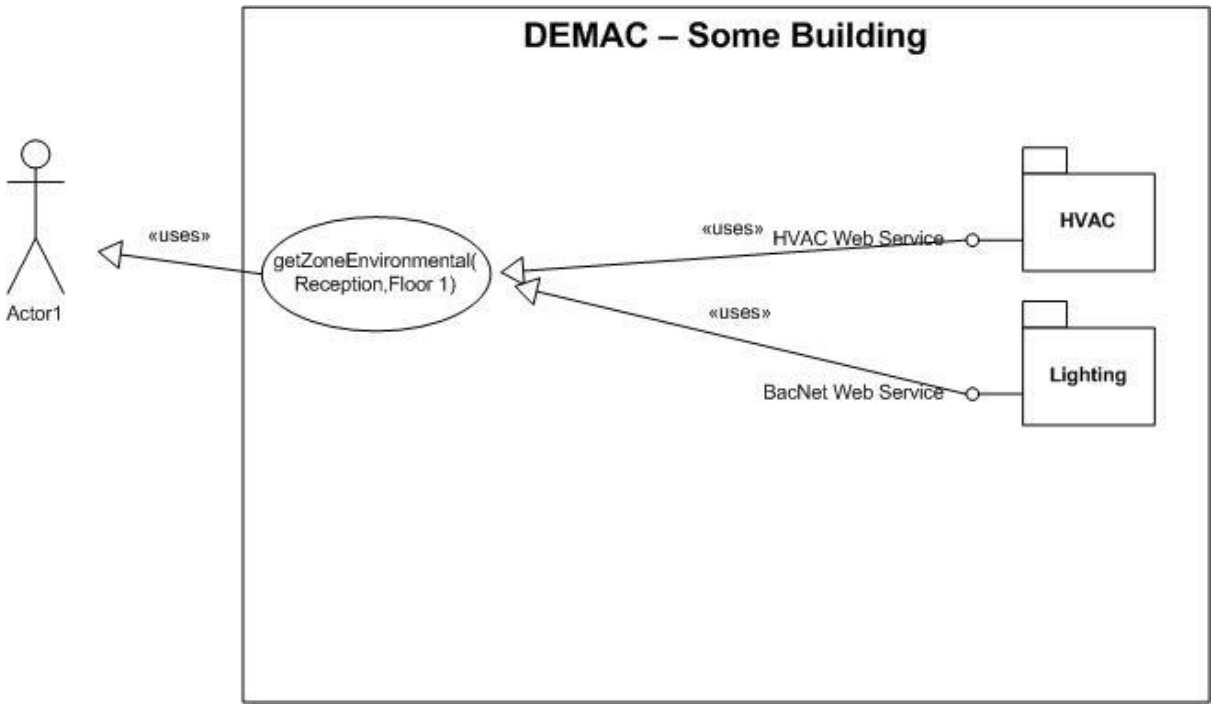

Figure 5-3 Use Case of a facility engineer retrieving zone information from the system

5.2.2 Low Level Solution Architecture Design

At the design stage two specific competing designs were considered. One design was that of SOA three-tier web application server architecture, but an alternative architecture incorporating the Domotics OSGi Gateway 2.0 (DOG 2.0) framework was also evaluated. The DOG 2.0 framework is a proven solution in the home automation arena. It has an OSGi gateway implementation at its core. The DOG 2.0 solution would provide the level of functionality required in terms of interfacing with building systems via configuring device drivers, facilitating the use of ontology to manage monitoring and controlling building systems. The DOG 2.0 framework could also be used as a platform in order to develop a web enabled application using its xml messaging system for communication between buildings and their intelligent systems.

The three-tier web server architecture approach as stated is an alternative for this application. This approach does not provide as much features as DOG 2.0 'out of the box' specifically for intelligent building systems, namely for bridging network protocols, or offering ontological support but what it does offer is a proven architecture for hosting web enabled distributed applications. The use of standard web technologies, with service oriented architecture is supported from the open source community and the enterprise level for a platform in which to begin. The SOA also offers simplicity in design, software reuse and the knowledge that existing research had also successfully developed applications for intelligent buildings in this way.

Ultimately what is required is to develop an application that meets with the requirements and can be developed within a rigid timeframe. In the case of each competing design the system (e.g. HVAC) objects are represented as objects which interface with the native protocol e.g. BACNet. The interoperability of each subsystem is permissible via the interface (web service or DOG 2.0 network driver). The users of the system in either case must access the system via the network or Internet and are served up their content from the web server. The users may then control or monitor data usage remotely. In order to establish the best design for DEMAC, both approaches were designed and their advantages and disadvantageous listed. In the implementation stage small applications were developed to evaluate the best solution design for DEMAC to meet with the requirements and also to be

developed within the timeframe. Below is the DOG 2.0 approach in Figure 5-4 Initial DOG 2.0 Architecture to meet requirements

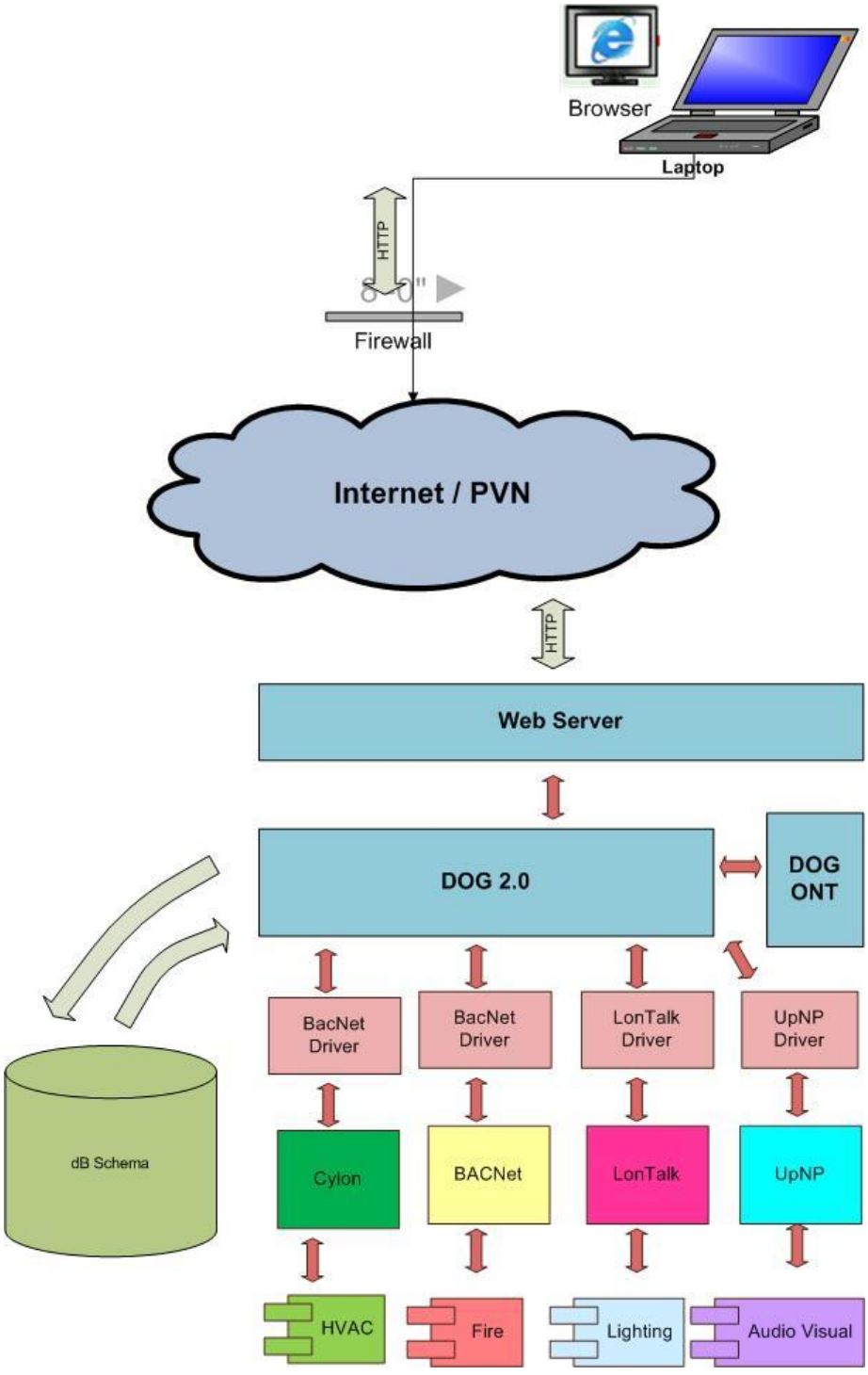

Figure 5-4 Initial DOG 2.0 Architecture to meet requirements

The SOA Architecture is depicted below in Figure 5-5 SOA Architecture design. A table which depicts the comparisons of each design is listed in Table 5-2 Comparison of opposing designs versus requirements. They would be used as a reference when determining which design solution to implement as the full application solution and also to ensure that both designs meet with the requirements.

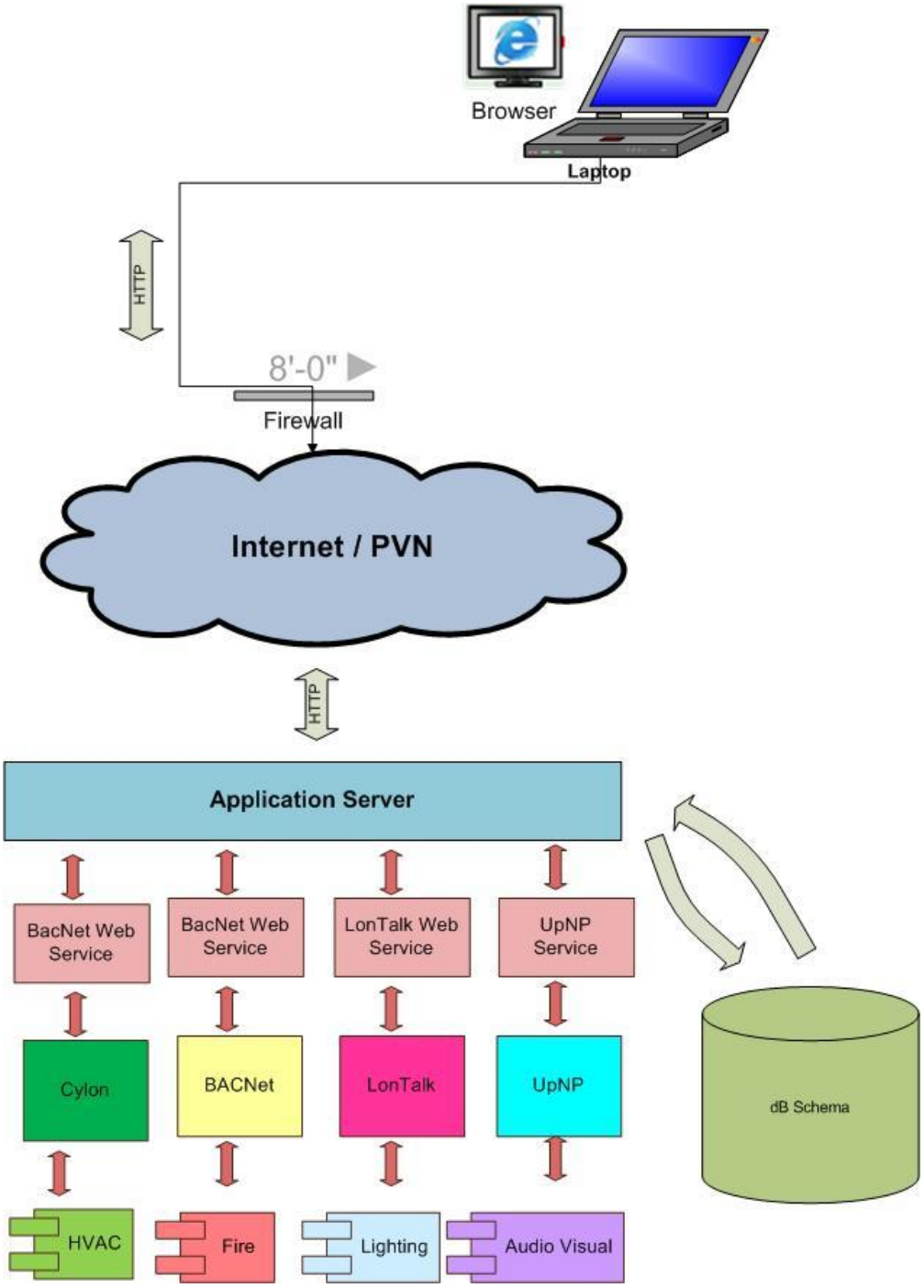

Figure 5-5 SOA Architecture design

Requirement	DOG 2.0 Features	SOA 3-Tier features
Automatic discovery of intelligent building devices	Web Semantic support	Web Semantics addition required
Network Interfaces for system interoperability	Configuration Drivers for standard building network protocols	Web Services required
Ease of development within timeframe	XML based messages for machine reading ability	SOAP messages are machine reading ability
Ease of development within timeframe	Open API proven in intelligent building management	API development necessary
Automatic discovery of intelligent building devices	OSGi gateway implementation	Web Service interfacing to protocols required
Hierarchical composition for ease of support Platform independent Control distributed buildings remotely over the Internet Real time – Monitoring and Control Scalability for multiple buildings & large buildings	Disjoint Web Server to DOG 2.0 interface required	Web Server combined with Application Server to host application
Ease of development within timeframe	API - Open source code and published documentation	Standard web technologies with supported open source servers
Interoperable with existing building systems and protocols	No drivers available for network protocol	Web Service interfaces to network protocols
Highly Usable Support heterogeneous networks & Extensible to new devices, buildings, networks	Html over http support required	Dynamic html over http support available
Secure access to zones	Security required	Security required
Support the concept of building zones & Data aggregation of energy consumption	DB interface required	DB interface required

Table 5-2 Comparison of opposing designs versus requirements

5.3 Design Features Overview

5.3.1 Building System Network Interface (driver / web services)

This feature is common to both designs and it is required to retrieve data from the low level system devices and also to make changes to these devices. The implementation ideally should hide the details of the network but provide an interface in which the data exchange can take place. Either the network configuration driver in DOG 2.0 or the use of web services will provide this feature.

For the web services implementation the interface is documented in the WSDL and the messaging between the two systems is via SOAP messages. The web service implementation will transfer requests and responses between the application and the devices located on the network system. In DOG 2.0 the network configuration driver acts as the interface and maps the DOG 2.0 XML messages to the relevant network requests and responses.

5.3.2 Building System Network DB Schema

In both designs the DB schema is required for three important functions.

1. The building system shall be captured in a two dimension tabular relational database schema. The schema will model the zones within the building and the devices contained therein. As previously stated the building physical infrastructure must be captured which is immediately comprehensive to the user. A hierarchical named structure not too dissimilar to directory structures will be used. The relations between different layers within the zones will be mapped out using foreign keys and joins between tables.
2. The physical contents of the building which make up the intelligent systems will also be captured in the schema. The devices names and locations will also be stored and mapped back to their location in the respective zones. These physical devices will only be accessible by the users from within their zones hierarchy. The numerical

address of the devices point (which the user retrieves and alters information) once again will be accessible from the zones.

3. The data store will store historical data for the system devices execution. The storage of this data will be retrieved by the application on a user defined periodic basis and stored in the data store. When required by the users for reports or analysis purposes the application will retrieve it and display it graphically. A scheduler class will be required in the SOA implementation but the DOG 2.0 has an in built scheduler class which can be leveraged for this.

5.3.3 Web Server

The web server is straightforward in that it serves up text and images in a browser friendly format for the user to interpret and interact with the application. The web server will accept connections from user's web enabled devices and forward on the requests to the application server. The web server will also present application data in html or other standard web technology from the application server to the user. Considering the requirement of the application the web server must be capable of interfacing with the application server and hosting dynamic and static content.

5.3.4 Application Server

The application server in the case of the DOG 2.0 design is the DOG 2.0 framework and DOG ONT itself, in the case of the SOA design; it is a combination of the application server and web server. Fundamental components in the application server are as follows:

1. Web Server component: This component is an interface between the web server and application server service and will create session and request parameter objects.
2. Application server service: this component will interpret user requests and determine what action to perform. Having decided on the action required it will build a response message and make connections to the pertinent sub components

(DB Manager, relevant SBS Network interface) in relation to fulfilling the response message.

3. DB Manager Component: This component is responsible to interpret a building type request and to populate the relevant building response objects from the data store. The appropriate building objects are comprised of the building hierarchical structure in the database (e.g. building zones, devices and points)
4. Network Interface: This component will interpret the request and connect to the appropriate network interface and device(s) and build up the response which may or may not require the retrieval and dispatch of real time data and build this into the response.

Data Logger Scheduler: This component will be required to determine time intervals and SBS device points and then capturing data relating to the point and storing it in the data store at the determined time interval.

Chart Graphing: This component is required to interpret historical data records and to graph them for presentation to users.

5.3.5 Security Requirements

With regards to DEMAC, it is intended that authorised users of the application to be able to connect from any PC on the Internet, it is also expected that these users if authorised may access any building that DEMAC manages (should their level of permissions allow it). With this level of global availability it will be necessary to secure the application from any form of unauthorised access. The monitoring of a small number of users with global permissions will need to be performed so that an audit trail is kept for legal reasons.

1. The first requirement will be for the authentication of users to the system.
2. The second requirement is that for each user that they are only permitted to use the application within their level of permissions.
3. The third requirement will be to prevent the replication or modifications of user's activities; this is to prevent imposters from gaining any level of unauthorised access to the application. For this reason it will be required to assure the integrity of each user's online activity so that the replication of messages is not possible.
4. The fourth requirement is to be able to keep a log of users activities for legal reasons.

5.3.6 HTTP Basic Authentication

This application utilises distributed web services and browser based clients, as such a more secure method of identifying and authorising users is required over HTTP Basic-Authentication. Basic-Authentication is not considered a strong enough method of authentication for this system. The distributed user's accounts and the ease with which Basic-Authentication can be compromised warrant a more advanced method of authorisation and message integrity.

5.3.7 Kerberos

Despite the fact that DEMAC is distributed with several web services across multiple buildings, it will be used within a single security domain that belongs to the company which manages the buildings. Kerberos is a network authentication protocol developed by MIT which uses strong cryptography and offers user authentication over the network. In this application development Kerberos meets the security requirements and interfaces with a distributed client server architecture. As already stated the Kerberos protocol employs strong cryptography which will ensure the integrity of messages within the system. The use of a third party trusted Key Distribution Centre as part of the Kerberos architecture will also provide the management of all user ID's and passwords.

5.4 Security and performance tradeoffs

The use of Kerberos in conjunction with the WS-Security and related standards meets all security requirements for DEMAC use across the Internet. The problem is that at what cost and what are the tradeoffs for this level of security? In a recent performance evaluation of the Axis2 web services engine the test results proved that plain text was the fastest message transfer method. Although this was expected, the test results also concluded that Secure Socket Layer was only marginally slower than plain text and that it outperformed the Web Services Security implementation of Kerberos on Axis2 by a large extent. These results identify that Kerberos is expensive in terms of performance cost. It is a factor to consider during the design of DEMAC. If feasible SSL is preferred for direct client - server communications where authentication concerns are not necessary, this trade off also should be considered during implementation. The other limiting factor of Kerberos is that it is not suitable over multiple security domains. Work has been proposed for the use of Public Key Infrastructure with Kerberos which would alleviate this as a concern, but for this implementation, Kerberos performance costs withstanding is effective for meeting the security requirements.

5.4.1 Example building system network

In order to explain the application design, it is necessary to articulate how the network of a smart building system (SBS) executes. There are three levels of the SBS, at the highest level is the SBS management software. The next level down is the communications controllers which manages multiple devices and or zones. The lowest level is the interaction between the devices (e.g. HVAC fan coil) and the controller which it is attached to - a field controller. A building may contain many hundreds of communications controllers and a multitude more field controllers (perhaps 10 or 20 times more). The communications controllers are accessible on the Ethernet network of the building. Each communications controller has its own unique address, and share the same network. A number of field controllers (e.g. 15) are in turn controlled by one communications controller. The field controllers operate on their own sub network, each having a unique address. See Figure 5-6 High level architecture diagram of SBS network for a network of controllers and devices.

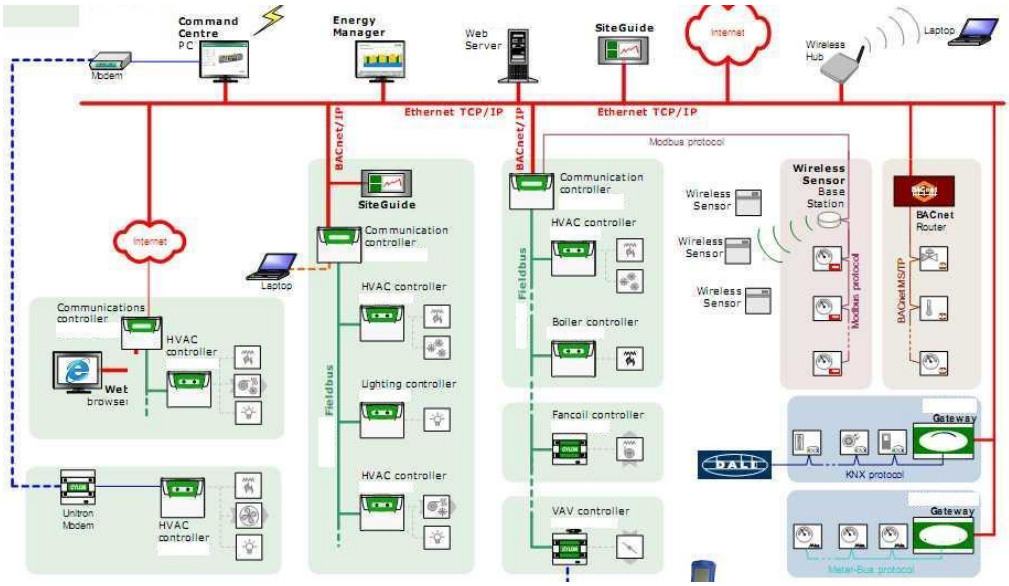

Figure 5-6 High level architecture diagram of SBS network

The field controllers are those controllers which directly control the building environment or system (e.g. Lighting, Heating). The field controllers receive data in both analogue and digital formats. The data is received directly from input signals from sensors in the building environment or indirectly from another field controller. The controller calculates an output operation based on these inputs. The output operation is determined by the controller's strategy, the strategy is a program executing on the controller itself. This output was used to

regulate the building environment. The output signal is sent to actuators which operate the plant machinery which in turn regulate the building environment e.g. heating.

Each field controller must be programmed directly for each building environment; this program is referred to as an engineering strategy. This programming is achieved via a PC based computer aided software engineering (CASE) tool. This engineering strategy must be downloaded to each controller via the PC and controller communications network. The field controller's strategy is devised by an engineer specialising in the HVAC industry. The strategy is fundamentally a program of mathematical calculations that the controller executes. In the example of a heating system the calculations are based on the input sensors plugged into the controller (e.g. heat sensors) and output actuators (ignition circuits for gas boilers, water pumps, fan coils). The goal is to control the heating system based on the temperature of a room, and the necessity of heating or cooling the room via a fan coil or other heat device. The strategy creation is graphical for the engineer on the engineering application and BMS PC. See figure 5.4 for a small strategy example.

The Engineering strategy has 5 principle components:

1. It is a graphical representation of input and output points connected to a mathematical module.
2. The points are unique for each field controller.
3. Points may be virtual and broadcast from one field controller to another via a communications controller.
4. The modules and points can be edited and saved multiple times.
5. The strategy is saved in a format that can be downloaded and executed on the controllers.

The points in a strategy are a means for data pertaining to the SBS system to flow between devices and sensors. Reading this data gives a user an indication of the operation of the system. By changing a value of one of these points (engineering users are qualified to interpret the points, strategies and devices) then changes to the operation of the system is put into effect.

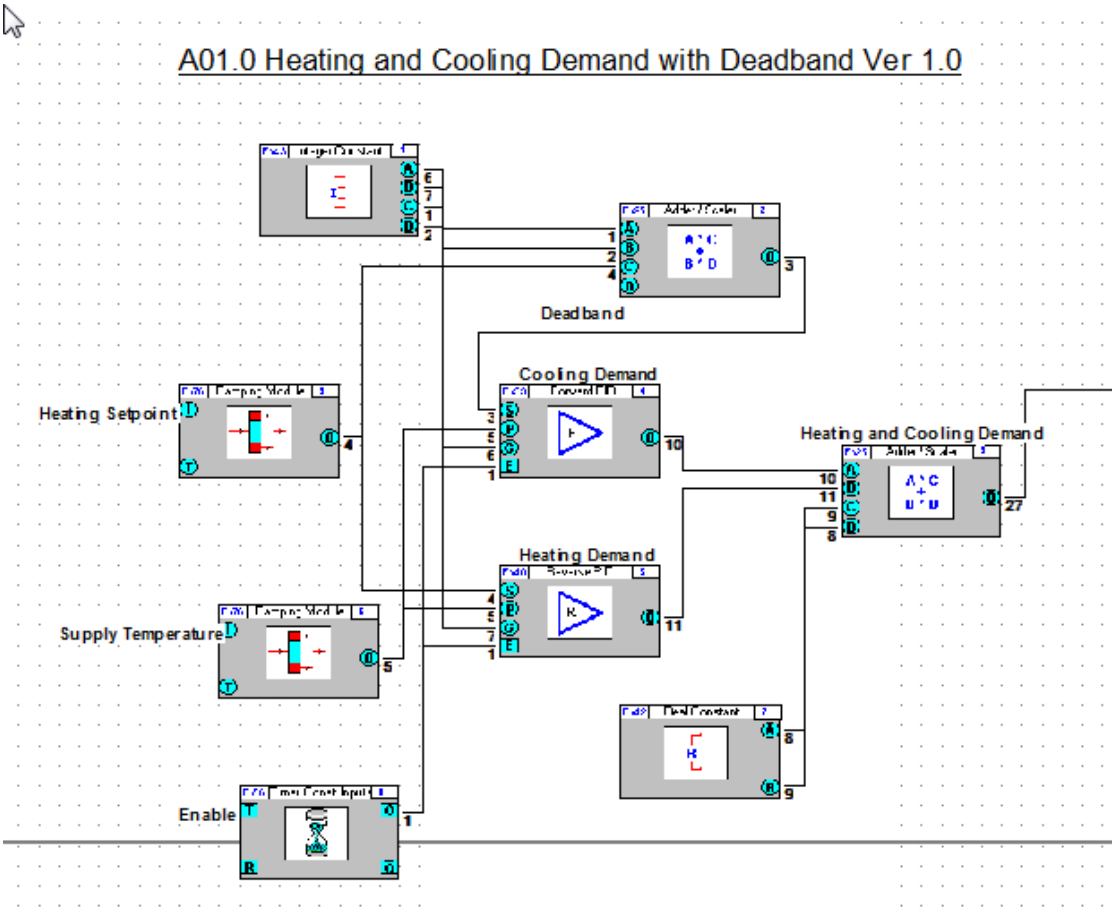

Figure 5-7 Example strategy for heating and cooling

6 Implementation

6.1 DOG 2.0 sample application and evaluation for DEMAC

The initial implementation was concerned with downloading, installing and evaluating the DOG 2.0 framework for DEMAC purposes. Following from the digest of the installation guide, the installation of the DOG 2.0 complete bundle (which contains all drivers and a sample house application) was executed but it failed to start satisfactorily. The drivers and configuration files which were attempted to be loaded by the application failed to load. The state of the application was that of continual attempts to load missing modules. The removal of the configuration files and drivers to allow a 'core' start only proved unsuccessful. It transpires that the DOG 2.0 'all' bundle jar files contained hard coded drivers and configurations that would always be required for a start. To circumvent this dilemma the 'core' bundle which did not include any drivers or network configurations was installed. This started successfully.

The next step was to evaluate the components of the framework and to download the project workspace (for Eclipse), source code and documentation and to build the framework. Following from a built framework the necessary sample applications and drivers were added. Unfortunately the eclipse workspace was not a single project within a workspace but a project for each and every modular component (jar file) for DOG 2.0. The DOG 2.0 framework was proving to be more problematic than had been expected. In order to build one project, the projects dependencies had to be determined and then built in the required order and then subsequently build each parent project. This was proving to be quite time consuming and unfortunately all projects could not be built but a sufficient number to allow me to begin an understanding of the framework and the API.

Having successfully built a sample project, a working application was available for use as a starting point for the evaluation of DOG 2.0. This application would serve as the web server or web server interface to the user. The code and java doc proved to be invaluable to get an understanding of the application logic and API. A network agnostic DOG 2.0 module which registered a device using the DOG 2.0 API was built. It could publish a message containing information relating to the status of the device and its operation. The method of subscribing

devices and subsequent scheduling of their status was well coded and structured. External to DOG 2.0 the next development item was an application which could connect to DOG 2.0 retrieve a list of devices, and select a device and subscribe to DOG 2.0 to retrieve message updates at regular intervals, this application was successful, but the messaging retrieval was quite slow.

The use of XML while slow would be of benefit with the network interface driver which was the next item to be developed. The DOG framework documentation was examined for further information on how to develop the driver. The documentation for DOG was incomplete with only information relating to messaging. There was no documentation on network drivers, how they are to be configured or where to even start. The existing drivers were also not documented. The example devices that were bundled with DOG were also not documented, further analysis of the code revealed that the devices while simulating sensors were hard coded with values. There was no easy way in which the applications could connect DOG 2.0 framework with an actual device or network of devices.

As a result of the development the conclusion arose that DOG 2.0 was not fit for purpose and was not going to be used in DEMAC. Fortunately the scheduling and publish subscribe messaging of DOG 541T proved beneficial, in that it proved useful for a similar approach to be taken in SOA DEMAC.

6.2 SOA DEMAC

6.2.1 Development Environment

The starting point for the SOA DEMAC implementation took off from where DOG 2.0 finished, that was to get a sample application working to prove the architecture design and then to extend its functionality to all design components and ultimately to implement all requirements. The high level overview of DEMAC is depicted below. This development networked environment was set up with a Tomcat Web Server representing the building and a LAN with Internet connectivity representing the Internet and a Firefox [1]browser client hosting the user interface. A MySQL server was co-hosted on the same computer as the Tomcat Server. In addition to this setup was a smart building system hosted in a commercial building some 20 miles away, access to this smart building system was via the internet. This setup was a realistic configuration to implement and evaluate the application.

The actual environment is depicted at a high level in Figure 6-1 DEMAC High level Overview and in more detail in Figure 6-2 DEMAC System level architecture; this level of detail indicates what components make up the DEMAC application. The choice of application server and data base server was based on Open Source licensing and supporting available documentation.

[1] Firefox is a browser client developed by the Mozilla open source community, www.mozilla.org

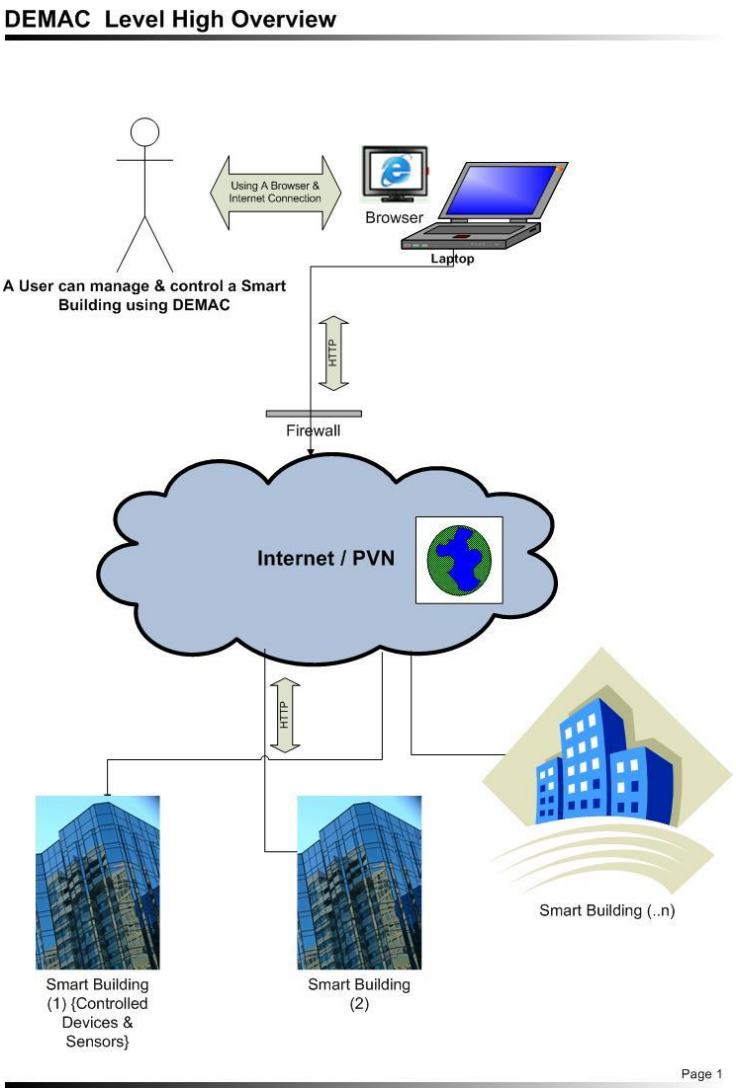

Figure 6-1 DEMAC High level Overview

The development life cycle was an iterative development lifecycle and also a top down affair. Initially the high level components were developed first. Subsequent development iterations saw additional functionality added until the application development was completed.

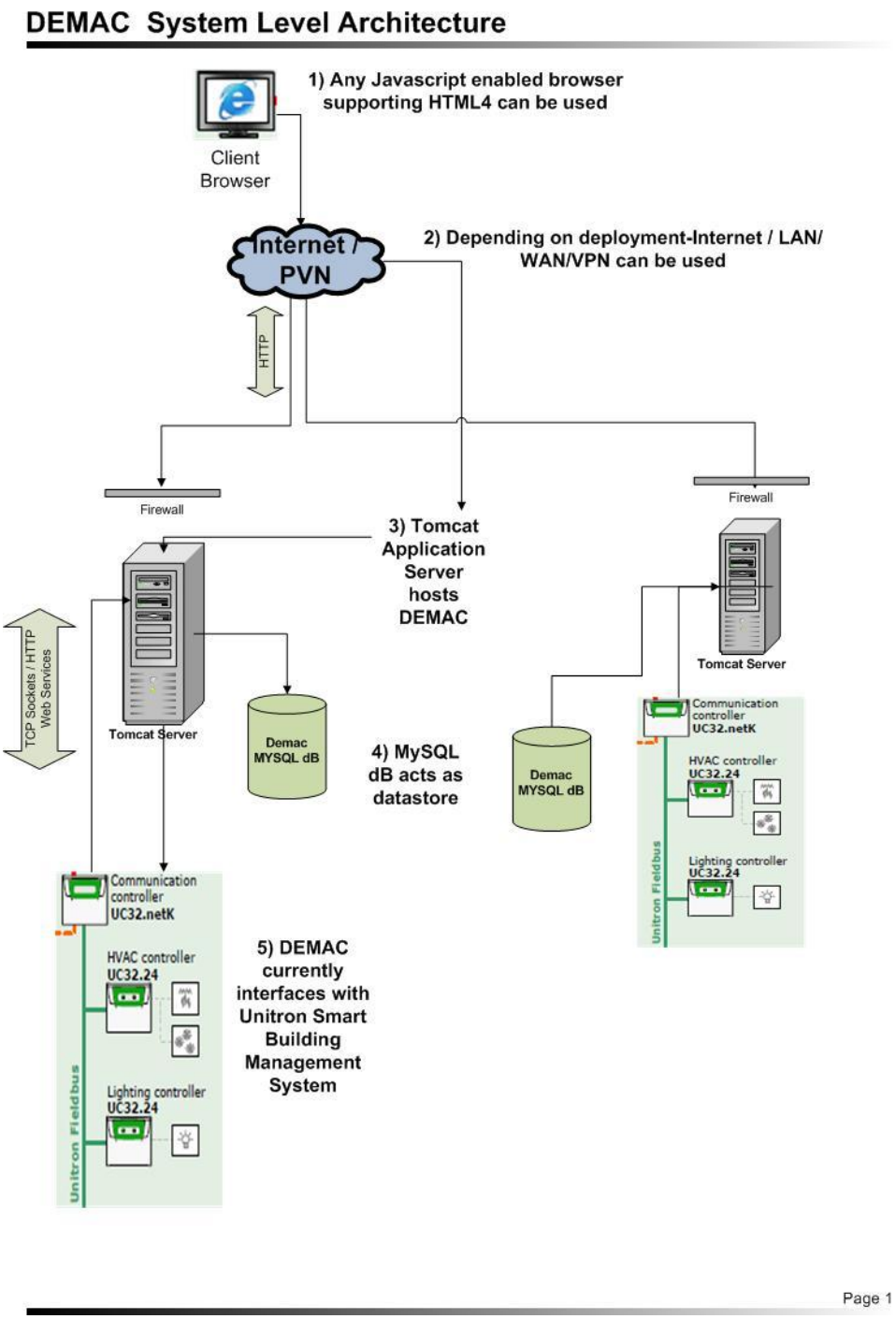

Figure 6-2 DEMAC System level architecture

6.2.2 Development Components

The packages which would house the main components and associated classes were devised. With regards developing the code, the components and their responsibilities from the design were implemented in this regard. Figure 5.9 indicates the package structure. The

classes for the Network Interface, DB Manager, and Application Server service were developed and tested in this order. Lastly the DB Schema and building objects were developed.

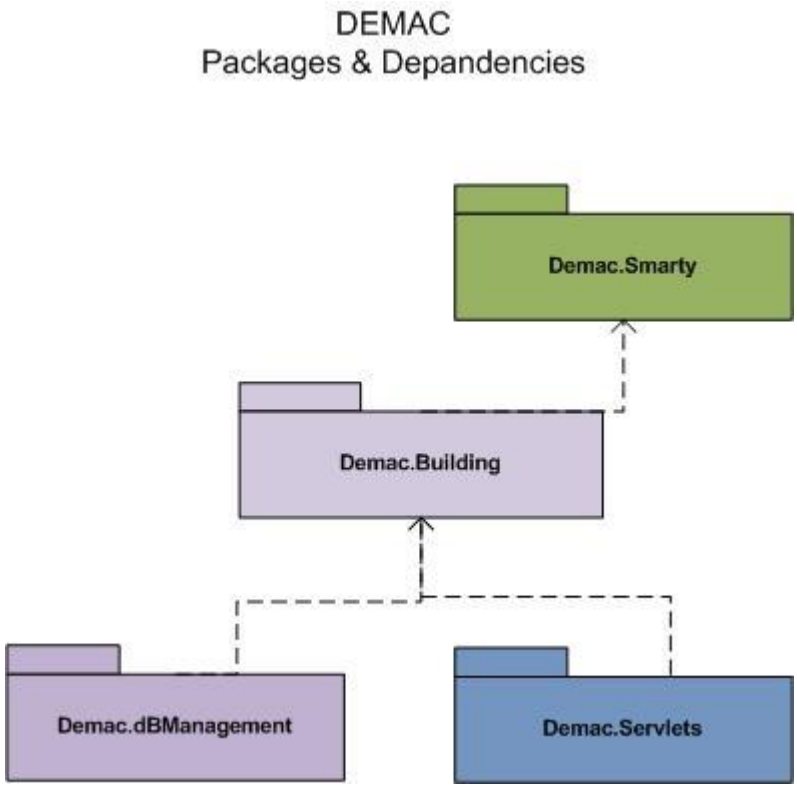

Figure 6-3 DEMAC Packages

The sub classes / helper classes are not discussed in detail here. Figure 6-3 DEMAC Packages illustrate the actual organisation and responsibilities for the main components and classes.

6.2.3 Smart Building System Network Interface

The Smart Building System network interface was developed with information derived from the actual API of the building system. In order to obtain this API of this system, I signed a legal Non Disclosure Agreement (NDA) with the company. This NDA prohibits me from disclosing any commercially sensitive information. For this reason I must be frank with the details. Within the class, private variables were set up to build the correct messages for interacting with the actual ‘controllers ‘and connecting devices.

The interface that I developed allowed a number of methods to interact with the controllers based on their address details and the details of the connecting devices.

The methods were appropriately named and accepted arguments, the method would either retrieve information from the SBS and or dispatch information to the SBS in order to execute a change to the system. These methods were then wrapped with generic methods for interfacing with any network interface for SBS. This wrapper class was then developed further into Web Services. This implementation would allow the DEMAC system to be interoperable with other SBS using the same Web Service interface.

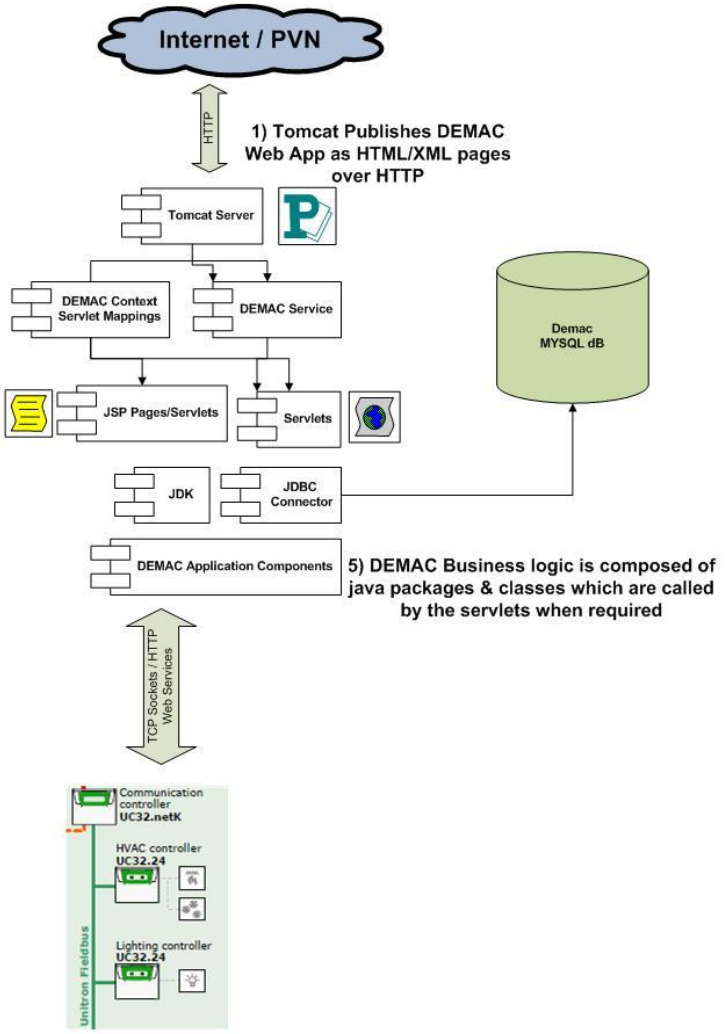

Figure 6-4 DEMAC Application Components

6.2.4 Database Manager and Schema

The Database manager is the interface between the application server service and the data store. Its core responsibilities are to do with accepting requests from the service and making the necessary DB connections, constructing DB Queries and joining tables where necessary in order to retrieve information from the DB. The data retrieved from the DB is constructed into the appropriate building objects and returned to the calling service component. The navigation of tables, result sets and connection pooling is managed by the manager component object.

The actual data model is hidden from the DEMAC service but appropriately named methods and objects are used to ensure that if required the objects can be sub classed and substituted if required for extending the DEMAC application with future Network Interfaces or additional features.

The schema is depicted in Figure 6-5 DEMAC schema, Where joins are required for extracting the data from multiple tables, foreign keys are used on referenced tables (indexes). This ensures data integrity on insertions and speeds up data retrieval.

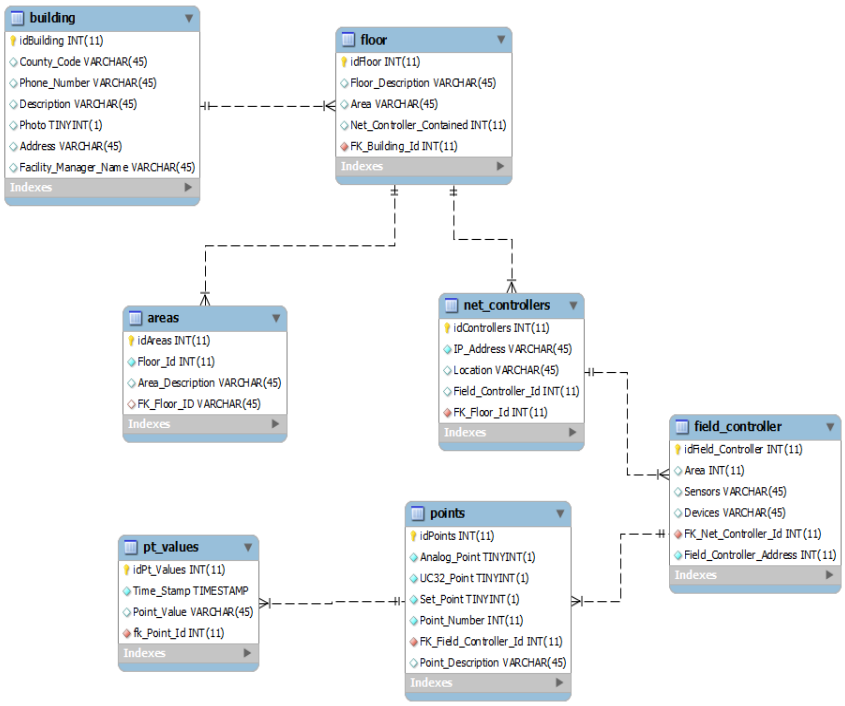

Figure 6-5 DEMAC schema

The structure of the schema is building independent and catalogs any building into a hierarchy of floors, area and controlling devices within areas. The description and points of the devices are also captured. The location and description of each zone and layers which represent a zone are captured so that zones may be targeted within buildings and their contained derives also.

The database manager component can only be used with a ‘database details’ object which contains the DB Uri, user name and password. This adds a layer of security protection to the system and prevents unauthorized access and changes to the system.

6.2.5 Data Logger Component

The purpose of this component is to capture historical information relating to the SBS execution and store it at predefined intervals for future report and analysis purposes. The ‘data logger’ initially obtains device information and a scheduling time (in a custom number of seconds e.g. 300) from the user. The component subsequently interfaces with the appropriate network interface component (based on device information) to retrieve live status messages from the device. The device information is (via the DB manager) stored in the data store. The data logger component repeats these tasks at every regular (e.g. 300 seconds) interval.

6.2.6 Data SVG Graph Component

This component is responsible for retrieving the historical data that the Data logger records. Once again interfaces to the DB manager are required. The data is dynamically graphed on a line chart with the values stored on the Y-Axis and the time intervals on the X-Axis. A Scalable vector graph is drawn which by its name is scalable in a browser to allow zooming in on certain points for clarity. Figure 6-6 SVG line graph illustrates one graph.

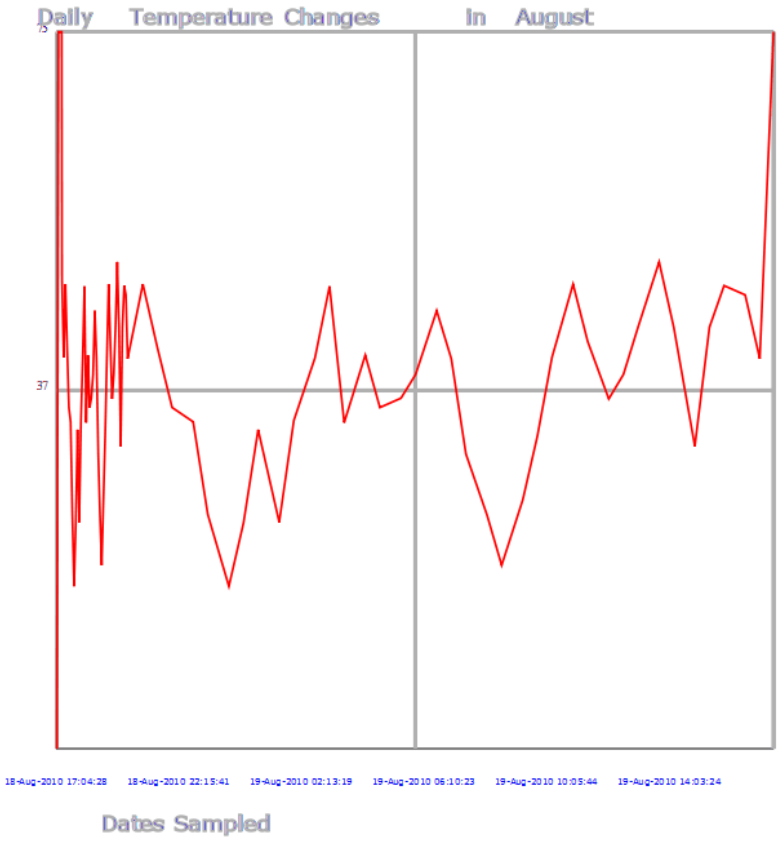

Figure 6-6 SVG line graph

6.2.7 DEMAC Service (Servlets)

The DEMAC service executes on the web application server and it interfaces with each of the aforementioned components. It is comprised of three important sub components the Servlet classes, the Java Server Pages (JSP) and the Servlet mappings. Both the JSP and Servlets classes effectively are UI interfaces which return html to the user, but they also act as server backend components. The JSP are essentially compiled into Servlets at runtime. I will refer to both items as Servlets for simplicity. The Servlet context is contained within the Deployment Descriptor. It essentially maps user URI's to Servlets. In DEMAC the URI effectively maps the user path through the application and this corresponds to how the application maps determines what objects are necessary and what methods are invoked in the backend classes.

The list of Servlets / JSPs correspondingly are mapped to the following URI's

./Building/

./Building/Floor

./Building/Floor/Area

./ Building/Floor/Area/NetworkController

./ Building/Floor/Area/ NetworkController /FieldController

./ Building/Floor/Area/ NetworkController /FieldController/Points

./ Building/Floor/Area/ NetworkController /FieldController/SVG

The Servlets use session information and request attributes to maintain state and to assist with user navigation changes. The zone, device information is important to ensure that information and changes are made to the correct device and zones.

Where user interaction is required within a Servlet URI (the same page that the user sees) embedded hyperlinks, html forms and JavaScript functions are used. These client side functions are used to interact with the user and when the user has completed their actions the URI & parameters are submitted to the server for server side action. In most cases a

change of URI is invoked (using a GET method call) to a Servlet which then deals with the request.

In Figure 6-7 a Use Case of a facility engineer retrieving zone information from the system is displayed. In this use case, device information may be retrieved from the system. During the user query of the system the 'Points' Servlet is invoked (in addition to others). In the case of the 'Points' Servlet, the devices, addresses, names and details corresponding to a FieldController are obtained from the DB Manager and their real time values from the network interface and returned to the user. Should the user wish to view historical data related to a device they invoke the SCG Servlet and are returned an SVG graphic containing all historical data. Should a user wish to change a device value they invoke a java script function which performs two actions, it opens a pop up window and retrieves and alternate device values following from a submit by the user the function submits these new values to the device via the network interface component. Figure 6-8 DEMAC Servlets and JavaScript operations illustrates this Servlet and java script operations.

Each Servlet returns html over http. The content of the html is a mixture of static and dynamic content. The dynamic content is dependent on the actions of a user in previous URI /Servlets. To reiterate, parameters and session information area used to maintain state should a user navigate away from the present URI for consistency and security of the building system.

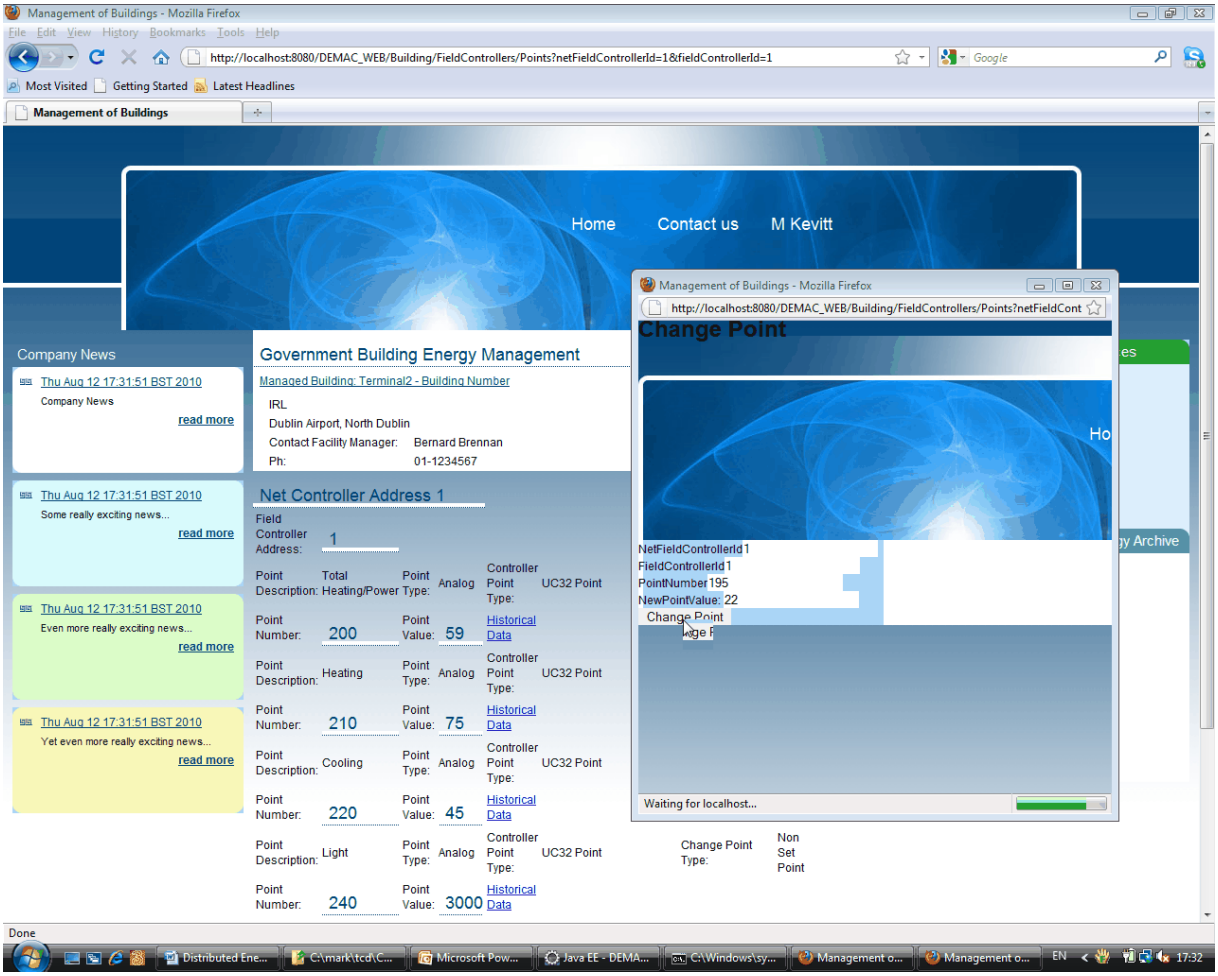

Figure 6-8 DEMAC Servlets and JavaScript operations

6.2.8 User Interface

The user interface is what is presented to the user in the browser from the Servlets. Structurally the UI is separated into five different components header, left, right, main and footer. Each component has a separate purpose and by only presenting updated components to the browser the network traffic is reduced.

Figure 6-9 DEMAC User Interface below illustrates the UI. A style sheet is used which retains the same look and feel throughout the application navigation. The header and footer remain static throughout but the left, main and right components are altered depending on user navigation. Figure 6-10 Device and device strategy information illustrates what a user may require with regards to device information. In this screenshot a heating and cooling device is displayed with an associated strategy. As previously stated java script executes on the client UI where user interaction is required. See figure 5.13 for an example popup window.

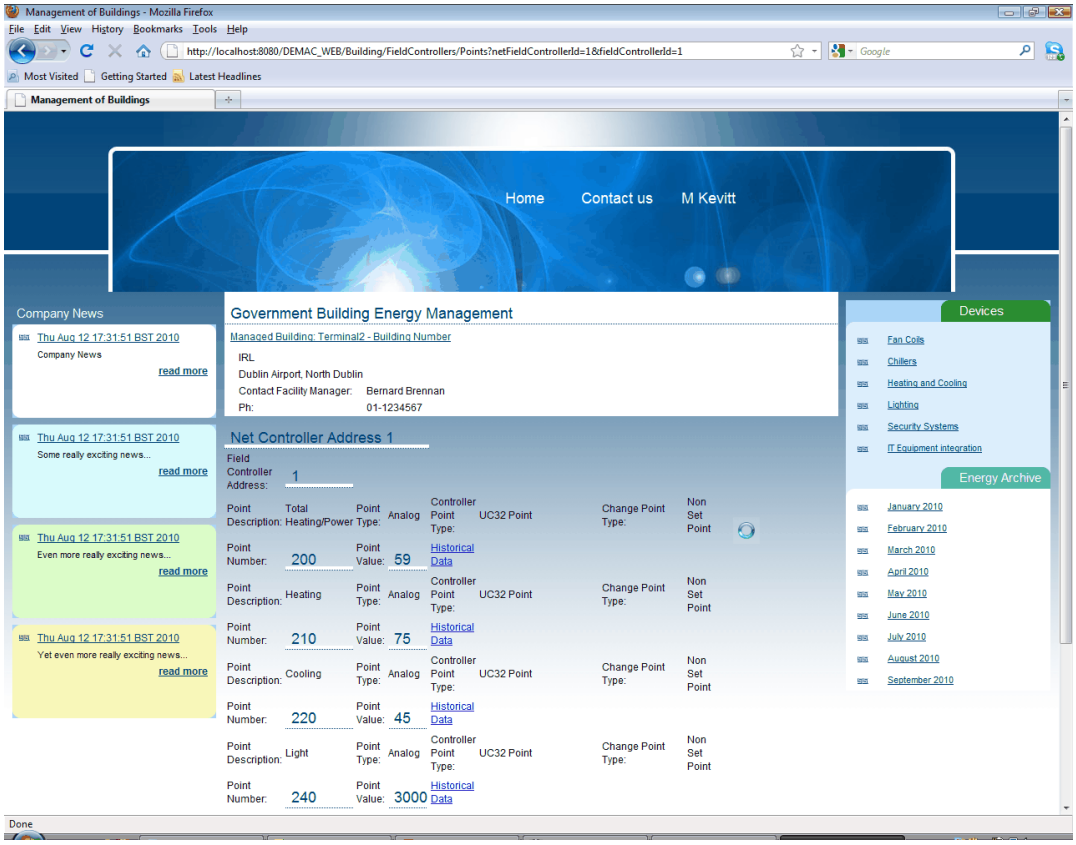

Figure 6-9 DEMAC User Interface

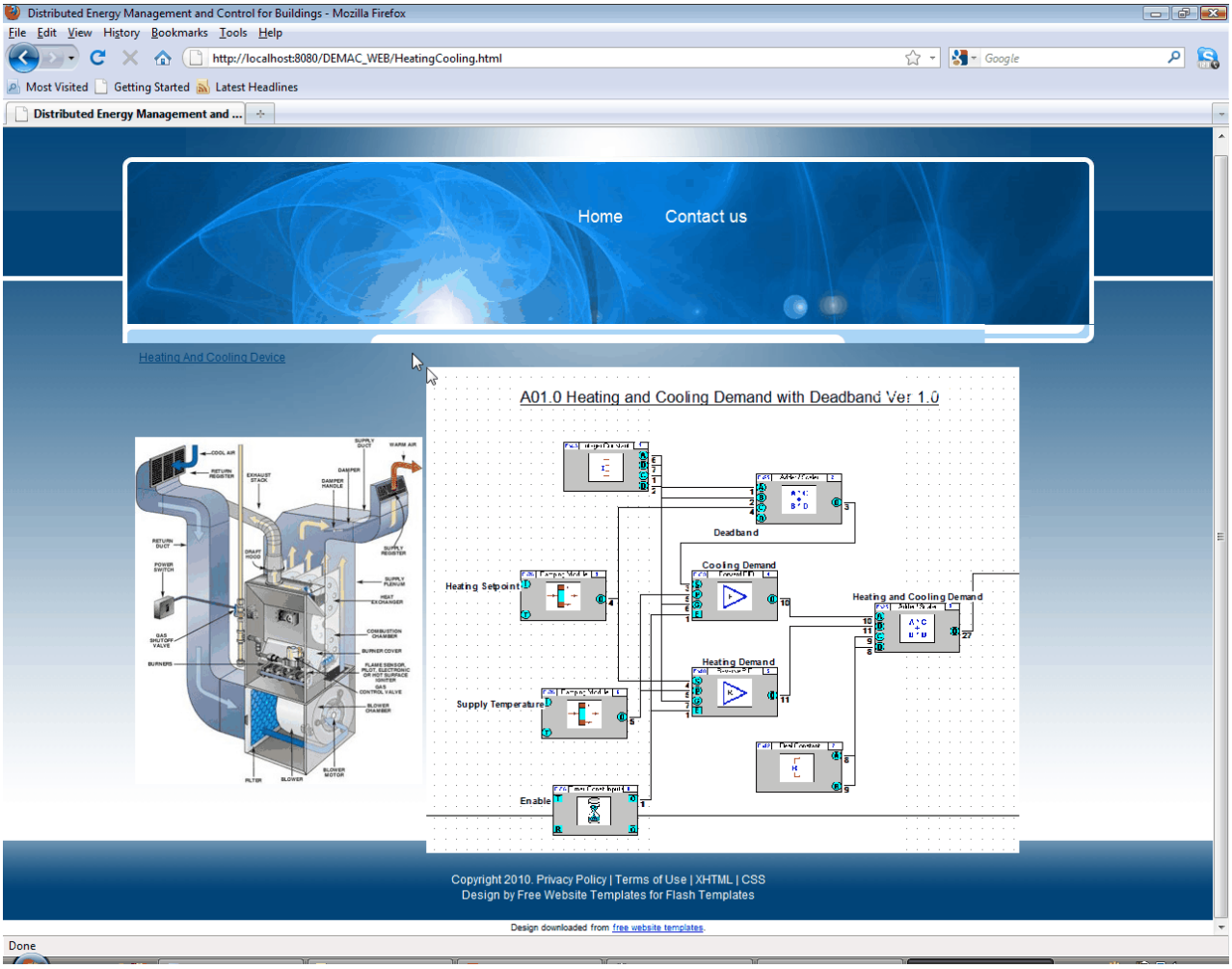

Figure 6-10 Device and device strategy information

7 Evaluation

DEMAC fulfils 14 out of 16 requirements, consult Table 7-1 DEMAC Requirements for a complete list. The two requirements number 3. Secure access to zones and number 5. Automatic discovery of intelligent building devices are not fulfilled. The primary aim of the thesis was focused on interoperability and the secondary on scalability. These aims were necessary while maintaining requirements of being highly available and accessible remotely in real time. These objectives were met. The objectives are explained in more detail in teh following three sections.

7.1.1 Interoperability

The architecture of DEMAC is designed to be able to interface to different network smart building systems. The component structure and use of wrapper classes facilitates the 'plug in' of different networks seamlessly. The use of web services means that existing network infrastructure can be used; firewalls may be penetrated using http traffic. The additional benefit of using web services is that additional network interfaces can be written for protocols which currently do not offer an interoperable interface. BACNet for example may be interfaced to DEMAC in this manner.

The building structure is modeled in the relational DB schema; this schema may be read by other SBS which may require access. This ability can assist with interfacing DEMAC with existing SBS software packages.

The web application user interface which is a portal in design and architecture allows for web applications plug-in or modifications without altering any code. This ability aids the extension of DEMAC to perhaps allow other SBS software systems to execute in conjunction with DEMAC. The left component perhaps may display other software systems information and act as its interface. Other possibilities include making additions to the Servlet URI where additional features may be added. This doesn't impact existing features.

Wong et al. developed an enterprise bus in which communication between device objects can take place at three levels of abstraction; DEMAC has component based architecture with web services interoperability. This object interchange currently is not developed, however DEMAC supports this development. DEMAC may be extended to include this functionality. A simple plug-in component may be deployed in the application server which permits this object's communication to happen.

Xiao developed an SOA architecture using web services which permitted BACNet and other open protocols to inter communicate. DEMAC has a similar architecture which permits this interoperability. What Xiao developed was targeted at interoperability but failed to capture a scalable solution which was also portable to multiple buildings.

The IEMN solution utilizes BACNet exclusively and also uses SOA three tier architecture as does DEMAC. The DEMAC solution can also be extended to incorporate open standard object communication such as BACNet and IEMN.

7.1.2 Scalability

The use of a relational DB greatly improves the vertical scaling of the application to capture large buildings SBS. It would not be feasible for the use of xml files for this purpose. The schema was designed with small tables with compact SQL using joins where necessary and foreign keys on referenced table indices. This model is scalable in size without impacting on performance. Xias's architecture did not use a relational database to capture the structure of a building which would detract from a scalable solution.

The use of style sheets, different static and dynamic components on the client reduces network traffic and improves latency between content changes. By selecting off the shelf application servers and web technologies allows for scalability in performance by using enterprise hardware and or software where required to grow DEMAC in scale.

Horizontal scalability is also achieved by having each building host its own application server. This allows clients to connect to buildings from any web enabled device without impacting network performance or creating bottlenecks.

7.1.3 Performance

The architecture design of DEMAC for scalability also aids to its brisk performance. The use of http is significant in optimizing network bandwidth. The use of XML is minimal to reduce the overhead of parsing. During performance testing of DEMAC, the network interface was able to retrieve six points in 1.2 seconds; this is the throughput from the SBS. This was not using the DEMAC application server or UI. This is an indication of the latency of point retrieval from the SBS to the application.

When DEMAC was included in the testing the same six points were retrieved in 2.12 seconds. Subtracting the latency of the SBS retrieval from DEMAC yields a metric of six points in 0.9 seconds.

Further tests proved that these times were consistent. Additional points were retrieved to determine if there was a degradation of the system with increased load, but 60 points did not alter this performance throughput.

The message format in DOG 2.0 was its own proprietary DOG 2.0 XML and despite the fact that pull parsing was used, it took 1.25 seconds to connect to DOG and retrieve a device message. If it took over one second for one message, it would not scale well. Adding nine additional devices and requesting additional messages improved with 3.4 seconds for 10 requests and messages, but it was still proving to be slow for use in DEMAC.

8 Conclusion and Further work

DEMAC fulfils 14 out of 16 requirements. The application achieves interoperability between heterogeneous networks and is extensible to support other network smart devices. It offers a horizontal and vertical scalable solution to building systems. It builds on existing research into smart building systems and offers improvements in scalability and interoperability while retaining the benefits of other applications. DEMAC offers simplicity and usability while maintaining a significant performance benefit during operation. Extensibility is another feature which the architecture and application solution of DEMAC offers.

While DEMAC fails to offer what other research applications (Automatic discovery of intelligent building devices - IEMN & Object interaction) have, it has the platform which can be extended to deliver these features. The automatic discovery of objects and their interaction can easily be plugged into the application server. In fact the inclusion of a relational database will offer improvements in this regard as once objects have been discovered and their operations recognised, they can be stored in the database with their zone information. This will reduce network traffic and reduce the number of broadcast messages in the system. A generic search could replace the need for broadcasts and discovery by developing a database search and discovery mechanism which responds to Objects broadcast messages. This further work could have significant contributions to intelligent building interaction. Another potential benefit and avenue for this extension would be the inclusion of web semantics to objects. A semantic web data store with a rules engine backend could yield immense benefits in inter system communications and smart device interaction.

The other outstanding feature and item which is lacking in DEMAC is the inclusion of web application server and web services security and encryption. While encryption is not a significant drawback the lack of user authentication to zones and systems could have serious consequences particularly as the system is intended to operate over the internet. This outstanding feature is an obvious future enhancement which unfortunately development time did not permit. Perhaps if DOG 2.0 was not pursued it would have been included offering most requirements.

9 Bibliography

[1] A. Anderson. Web services policies. *Security Privacy, IEEE*, 4(3):84 –87, may-june 2006.

[2] J. Bai, H. Xiao, T. Zhu, W. Liu, and A. Sun. Design of a Web-based Building Management System using Ajax and Web Services.

[3] Paolo Baronti, Prashant Pillai, Vince W.C. Chook, Stefano Chessa, Alberto Gotta, and Y. Fun Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications*, 30(7):1655 – 1695, 2007. Wired/Wireless Internet Communications.

[4] Damien Cassou, Benjamin Bertran, Nicolas Loriant, and Charles Consel. A generative programming approach to developing pervasive computing systems. In *GPCE '09: Proceedings of the eighth international conference on Generative programming and component engineering*, pages 137–146, New York, NY, USA, 2009. ACM.

[5] Han Chen, Paul Chou, Sastry Duri, Hui Lei, and Johnathan Reason. The design and implementation of a smart building control system. In *ICEBE '09: Proceedings of the 2009 IEEE International Conference on e-Business Engineering*, pages 255–262, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Eran Chinthaka. Axis2 architecture, 2006. This is an electronic document. Date of publication: November 9, 2006. Date retrieved: March 2, 2010. Date last modified: [Date unavailable].

[7] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86 –93, mar/apr 2002.

[8] P.T. Eugster, R. Guerraoui, and J. Sventek. Distributed asynchronous collections: Abstractions for publish/subscribe interaction.

[9] Paul Grace, Gordon Blair, and Sam Samuel. A marriage of web services and reflective middleware to solve the problem of mobile client interoperability. In *ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies*, pages 506–511. Trinity College Dublin, 2003.

[10] S. Graham, D. Davis, S. Simeonov, T. Boubez, R. Neyama, and Y. Nakamura. Building Web Services with Java: Making Sense of XML. *Soap, Wsdl, and Uddi, Sams, Indianapolis, IN*, 2001.

[11] T. Gu, H.K. Pung, et al. Toward an OSGi-based infrastructure for context-aware applications. *IEEE Pervasive Computing*, pages 66–74, 2004.

[12] LK Haakenstad, A.T. Inc, and WA Redmond. The open protocol standard for computerized building systems: BACNet. In *Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on*, volume 2, 1999.

[13] H.Y. Huang, J.Y. Yen, S.L. Chen, and F.C. Ou. Development of an intelligent energy management network for building automation. *IEEE Transactions on Automation Science and Engineering*, 1(1):14–25, 2004.

[14] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[15] K.S. Kim, C. Park, and J. Lee. Internet Home Network Electrical Appliance Control on the Internet with the UPnP Expansion. In *Proceedings of the 2006 International Conference on Hybrid Information Technology-Volume 02*, pages 629–634. IEEE Computer Society, 2006.

[16] R. Kistler, S. Knauth, and A. Klapproth. UPnP in Integrated Home-and Building Networks.

[17] M.J. Lee, J. Zheng, Y. Ko, and D.M. Shrestha. Emerging standards for wireless mesh technology. *IEEE Wireless Communications*, 13(2):56, 2006.

[18] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan. An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks. In *33rd IEEE Conference on Local Computer Networks*. Citeseer, 2008.

[19] Desheng Li, Haiyang Wang, Lizhen Cui, and Kanglin Gao. Collaborative design of web services composition process based peer-to-peer architecture. pages 525 –530, april 2007.

[20] Hongqi Li and Zhuang Wu. Research on distributed architecture based on soa. pages 670 –674, feb. 2009.

[21] Tokunaga E. Ishikawa H. Ueno D. Kurahashi M. Iwasaki K.-Nemoto M. van der Zee A. Nakajima, T. Software infrastructure for building large-scaled smart environments. pages 82 – 89, Jan. 2003.

[22] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner. Web services in building automation: Mapping knx to obix. volume 1, pages 87 –92, june 2007.

[23] oasis open.org. Internet Resource for Open standards. *http://www.oasis-open.org/committees/obix/faq.php*, 2010.

[24] T.J. Park, Y.J. Chon, D.K. Park, and S.H. Hong. BACNet over ZigBee, A new approach to wireless datalink channel for BACNet. In *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 1, 2007.

[25] Kuang-Yu Peng, Shao-Chen Lui, and Ming-Tsung Chen. A study of design and implementation on soa governance: A service oriented monitoring and alarming perspective. pages 215 –220, dec. 2008.

[26] S. Perera, C. Herath, J. Ekanayake, E. Chinthaka, A. Ranabahu, D. Jayasinghe, S. Weerawarana, and G. Daniels. Axis2, Middleware for Next Generation Web Services. In *Proceedings of the IEEE International Conference on Web Services*, page 840. IEEE Computer Society, 2006.

[27] Thinagaran Perumal, Abd Rahman Ramli, Chui Yew Leong, Khairulmizam Samsudin, and Shattri Mansor. Middleware for heterogeneous subsystems interoperability in intelligent buildings. *Automation in Construction*, 19(2):160 – 168, 2010.

[28] I. Pusnik. Energy efficiency of buildings. *Elektrotehniski Vestnik*, 74(5):248–254, 2007.

[29] C. Reisse, C. Burghardt, F. Marquardt, T. Kirste, and A. Uhrmacher. Smart environments meet the semantic web. In *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, pages 88–91. ACM New York, NY, USA, 2008.

[30] D. Saha and A. Mukherjee. Pervasive computing: a paradigm for the 21st century. *Computer*, 36(3):25 – 31, mar 2003.

[31] C. Schropfer and M. Schonherr. Introducing a method to derive an enterprise-specific soa operating model. pages 235 –244, sept. 2008.

[32] Hong Seong Park Daeyoung Kim Young-joo Kim Seong Hoon Kim, Jeong Seok Kang. Upnp-zigbee internetworking architecture mirroring a multi-hop zigbee network topology. *Consumer Electronics, IEEE Transactions on*, 55(3):1286 –1294, August 2009.

[33] D. Snoonian. Smart buildings. *IEEE Spectrum*, 40(8):18–23, 2003.

[34] W3C. Web Services Architecture. *http://www.w3.org/TR/ws-gloss/*, 2002. (www.greenintelligentbuildings.com, 2005)

[35] JKW Wong, H. Li, and SW Wang. Intelligent building research: a review. *Automation in Construction*, 14(1):143–159, 2005.

[36] J. Yu and P. Lalanda. Integrating UPnP in a development environment for service-oriented applications.