DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Adaptive Context-Aware Control and Monitoring System for Smart Environments

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5511: Projeto de Fim de Curso

Rodrigo Schmitt Meurer

Florianópolis, Março de 2017

Adaptive Context-Aware Control and Monitoring System for Smart Environments

Rodrigo Schmitt Meurer

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Dr. Antônio Augusto Medeiros Fröhlich

____

Assinatura do Orientador do Local de Trabalho

Prof. Dr. Jomi Fred Hübner

____

Assinatura do Orientador da UFSC

Banca Examinadora:

Prof. Dr. Antônio Augusto Medeiros Fröhlich
Orientador na Empresa

Prof. Dr. Jomi Fred Hübner
Orientador no Curso

Prof. Dr. Hector Bessa Silveira
Responsável pela disciplina

Prof. Dr. João Carlos Espíndola
Avaliador

Andjara Consentino
Debatedor

Murilo Ramos Carraro
Debatedor

Acknowledgements

After an intensive period of six years: writing this note of thanks is the finishing touch on my monography. It has been a period of intense learning for me, not only in the scientific field, but also on a personal level. I would like to reflect on the people who have supported and helped me so much throughout this period.

To Professor Jomi Fred Hübner, academic advisor, with indisputable competences, who in the same way shared his extensive knowledge throughout the development of this project.

To Professor Antônio Augusto Medeiros Fröhlich, workplace advisor, that awakened and fostered my interests in the fields of embedded systems and Internet of Things, and always accompanied my projects carried out in the Software/Hardware Integration Lab, sharing his knowledge and experience and for enlightening me the first glance of research.

To my mother Dione, my Father Sandro and my brother Rafael for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this document. This accomplishment would not have been possible without them.

To my girlfriend Samara, for the support in all the moments of this project, from backing my decisions to putting up with my in the moments of stress.

I thank my fellow labmates in the Software/Hardware Integration Lab for the stimulating discussions and for all the direct or indirect help.

Finally, I would like to thank my friends for their friendship and support, that during this period made the long hours of study lighter and more fun, and for the numerous memories that we made together.

Resumo Estendido

Com o advento da internet das coisas (IoT), existem dispositivos que podem se comunicar com a internet enviando e recebendo dados. Muitos desses dispositivos são utilizados em casas, salas, escritórios para prover um melhor conforto ou até poupar energia e tempo dos usuários. Um sistema de controle para ambientes inteligentes deve automaticamente ajustar seus parâmetros e controlar seus dispositivos, para as preferências do usuário, baseado em dados previamente coletados dos sensores de dispositivos inteligentes que apontam o comportamento e interação do usuário com o ambiente. Esse trabalho descreve o desenvolvimento de um sistema que leva os atuais requerimentos da Internet das Coisas em consideração.

Uma solução para ambientes inteligentes da internet das coisas, precisa contar com vários módulos que desempenham funções distintas. Para implementação de um ambiente inteligente, esse projeto considera quatro componentes principais. Primeiramente, uma sala inteligente onde existem vários sensores e atuadores disponíveis para controle das distintas variáveis do ambiente. Um *gateway* inteligente que faz a ponte entre os dispositivos da sala e a internet, além de detectar usuários. Um aplicativo híbrido que permita o usuário controlar e monitorar o sistema remotamente através da internet, recebendo em tempo real os dados provenientes de sensores do ambiente. Finalmente, um algorítimo ciente de contexto para controle automático do ambiente.

Para implementação da sala inteligente, foi escolhida uma sala do Laboratório de Integração de Software e Hardware (LISHA) da UFSC. Essa sala já havia sido parcialmente instrumentada com dispositivos inteligentes em projetos prévios de internet das coisas, capazes de medir consumo energético de quatro tomadas e três lâmpadas, além de controlar o ar-condicionado e lâmpadas. Para isso foram utilizados microcontroladores *EPOSMote III*. Esses microcontroladores rodam o sistema operacional *EPOS* e possuem conectividade sem fio *IEEE 802.15.4*. O *firmware* presente nas lâmpadas estava funcional, enviando mensagens *Modbus* do consumo a cada segundo, entretanto, todas as tomadas tiveram seu *firmware* reprogramado para agir da mesma maneira. Foram também desenvolvidos nodos extras para a coleta de dados de luminosidade e temperatura internos e externos, presença e umidade.

Um *gateway* de controle embarcado para internet das coisas estende as funcionalidades de um simples *gateway*, provendo inteligência e recursos para processamento das aplicações locais. O gateway é capaz de filtrar e avaliar as mensagens *Modbus* recebidas dos microcontroladores além de implementar tarefas de gerenciamento de mais alto nível. Esse *gateway* foi implementado na plataforma Intel Galileo 2ª geração.

A placa foi conectada a um microcontrolador EPOSMote III, que recebe as mensagens *Modbus* pela rede *IEEE 802.15.4* e as encaminha através de comunicação serial USB para serem processadas pela placa Galileo. Além da conectividade *IEEE 802.15.4*, a placa também foi conectada via Ethernet a um roteador sem fio *IEEE 802.11*, que além de prover conectividade à internet para a placa, permite que ela esteja localizada na mesma sub-rede a qual o celular dos usuários da sala se conectam. Isso permite que a placa escaneie a sub-rede e identifique os usuários presentes no ambiente. Essa plataforma também possui uma instância de um servidor *HTTP* para estabelecer comunicação entre ela e o servidor. Sensores e LEDs foram adicionados a placa para obter ainda mais informação e fornecer *feedback* do estado dos serviços desempenhados pela placa. Um Daemon integra todas essas atividades executando-as paralelamente com *threads*.

Uma aplicação híbrida, lida com uma questão crítica de projetos para internet das coisas, a portabilidade. Já que a grande maioria dos dispositivos de IoT podem ser remotamente controlados, um usuário deve ter acesso ao controle desses dispositivos da maneira mais conveniente possível. Por esse motivo foi implementado um aplicativo híbrido baseado no *framework Apache Cordova*, que gera aplicativos para diferentes plataformas a partir de uma única implementação. O aplicativo desenvolvido possibilita que o usuário monitore os dados do ambiente em tempo real e controle todos os dispositivos possíveis. Para isso é estabelecido um protocolo de comunicação através de requisições *HTTP REST* com o servidor.

Nesse projeto, os dados coletados de sensores de dispositivos inteligentes são usados para definir contextos e preferências do usuário. Uma combinação de técnicas e conceitos de *machine learning* e *data mining* são utilizados para realizar predições de controle para os dispositivos baseados no contexto atual. Um classificador de floresta randômica é aplicado nos dados coletados da sala e nos dados previamente inseridos para o controle de um dado dispositivo, para obtenção da relevância dos dados para o controle de um dado dispositivo. Feito isso uma rede neural é treinada, baseada nos dados coletados da interação do usuário com a sala, utilizando somente os dados relevantes para o treino. Treinada a rede, o algorítimo faz predições de controle da sala. Os conceitos de *reinforcement learning* são utilizados para atribuir recompensas ou punições as predições da rede, onde uma predição certa (quando não existe intervenção ou ajuste por parte do usuário após uma predição de controle) é recompensada sendo adicionada ao conjunto de dados do usuário, e uma errada (quando o usuário faz alguma alteração para uma dada predição de controle do algorítimo) é punida adicionando as novas preferências do usuário e retreinando a rede. Dessa maneira, o algorítimo continuamente aprende os comportamentos e preferências

do usuário. Ao final das predições, pode ser adicionado um otimizador, com heurísticas de redução de consumo energético. Nesse trabalho foi utilizado um controlador *fuzzy* onde foram implementadas regras de economia de energia no uso de lâmpadas de acordo com a luminosidade.

A implementação dos componentes do sistema foram devidamente validados, avaliando suas respectivas métricas. Para validação e coleta de resultados, o sistema funcionou durante um período de aproximadamente duas semanas na sala inteligente do Laboratório de Integração de Software e Hardware da UFSC. O sistema foi configurado para procurar por apenas um usuário e aprender a controlar uma das lâmpadas da sala de acordo com o comportamento deste. Todos os requisitos funcionais e não funcionais delineados no projeto foram cumpridos.

A utilização de um gateway embarcado para controle se mostrou bastante efetiva. O *gateway* foi capaz de detectar o usuário presente na sala e controlar os dispositivos de acordo com as predições feitas pelo algorítimo ciente de contexto. A filtragem e avaliação das mensagens recebidas evitou que dados errôneos fossem incorporados nos bancos de dados e permite a redução de processamento nos microcontroladores.

O aplicativo implementado foi capaz de controlar e monitorar a sala. O uso de um framework de desenvolvimento para gerar aplicações híbridas permitiu que isso fosse possível de diferentes plataformas (Android e Web foram testadas).

Dados contextuais do ambiente se mostraram muito uteis para o controle e a otimização de consumo energético, validando o conceito proposto por esse trabalho. A arquitetura do algorítimo, utilizando técnicas de *data mining* para redução da dimensionalidade dos dados, acompanhado das capacidades de modelagem de não-linearidades das redes neurais, também se mostrou adequada para a aplicação em questão.

Palavras-chave:Ciência de Contexto, *Machine Learning*, Internet das Coisas, Ambientes Inteligentes, Inteligência de Ambientes, Computação Pervasiva, Automação Inteligente, Sistemas Embarcados, *Data Mining*, Computação Ubíqua.

Abstract

With the advent of the Internet of Things, devices can now receive and send data to the Internet. Many of these devices are used in homes, rooms and offices for better comfort, save user's time and energy. A complete smart space control system should automatically adjust its settings to the user preferences based on data previously collected from the smart things. In this project we use environmental data collected from sensors of the smart things to define contexts and preferences of the user for such. By feeding contextual data to a context-aware decision engine, composed by a combination of machine learning and data mining techniques and concepts. The context-aware decision engine is capable of identifying important data co-relations and use the environment's contextual information to make intelligent control decisions and adjust the environment's settings to the user's preferences and reduce the overall power consumption, yielding better services to the user and improving human-technology interaction.

Keywords: Context-Aware, Machine Learning, Gateway, Internet of Things, Smart-Environment, Ambient Intelligence, Pervasive Computing, Intelligent Automation, Ubiquitous Computing, Embedded Systems, Data Mining.

List of Figures

List of Tables

List of abbreviations and acronyms

AC:	Air Conditioning
ANN:	Artificial Neural Networks
API:	Application Program Interface
ARP:	Address Resolution Protocol
CPU:	Central Processing Unit
DHCP:	Dynamic Host Configuration Protocol
DNN:	Deep Neural Networks
EPOS:	Embedded Parallel Operating System
GPIO:	General Purpose Input / Output
GPU:	Graphical Processing Unit
HTTP:	Hyper-Text Transfer Protocol
IoT:	Internet of Things
IP:	Internet Protocol
JSON:	JavaScript Object Notation
LDR:	Light Dependent Resistor
LED:	Light Emmiting Diode
MAC:	Media Access Control
MCU:	Microcontroller Unit
MLP:	Multi-Layer Perceptron
PIR:	Passive Infra Red
PKD:	Prior Knowledge Data
RD:	Room Data
REST:	Representational State Transfer

SCADA:	Supervisory Control and Data Acquisition
TCP:	Transfer Control Protocol
UCD:	User Command Data
URD:	User Room Data

Contents

1 Introduction

In the upcoming world of the Internet of Things (IoT), devices contain numerous new capabilities, including the ability to receive and send data to the Internet. According to CISCO [1] and Business Insider [2] intelligence report on IoT, by 2020 there will be approximately 50 billion devices connected to the Internet. Of these amount, the report claims that the smart spaces market will make up roughly 27% of the whole IoT Market. These smart spaces consist of a series of electronic devices that are used to perform functions ranging from security to comfort and entertainment.

Among many reasons for the increasing popularity of the IoT, the most prominent are Convenience and Cost Reduction. Convenience is strongly related to time-saving. In today's world, where everything is moving so fast, every second has its value. The proof for that is that most of the technologies we use today are based on convenience. For example, cars get us where we need to go quicker and computers get work done faster. Smaller conveniences in the home will be desirable because they allow the home to save the user's time as well.

As civilization grows, it constantly needs more energy to power itself, and so does the cost of energy. This leads to heavy pressure on efficient use of energy. Smart systems help the user do this and save them money at the same time. The smart home system is able to monitor certain processes that use energy in the house and can control the amount of energy being used. A primary example would be with lights, as they are often left on when they don't need to be. A smart home system can be configured to turn the lights off after a certain amount of time, or when there is no user present in the environment. The smart home system may also allow the user to control the lights from his phone so they can be turned off.

Even though much has been developed in the field of the IoT, of the current devices on the market only a few make use of the data surrounding them. Moreover, most of them underperform due to the lack of adequate autonomy and learning ability, ending up just being used to remotely control some appliances. Meanwhile, the exponential growth of the number of controllable devices the user must manually adjust to his preferences every time he enters the room may make his life yet harder.

A complete smart space control system should automatically adjust its settings to the user preferences based on data previously collected from the smart things. All of this data (indoor and outdoor temperature and lighting, time, season, particular and overall energy consumption, air humidity, CO2, etc.) can be used to define contexts and the preferences of the user for such. Improving technology access to context can enrich human-technology interaction which yields services more useful to the user.

With this scenario in mind, this document describes the development of a system that takes the current IoT requirements into account. Sensors in the room gather data from the environment and feed it to the system. Machine learning and data mining algorithms are able to extract contextual meta-data and continuously learn the user's behavior and interaction with the environment. The information from these algorithms is used to automate control decisions for the devices and assist in energy efficiency.

The remainder of this document is organized as follows: Chapter 2 provides a brief overview of the main terms and technologies used and referred to along the work. Chapter 3 provides an overview of the current state-of-art in control of smart environments. The main objectives and requirements of this work are detailed in Chapter 4. Chapter 5 covers the system implementation. The results of the implementation are discussed in Chapter 6. Finally, Chapter 7 provides the final conclusions and project outlook.

2 Background

This chapter describes the main concepts, technologies, and tools that were used in this work, as well as in the research and development field. The theoretical background was the first stage of the project development, which allowed the understanding of Internet of Things, Embedded Systems and Machine Learning techniques leading concepts. Besides, this study made possible the familiarization with the primary tools and technologies currently employed in this kind of systems. The following sections provide a brief overview of these topics and pointers to the bibliographic sources.

2.1 Internet of Things

The Internet of Things (IoT) is the concept of connecting any device to the internet. These devices may range from cell phones, coffee makers, washing machines, headphones, lamps, wearable devices and almost anything else one may imagine. This vast range also applies to components of machines, for example, a jet engine of an airplane or the drill of an oil rig. The IoT is a giant network of connected “things” (which also includes people). The relationship may be between people-people, people-things, and things-things [3].

2.2 Smart Environments

The concept of smart environments evolves from the definition of ubiquitous computing that promotes the ideas of "a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network."

Cook and Das [4] define smart environment as "a small world where different kinds of smart device are continuously working to make inhabitants’ lives more comfortable." Smart environments aim to satisfy the experience of individuals from every environment, by replacing the hazardous work, physical labor, and repetitive tasks with automated agents.

Smart environments are broadly classified to have the following features [5]:

- Remote control of devices, like power line communication systems to control devices.

- Device Communication, using middleware, and Wireless communication to form a picture of connected environments.
- Information Acquisition/Dissemination from sensor networks.
- Enhanced Services by Intelligent Devices.
- Predictive and Decision-Making capabilities.

2.3 Context-aware computing

Context-aware computing is a style of computing in which situational and environmental information about people, places, and things is used to anticipate immediate needs and proactively offer enriched, situation-aware and usable content, functions, and experiences. Such context-aware systems adapt according to the location of use, the collection of nearby people, hosts, and available devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment [6].

2.4 Data Mining

Data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. Data mining software is one of some analytical tools for analyzing data. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases [7].

2.4.1 Random forests

One of the crucial functions of data mining is to analyze the relevance of a given variable to another. This way it is possible to reduce the size of the analytical space and improve the accuracy of the used algorithms.

Random forests or random decision trees are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way. The measure the variable importance in a data set is made during the fitting process. The out-of-bag error for each data point is recorded and averaged over the forest. To measure the importance of a feature after training, the values of the feature are permuted among the training data and the out-of-bag error is again computed on this perturbed data set. The importance score for the feature is computed by averaging the difference in out-of-bag error before and after the permutation over all trees. The score is normalized by the standard deviation of these differences. [8]

2.4.2 Extra Trees Classifier

Adding one further step of randomization yields extremely randomized trees, or ExtraTrees. These are trained using bagging and the random subspace method, like in an ordinary random forest, but additionally the top-down splitting in the tree learner is randomized. Instead of computing the locally optimal feature/split combination (based on, e.g., information gain or the Gini impurity), for each feature under consideration, a random value is selected for the split. This value is selected from the feature's empirical range (in the tree's training set, i.e., the bootstrap sample) [9].

The Extra Trees Classifier[1] implements a meta-estimator that fits some randomized decision trees (a.k.a. extra-trees) on various subsamples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. Additionally, this classifier can rank the features of datasets accordingly to a selected feature.

2.5 Machine Learning

Machine learning is a field of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data.

The process of machine learning is similar to that of data mining. Both systems search through data to look for patterns. However, instead of extracting data for human comprehension, as is the case in data mining applications, machine learning uses that data to detect patterns in data and adjust program actions accordingly. Machine learning algorithms are often categorized as being supervised or unsupervised. Supervised algorithms can apply what has been learned in the past to new data. Unsupervised algorithms can draw inferences from datasets [10].

[1] ExtraTreesClassifier, <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html>, Accessed: January/2017

2.5.1 Artificial Neural Networks

An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is inspired by the structure and functional aspects of biological neural networks. Computations are structured concerning an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables [11].

2.5.2 Fuzzy Logic and Fuzzy Control

Fuzzy logic is an approach to computing based on "degrees of truth" rather than the usual "true or false" (1 or 0) Boolean logic on which the modern computer is based. Fuzzy logic includes 0 and 1 as extreme cases of truth (or "the state of matters" or "fact") but also includes the various states of truth in between so that, for example, the result of a comparison between two things could be not "cold" or "hot" but "38% of coldness."

Fuzzy controllers are very simple conceptually. They consist of an input stage, a processing stage, and an output stage. The input stage maps sensor or other inputs, such as switches, thumbwheels, and so on, to the appropriate membership functions and truth values. The processing step invokes each applicable rule and generates a result for each, then combines the results of the rules. Finally, the output stage converts the combined result back into a particular control output value [12].

2.6 Microcontrollers

A micro-controller (or MCU, short for micro-controller unit) is a small computer (SoC) on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. This project comprises two different MCUs with different processing capabilitie, which are described below.

2.6.1 Intel Galileo Gen. 2

The Intel Galileo Gen 2 board[2], shown in figure 1, is based on the Intel Quark SoC X1000, a 32-bit Intel Pentium-class system on a chip (SoC). It is the first board based on Intel architecture designed to be hardware and software pin-compatible with shields designed for the Arduino Uno R3. The Galileo board is also software-compatible with the Arduino Software Development Environment.

[2] Intel Galileo Gen 2, <https://www.sparkfun.com/products/13096>, Accessed: January/2017

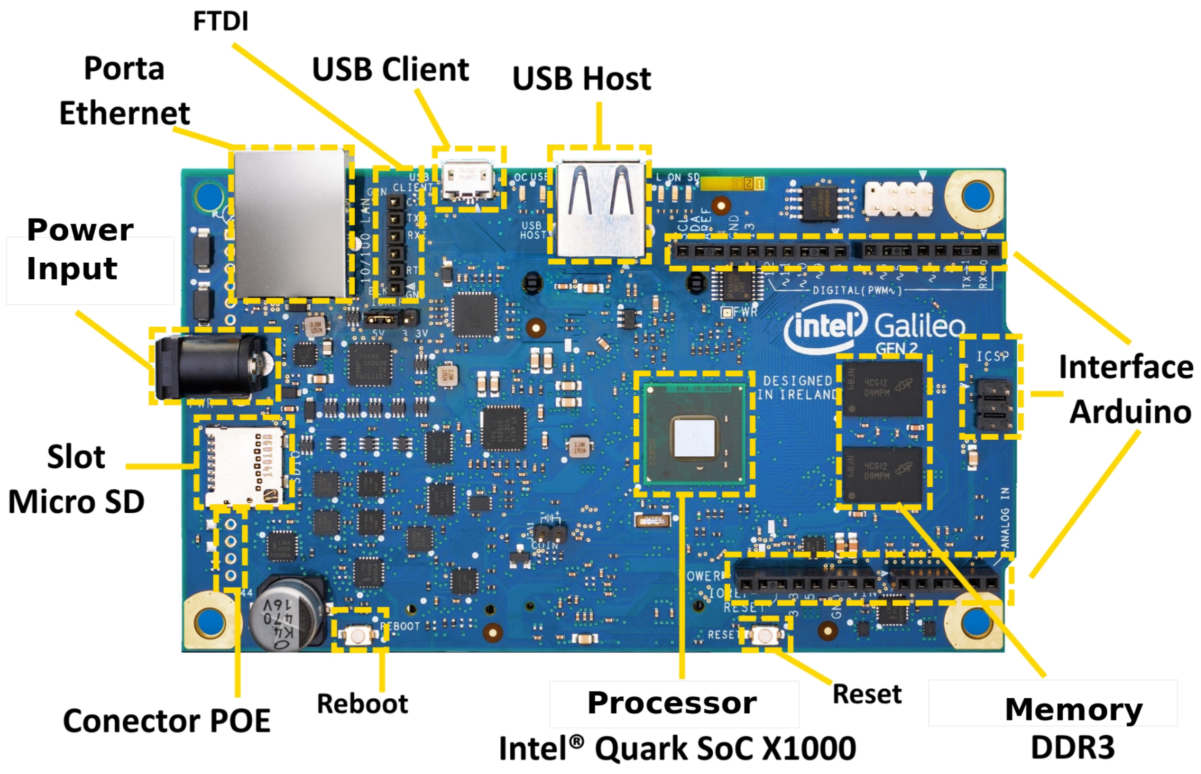


Figure 1 – Intel Galileo Gen. 2 Board

Source: Adapted from Sparkfun Electronics

In addition to Arduino hardware and software compatibility, the Galileo board has several PC industry standard I/O ports and features to expand native usage and capabilities beyond the Arduino shield ecosystem. A full sized mini-PCI Express slot, 100Mb Ethernet port, Micro-SD slot, USB TTL UART header, USB Host port, USB Client Port, and 8 MByte NOR flash come standard on the board. The genuine Intel processor and surrounding native I/O capabilities of the SoC provides for a fully featured offering for embedded system development.

2.6.2 EPOSMote III

EPOSMote III [13] is a micro-controller developed at LISHA. This piece of hardware consists of a Texas Instruments CC2538 SoC, that combines a powerful ARM Cortex-M3-based MCU system with up to 32KB on-chip RAM and up to 512KB on-chip flash with a robust IEEE 802.15.4 radio, all in a 8x8mm^2 QFN56 footprint.

2.7 Operating Systems

The choice of an operating system is directly tied to the hardware available, application and the tasks that the system should perform. In the subsections below, are

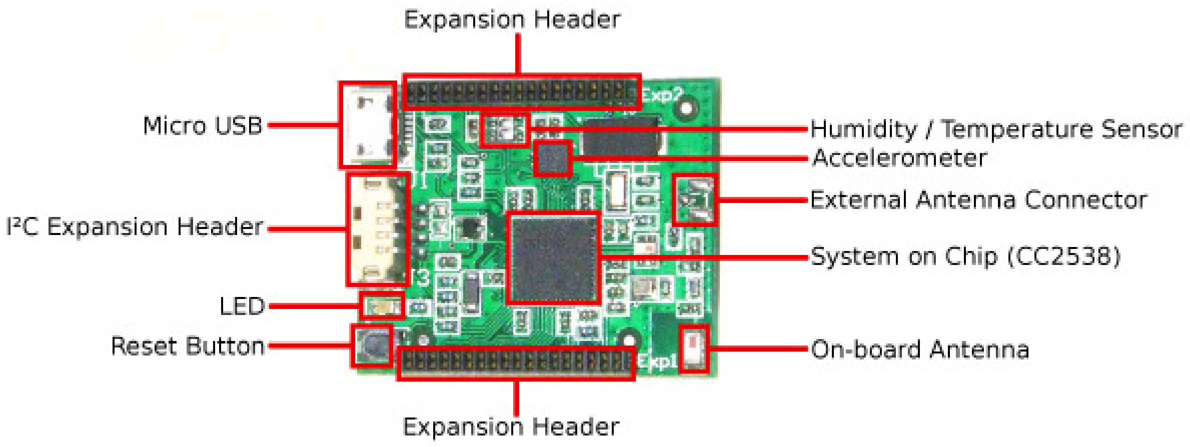


Figure 2 – EPOSMoteIII Board.

described the operating systems chosen for this project and the reason for the choice.

2.7.1 Ubilinux

The application running in the Intel Galileo Gen. 2 board complies many different functionalities: controlling GPIO pins, parsing sensor data, scanning the network for a specified MAC address and interfacing applications through Ethernet. This requires many different packages and features from the operating system. This vast range of features is only provided by a general purpose OS, such Linux. Linux comes in many different distributions that aim for different uses.

Instead of using the Linux image available on many of the Galileo Gen. 2 tutorials, for this project an alternative embedded Linux distribution based on Debian Jessie was chosen, called Ubilinux[3]. This distribution is targeted at embedded devices that have limited memory and storage capabilities, such as the Intel Galileo Gen. 2. This Linux distribution uses the same package manager as many other Debian-based distributions, the apt-get, and therefore, has access to a wide range of packages.

Also, there is a python module, called wiringx86[4], developed by Emutex that provides a straightforward and unified API to talk to the GPIO pins on Intel Arduino capable boards, such as Galileo Gen. 2. This made possible the elimination of the Arduino programming interface from our application, bringing everything together in a single development environment.

[3] Ubilinux, <https://emutex.com/products/ubilinux>, Accessed January/2017
[4] Wiringx86, <https://github.com/emutex/wiring-x86>, Accessed January/2017

2.7.2 EPOS

The operating system used in EPOSMote III is the EPOS[5]. The Embedded Parallel Operating System aims at automating the development of embedded systems so that developers can concentrate on what really matters: their applications. EPOS relies on the Application-Driven Embedded System Design (ADESD) [14] method to guide the development of both software and hardware components that can be automatically adapted to fulfill the requirements of particular applications. EPOS features a set of tools to support developers in selecting, configuring, and plugging components into its application-specific framework. The combination of methodology, components, frameworks and tools, enable the automatic generation of application-specific embedded system instances. Figure 3 shows a diagram of the EPOS and EPOSMote interaction.

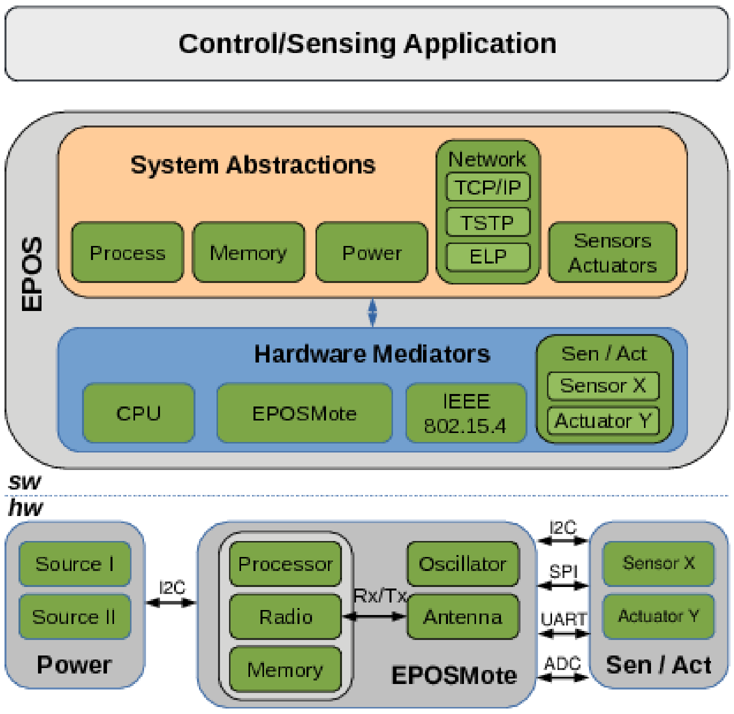


Figure 3 – EPOS@EPOSMoteIII.

2.8 Programming Languages

The proposed system complies many features, functionalities and still should attend to certain QoS requirements such as performance, resource usage, reliability, etc. Every programming language has its advantages and drawbacks. The following subsections describe the programming languages used in this project.

[5] EPOS, <http://epos.lisha.ufsc.br>, Accessed January/2017

2.8.1 BASH

Bash[6] is the GNU Project's shell. Bash, the Bourne Again SHell, is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh). It is intended to conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard. It offers functional improvements over sh for both programming and interactive use. In addition, most sh scripts can be run by Bash without modification. It's reasonably robust for automating and integrating disparate executables, and more importantly it's installed by default on nearly every flavor of Unix.

2.8.2 C++

C++[7] is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. This language is used for programing the EPOSMote III firmwares.

2.8.3 Python

Python 3.5[8] was chosen as the main programming language for both the gateway and the context-aware decision engine. Python is a widely used, high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. For this reason was the choice for implementation of the gateway daemons and machine learning algorithms.

2.8.4 HTML and CSS

HTML is a computer language devised to allow website creation. These websites can then be viewed by anyone else connected to the Internet. It is relatively easy to learn, with the basics being accessible to most people in one sitting; and quite powerful

[6] BASH, <https://www.gnu.org/software/bash/>, Accessed January/2017
[7] C++, <https://isocpp.org/>, Accessed January/2017
[8] Python 3.5, <https://www.python.org/downloads/release/python-350/>, Accessed January/2017

in what it allows you to create. It is constantly undergoing revision and evolution to meet the demands and requirements of the growing Internet audience under the direction of the W3C, the organisation charged with designing and maintaining the language.

2.8.5 JavaScript and JQuery

JavaScript [15] is a high-level, dynamic, untyped, and interpreted programming language. It has been standardized in the ECMAScript language specification. Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content production; the majority of websites employ it, and all modern Web browsers support it without the need for plug-ins.[6] JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting object-oriented,[8] imperative, and functional programming styles.[6] It has an API for working with text, arrays, dates and regular expressions, but does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

jQuery [16] is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript. Query Mobile is a HTML5-based user interface system designed to make responsive web sites and apps that are accessible on all smartphone, tablet and desktop devices.

2.9 Libraries and Modules

Along with the programming languages several libraries and modules were used to provide them with the additional required functionalities. The most important in the scope of this project are listed and described in the following subsections.

2.9.1 Pandas

Pandas[9] is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

[9] Pandas, <http://pandas.pydata.org/>, Accessed January/2017

2.9.2 Flask

Flask[10] is a small and powerful Python framework for the development of web applications. This framework is extensively documented and provides RESTful request dispatching, which is used to communicate the server, the application and the gateway.

2.9.3 Keras

Keras[11] is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation.

2.9.4 Theano

Theano[12] is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It can use GPUs and perform efficient symbolic differentiation.

2.9.5 SkFuzzy

Scikit-Fuzzy[13] is a collection of fuzzy logic algorithms intended for use in the SciPy Stack, written in the Python computing language. This module allows the creation of fuzzy control algorithms, visualization of the fuzzy sets and simple integration.

2.9.6 Scikit-Learn

Scikit-learn[14] is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to inter-operate with the Python numerical and scientific libraries NumPy and SciPy.

2.9.7 Network Mapper

The tool used to sweep the network searching for MAC addresses is called Nmap[15]. Nmap ("Network Mapper") is a free and open source utility for network discovery and security auditing. Nmap uses raw IP packets in novel ways to determine what hosts

[10] Flask Web Application Framework, <http://flask.pocoo.org/>, Accessed January/2017
[11] Keras, <https://keras.io>, Accessed January/2017
[12] Theano, <http://deeplearning.net/software/theano/>, Accessed January/2017
[13] SkFuzzy, <http://pythonhosted.org/scikit-fuzzy/>, Accessed January/2017
[14] Scikit-Learn, <http://scikit-learn.org>, Accessed January/2017
[15] Network Mapper (Nmap) <https://nmap.org/>, Accessed January/2017

are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. It was designed to rapidly scan large networks, but works fine against single hosts.

2.9.8 Apache Cordova

Apache Cordova[16] (formerly PhoneGap) is a popular mobile application development framework. It is an open source software that enables software programmers to build applications for mobile devices using CSS3, HTML5, and JavaScript instead of relying on platform-specific APIs like those in Android, iOS, or Windows Phone. It enables wrapping up CSS, HTML, and JavaScript code depending upon the device platform. It extends the features of HTML and JavaScript to work with the device. The resulting applications are hybrid, meaning that they are neither truly native mobile application nor purely web-based.

[16] Apache Cordova, <https://cordova.apache.org/>, Accessed January/2017

3 Related Work

The lack of knowledge regarding the state-of-the-art of an individual research field may lead to many misconceptions during the development phase, and therefore is of great importance. This chapter contains a brief presentation of the related works. The works presented in this chapter relate to the one presented in this document from basic approaches, evaluations or possible further applications. They also provide enough information for the comparison of techniques and strategies.

In the paper *"Middleware for the Internet of Things: A Survey"* [17], the authors outline a set of requirements for IoT middleware and present a comprehensive review of the existing middleware solutions. The proposals of existing middlewares for the IoT are diverse and involve various design approaches and support different requirements. The author's paper puts these works into perspective and presents a holistic view of the field. In doing this, the key characteristics of IoT infrastructure (heterogeneous, resource-constrained, ultra-large-scale network and number of events, context-aware, etc.) and applications (Real-time, Diverse, Secure, etc.), as well as the requirements of IoT's middleware are identified. Based on these requirements, a comprehensive survey of these middleware systems focusing on current, state-of-the-art research has been presented. Most of these proposals have been reviewed and summarized in terms their supported functional, nonfunctional, and architectural requirements. Finally, open research issues, challenges and recommended possible future research directions are outlined. Although the existing middleware solutions address many requirements associated with middleware in IoT, some requirements and related research issues remain relatively unexplored, such as scalable and dynamic resource discovery and composition, system-wide scalability, reliability, security and privacy, interoperability, integration of intelligence, and context-awareness. Design decisions and implementations in the project described by this document followed the guidelines from this survey and took specially interoperability, integration of intelligence, context-awareness requirements and challenges into account.

Lee and Chung propose a system architecture for context-aware home applications covering all aspects of ubiquitous computing and network including applications, middleware, gateway, OS, sensor network, etc. [18]. They approach this problem by generating and defining applications that are proper to show ubiquitous computing applications at home. Secondly, listing the technologies needed to develop such scenarios. The authors focused on four main scenarios: Media-Services, Healthcare Services, Control Services and lastly Management Services. Figure 4 illustrates the structure of the proposed system. The lowest level of the system is components that directly

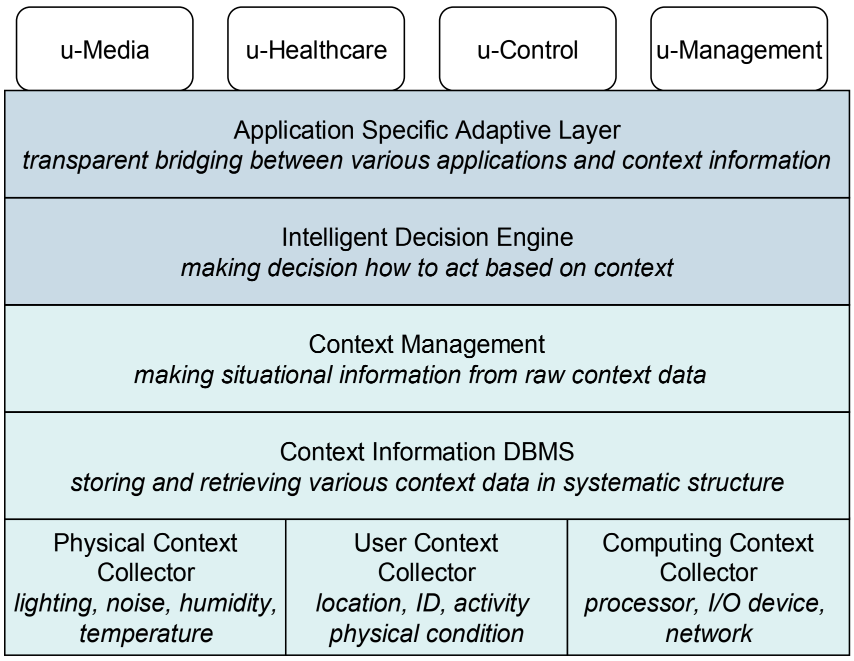

u-Media
u-Healthcare
u-Control
u-Management
Application Specific Adaptive Layer
transparent bridging between various applications and context information
Intelligent Decision Engine
making decision how to act based on context
Context Management
making situational information from raw context data
Context Information DBMS
storing and retrieving various context data in systematic structure
Physical Context Collector
lighting, noise, humidity, temperature
User Context Collector
location, ID, activity physical condition
Computing Context Collector
processor, I/O device, network

Figure 4 – Structure of the Context-Aware Home Application Server

Source: [18]

sense context. Context information DBMS manages context information using ordinary DBMS and offers upper layer the access to it. Context Management constructs high-level situational information. The Intelligent Decision Engine operates like an intelligent agent. It should be able to present the information and services to a user or execute a service. Lastly, The Application Specific Adaptive Layer should provide a system with the capability to bridging various application and context information. Despite the complete description of the system architecture, the system architecture was not validated, because the system was not implemented.

An operational definition of context and context-aware systems is provided by Dey in the paper *"Understanding and Using Context"* [19]. The author characterizes context as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. Further, the author defines a context-aware system as any system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. He also proposes which are the main features that should be implemented in a context-aware system: *i)* presentation of information and services to a user; *ii)* automatic execution of a service for a user; *iii)* tagging of context to information to support later retrieval. In the paper a new abstraction is introduced, the *situation*. Situations, however, rely on human classification and require too much user input. The system proposed in this work, on the other hand, proposes a composition of data mining and machine learning techniques to make user classification unnecessary.

The study presented by [20] aims at developing an indoor temperature control method that could provide comfortable thermal conditions by integrating heating system control and the opening conditions of building envelopes. Artificial neural network (ANN)-based temperature control logic was developed for the control of heating systems and opportunities at the building envelopes in a predictive and adaptive manner. Analysis results revealed that the ANN-based control logic improved the indoor temperature environment with an increased comfortable temperature period and decreased overshoot and undershoot of temperatures outside of the operating range. The ANN-based temperature control logic was able to maintain the indoor temperature more comfortably and with more stability within the operating range due to the predictive and adaptive features of ANN models. The proposed logic did not show significant superiority in energy efficiency over the conventional logic.

The authors Zheng, Wang and Black approach context-awareness in a smart home with the detection and recognition of human activity [21]. Their smart home had a set of state-change sensors installed and was tested during two weeks. Their system is based on a self-adaptive neural network called Growing Self-Organizing Maps that exhibits several appealing features in the analysis of patterns encoded in daily activity. Even though their method has proven to be very suitable for detecting and recognizing human activity, it does not address some crucial issues such as automatic control based on context and composition of machine learning techniques to improve the results of the system.

In the work of Wanglei and Shao [22], the authors propose the implementation of an Adaptive Neuro-Fuzzy Inference system (ANFIS) to optimize current home environment and make responses based on previous habitual behaviors. Their implementation combines fuzzy control and neural networks. They also propose the use of K-means clustering algorithm to deal with the main problems related to the use of such algorithms, which are caused by the high number of variables, the exponential growth of fuzzy rules and back-propagation algorithm slow learning. The implementation reported consists of 5 layers:

1. Fuzziness of the variable. The output is membership degree of the corresponding category.

2. Right value of the rules. The output is the multiplication of the membership degrees.

3. Normalization Output is the standardization of the value.

4. Defuzziness of the variables. The output is the weighted value for each rule.

5. Output of ANFIS.

To evaluate their implementation, the authors used Sunlight and Lamplight measured through sensors, the open ratio of the curtain by user experience. Data was collected from 6:00 to 17:00 every 3 minutes. 220 sets collected (150 training sets and 70 testing sets). Even though the authors claim their approach can overcome the exponential growth of fuzzy rules due to the variables number and learning speed of usual back-propagation algorithms, they did not evaluate any of these points.

Zhang *et al.* describe in their paper [23] a smart home control system based on information fusion. Their proposed framework includes internet access and data acquisition modules, in-house networking services with wireless connectivity, embedded computational units in home appliances and an information fusion controller based on fuzzy logic and neural networks. Fuzzy logic is used to combine heterogeneous information from different sources and to execute an inference mechanism in terms of IF-THEN rules. Neural Networks were used to provide the system with learning ability. The case study for this scheme consisted of the control of an alarm clock able to generate a suitable alarm lead time. The system acquires weather and traffic conditions and uses the fuzzy controller and neural networks to determine a suitable alarm time. The proposed system has proven to be successful in generating a suitable alarm lead time. Differently then what is proposed in this work, the authors use fuzzy controller to generate the inputs for the neural network. In this work fuzzy controllers are used as optimizers able to enhance energy efficiency. Additionally, the authors assume only a few variables that they consider relevant as input for their system, they do not consider a real IoT environment where there are numerous sources of data and the user does not know enough to select the most appropriate variables to use in the system.

Ye *et al.* propose an adaptive smart home control system composed of two learning algorithms: a toddler algorithm based on ant colony optimization to learn user operation rules and an unnamed correlation algorithm to identify rule relations [24]. The algorithms run in a central controller uses the feedback information from the household appliances to find out the user's habits. The nodes of this system proposed in their work use PLC (Power Line Carrier) modules to communicate with each other. The authors claim in the paper that the system has reached the design requirements, however, they identify two unsolved issues in their system. It needs a long time to learn the rules of user behavior and after it has learn a rule, it is very difficult to modify that rule.

It is possible to conclude that there are still lots of points that most of the current existing approaches do not deal with. This work, takes into account many of the issues encountered in the related researches and proposes a novel composition of technologies to implement a smart environment.

4 Project Goals and Requirements

This chapter describes the main goals and requirements of this project. The fundamental components of the system are described and for each one the primary requirements are specified.

4.1 Development of an Intelligent Gateway and Integration with Smart Room

As IoT grows and billions of devices need to connect to the world, one of the most critical components of future Internet of Things systems may be a device known as an "IoT gateway." IoT gateways perform several key functions such as device connectivity, protocol translation, data filtering and processing, security, updating, management and more. Newer IoT gateways also operate as platforms for application code that processes data and becomes an intelligent part of a device-enabled system. The system described in this document considers a gateway that isolates smart things from the TCP/IP world, integrating different technologies and providing an environment for secure operation [17]

4.1.1 Intelligent Gateway Main Requirements

- **User Detection:** The gateway should be able to scan the local network and detect users by the MAC address of their smartphone.
- **Bridge different communication protocols:** The devices inside LISHA's Smart Room communicate through Modbus messages over IEEE 802.15.4. The user and server on the other hand, communicate with the gateway via TCP/IP over Ethernet. Therefore the gateway must be able to successfully bridge these two standards.
- **Parse and Filter Sensor Data:** The smart devices inside the room send raw data in Modbus messages to the gateway. The gateway must be able to parse the Modbus messages as well as convert sensor data to units of the International System and filter outliers of erroneous data.
- **Communicate with the Internet:** User commands and server queries are received and issued by the gateway through HTTP REST requests.
- **Periodically Log Environment Data to the Server:** Environment data from sensors are used for the training and preference prediction of the context-aware

control algorithm. For this reason data should be logged periodically accordingly to an adequate sampling period.

- **Query Context-Aware algorithm for environment Adjustments:** The gateway must be able to issue prediction queries to the server running the context-aware algorithm for the connected user.

- **Ensure QoS:** All the described functions must run in parallel, so the user comfort is not impaired.

4.2 Development of a hybrid application

Portability is a crucial issue in IoT projects. Since most things are capable of being remotely controlled, a user must be able to have access to what he wants in the closest platform. If a user is surfing the Internet on his computer, it is easier to open another tab in the browser than to find his smartphone and initiate an application to control a device or check the current consumption of an appliance. Therefore, cross-platform implementations of the system are crucial. The implementation of such, however, is not trivial. Android applications are implemented in Java while web apps can be implemented in a series of other languages such as HTML, JavaScript, CSS, Java e etc.

4.2.1 Hybrid Application Main Requirements

- **Cross-Platform Availability:** It should be possible to run the application in different platforms and devices.

- **Remote device control capabilities:** A user should be able to remotely turn devices on and off, independently of where he is.

- **Real-Time data visualization:** A user must be capable of checking the current device power consumption, temperature and every other environment information available.

- **Data History Visualization:** The application should be able to query user history and display the data in form of a time-series chart.

4.3 Development of a Context-Aware Environment Control Algorithm

Most of Supervisory Control And Data Acquisition (SCADA) systems are still able to cope with the current IoT and industrial technologies. These systems work with pre-defined rules and devices, but the slightest change may cause the whole system to be re-adapted. The addition of a new device to the network, for example, may require a technician to integrate the technology, evaluate the impact of the device and reformulation of a series of rules. When we consider Cisco and Business Insider IoT market and device number expectations, it is clear to notice that these systems will not be capable of dealing with this. SCADA systems also fall short on many other features that are very desirable in the future world of the IoT, such as data aggregation, predictive analytics, etc.

Intelligent control in smart environments can be realized by analyzing the data in a sensor network and the user's previous behavior of operation to the available appliances and devices, without user intervention. This control system can predict and control the household appliances intelligently to make whole home environments more environmentally friendly and comfortable. To improve the learning ability of home control system, to take full advantage of the sensor network data we propose an Adaptive Context-Aware decision engine, that uses previous user meta-data to learn how to adapt the room to the user preferences automatically. By identifying key correlated variables, we can select the most relevant data for the control of an individual device.

4.3.1 Context-Aware Algorithm Main Requirements

- **Automatic Identification of Relevant Information:** Smart Environments are composed of a series of mixed data sources. The use of the entire dataset for training and forming the control rules of certain devices may lead to overfitting and unreliable actions. The context-aware control algorithm should be able to identify the features of data that are relevant for the control of an individual device to provide better statistical support to the decisions taken by the algorithm.

- **Inclusion of previous pre-defined rules:** To prevent the decision engine from being idle in the bootstrap phase (while there is not enough user behavior information for the algorithm to be successfully trained), the context-aware algorithm should be able to incorporate previous rules or user preferences.

- **Continuous User Preference Learning:** New user behaviors may show up with time. For this reason, the system should be able to keep learning from data.

- **User-Specific Preference Prediction:** Environment adjustment predictions should be made according to the user that is currently present in the given environment.

- **Heuristic-based Energy Footprint Optimization:** Since in general users are not concerned, or not capable of watching the whole time the energy consumption of the devices they use in day-to-day activities, the system should implement optimizers for the prediction of the context-aware algorithm that takes energy saving heuristics into account.

4.4 System Integration

A system is an aggregation of subsystems cooperating so that the system can deliver the overarching functionality. System integration involves integrating existing often disparate systems. The above-described modules and subsystems should be joined as a single system, ensuring that each integrated subsystem performs the required functions.

4.4.0.1 System Integration Requirements

- **Successful communication between disparate modules:** Modules must exchange data successfully. This includes the acknowledgement of messages and and checking whether data did not get corrupted.
- **Resume operation after shutdown without errors:** If a subsystem or the whole system comes to fail, it should be able to resume operation normally after detecting the fault.
- **Maintain functioning without significant overhead:** The subsystems share resources and must not affect each other's functionalities or performance in any significant way.
- **Ensure non-existence of dead-locks:** Should different threads come to hang in execution, they should not prevent one another from finishing.

4.5 Validation

Validation is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. The primary *metrics* for the evaluation of the system are: *i)* prediction accuracy of the user's preferences regarding a given context; *ii)* ability to reduce the energy consumption of controllable devices through power saving heuristics; and *iii)* ability to learn new user behaviors with time.

For a successful system outcome, all of them must be attended. These performance indicators must be met to indicate that the system has been built according to the requirements and design specifications as well as ensuring that the product meets the user's needs.

5 Project Development

This chapter details the development process of the project. Firstly, a general overview of the system is made, describing the main components and how they interact with each other. Secondly, the Smart Room is described. The following sections describe the other system's modules and functionalities detailing the implementation.

5.1 System Overview

The system proposed by this project is made of four main components. These components are shown in Figure 5. The user application, the cloud server that hosts both the database and context-aware decision engine, the Smart Room at the Software/Hardware Integration Lab, and lastly, the Internet of Things Gateway.

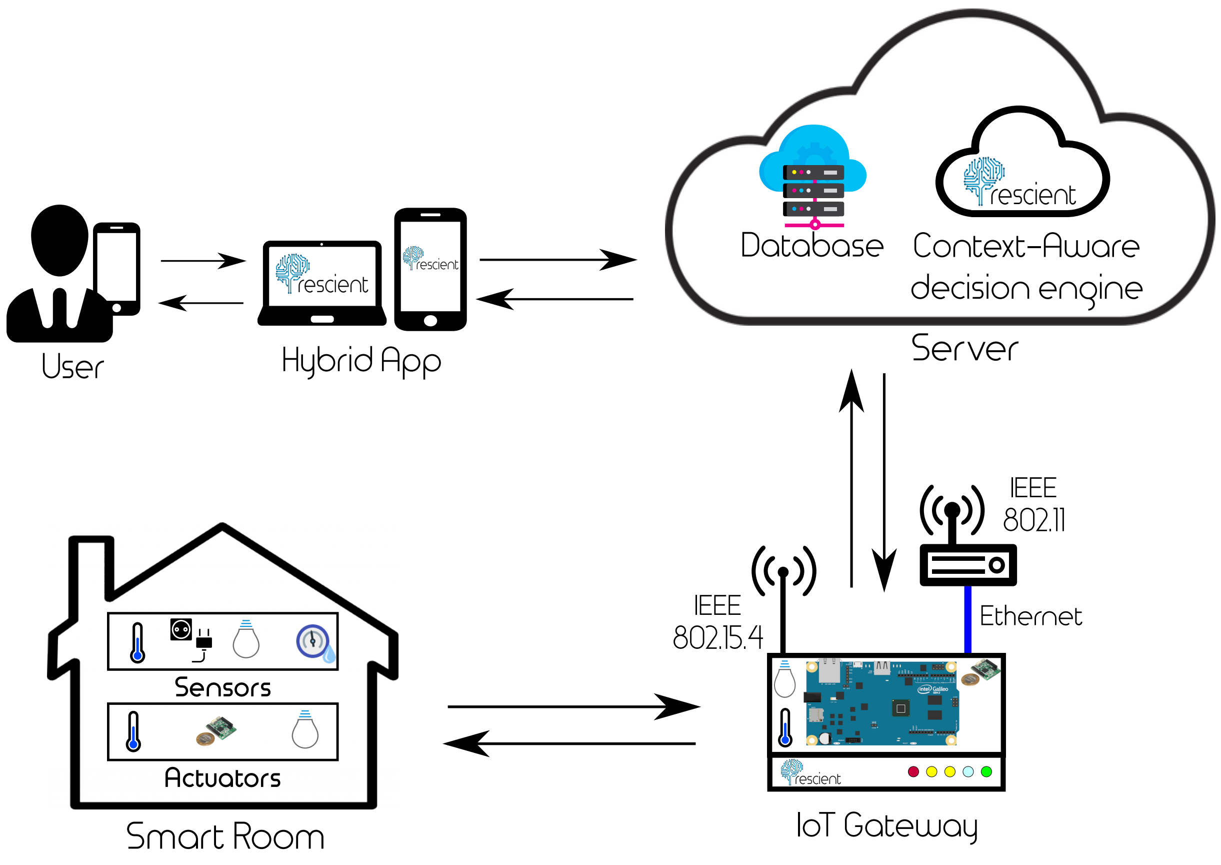


Figure 5 – System Overview

The hybrid application provides the user interface to the system. It is called hybrid application due to the framework used for the development, it allows the generation of cross-platform applications, in other words, the resulting application can run either on a web browser or a mobile device. It can show real-time data as well as data history from the smart room sensors. The user is also allowed to control the smart devices through the hybrid application. Figure 6 presents a use case diagram for the hybrid user application. This application interfaces with the HTTP server via RESTful requests.

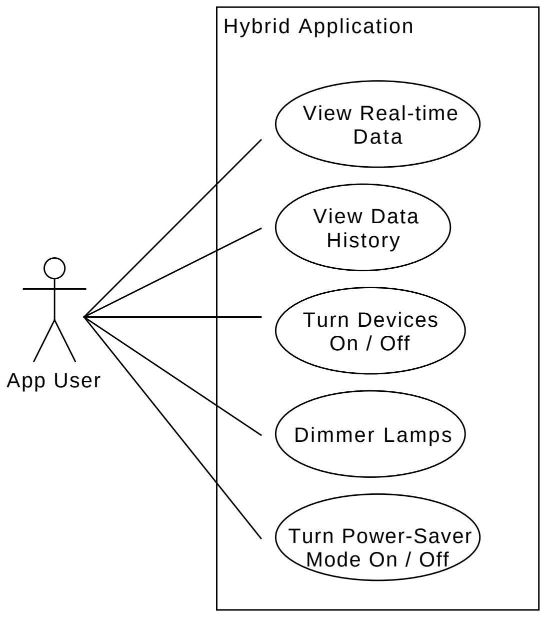


Figure 6 – Use case diagram for User Application.

The server is responsible for a series of functions, it receives data from the gateway and stores it in the database, relays commands from the application to the gateway and hosts the context-aware decision engine. The decision engine queries information from the database, selects the most relevant features for the control of a certain output and trains a neural network to learn how to control the room based on the user's previous behavior. The result from the neural network may or may not be optimized by a fuzzy controller. The server functionalities are illustrated by the use case diagram on Figure 7.

The smart room features a set of EPOSMote III microcontrollers with a series of sensors employed for periodically reporting the measured data via Modbus messages over IEEE 802.15.4 protocol. The sensors are used to collect data such as energy consumption of power outlets and lamps, temperature, air humidity, luminosity, etc. The motes are also capable of actuating on devices, on the lights, for example, the motes can turn them on or off and control the luminosity intensity. The use case diagram shown in Figure 8 present the basic functions of the EPOSMote III.

A gateway is used to bridge the IEEE 802.15.4 devices (EPOSMote III) with the Internet. This gateway consists of an Intel Galileo Gen. 2 Board that is connected via serial USB to an EPOSMote III. The EPOSMote III is used as a message relay, it receives the Modbus messages from other Motes inside the smart room and forwards

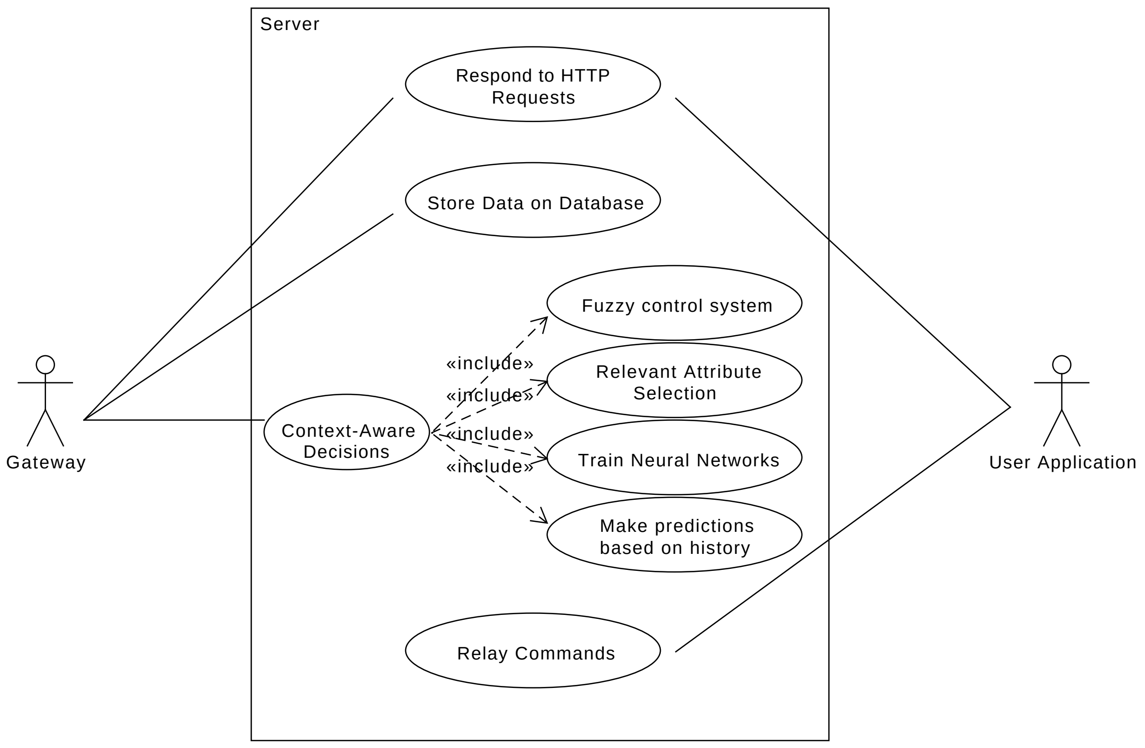


Figure 7 – Use case diagram for the server.

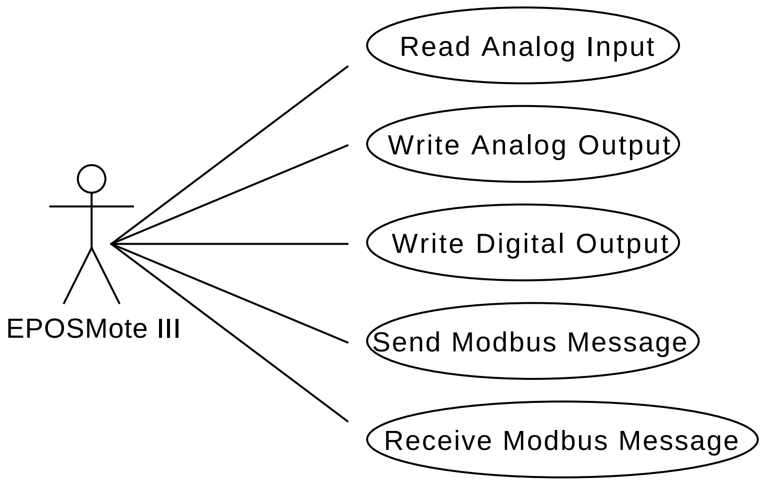


Figure 8 – Use case diagram for EPOSMote III.

these messages to the Galileo board. It is also able to send Modbus messages that come from the Galileo board to any mote in the room. In the other end, the Galileo board is connected to a wireless router via Ethernet cable, therefore is virtually located in the same subnetwork as any other device connected to the router. Additionally, the board is also connected to presence, temperature and luminosity sensors, and 5 LEDs that provide information about connectivity and operation for the user. Data collected in the gateway is converted into JSON and sent to an HTTP web server via RESTful requests and stored in the database. Figure 9 displays a use case diagram with the functions that should be performed by the gateway.

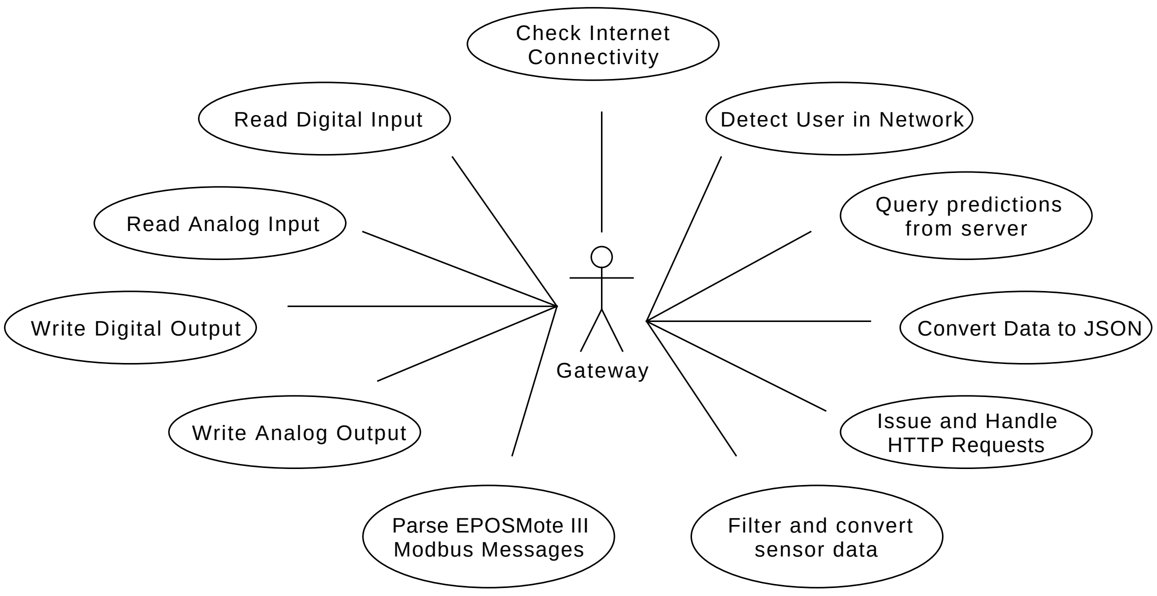

Check Internet Connectivity
Read Digital Input
Detect User in Network
Read Analog Input
Query predictions from server
Write Digital Output
Convert Data to JSON
Gateway
Write Analog Output
Issue and Handle HTTP Requests
Parse EPOSMote III Modbus Messages
Filter and convert sensor data

Figure 9 – Use case diagram for the Gateway.

5.2 LISHA's Smart Room

The Software/Hardware Integration Lab (LISHA) at Federal University of Santa Catarina (UFSC) already had a smart room. In former Internet of Things projects, some smart things were developed and deployed for testing inside one of the Lab's room. Later these devices were used to automate UFSC's Smart Solar Building [25]. The floor plan of the room with the position of the installed devices is illustrated by Figure 10.

In this project, the EPOSMoteIII with EPOS operating system were used as the base hardware for all mobile nodes. The node's firmware is written in the programming language C++. Communication between leaf and central nodes are made wireless via an IEEE 802.15.4 network with a star topology. The devices periodically report sensor measurements to the central node through the exchange of ASCII Modbus messages, without any previous data conversion (data is sent in raw format). Table 5.2 shows the Modbus message format, while Table 5.2 shows some of the message examples using Modbus for the devices in the room.

Data conversion to meta-data is done entirely in the gateway, which puts data into the correct format and posts it for database storage in the cloud.

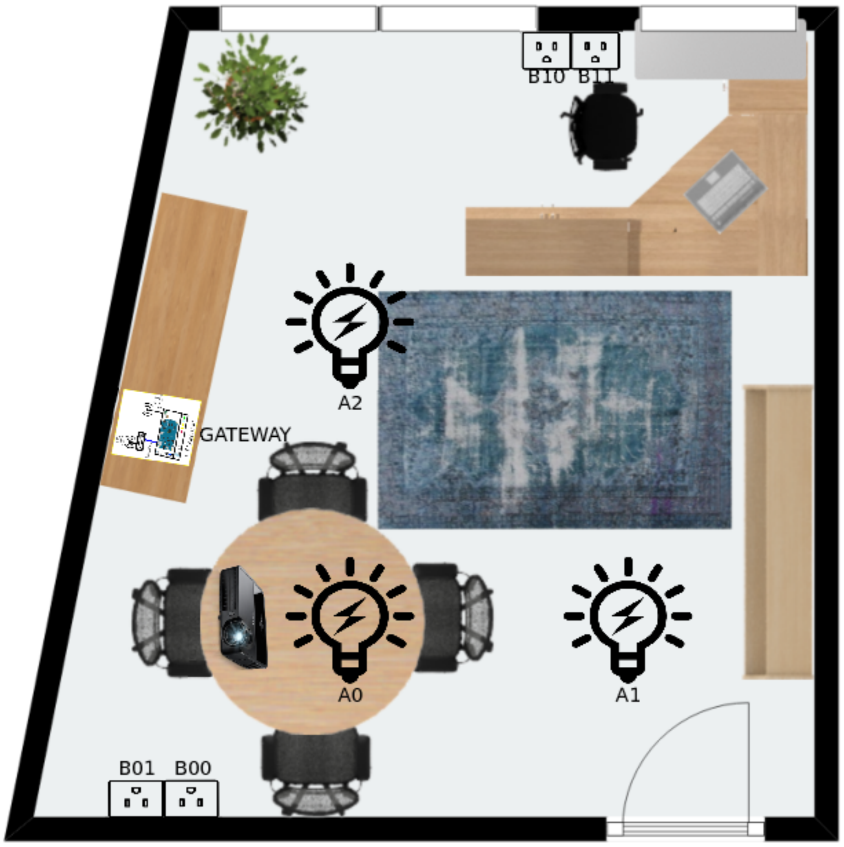


Figure 10 – Smart Room at LISHA UFSC

Field	**Length (char.)**	**Function**
Start	1	Starts with colon (':')
Address	2	Station Address
Function	2	Indicates the function codes (i.e. read/write coil/inputs)
Data	n	Data + length are function-dependent
CSM	2	Checksum for integrity checks
End	2	Carriage return and line feed (CR + LF)

Table 1 – Modbus Message Format.

Device ID	**Type**	**Functions**	**Registers**
0xA0	light A0	Write Single Coil (0x05) Write Single Register (0x06) Read Holding Register (0x03)	Turn light A0 On/Off Dimm light A0 Read Energy Consumption
0xAC	Air Conditioner	Write Single Coil (0x5) Write Single Register (0x06)	Turn AC On/Off Select temperature
0xB0	Power Outlet B0	Read Register (0x03)	Read Energy Consumption

Table 2 – Modbus Message Examples for the devices in the Smart Room

5.2.1 Power Outlets

Another device developed at LISHA is called the SmartPlug, shown in Figure 11. The smart plug is a device that allows the control and energy consumption measurement of appliances. Four smart plugs were already installed in the smart room, but were inactive. All the EPOSMote III inside the power outlets were reprogrammed to send every second a Modbus message reporting the current energy consumption.

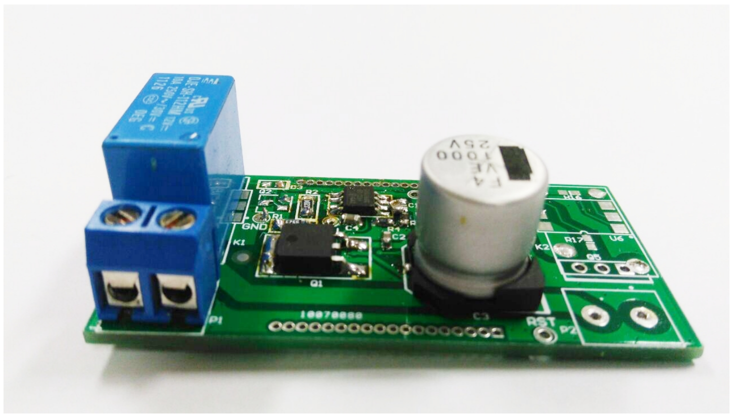


Figure 11 – Smart Plug.

The analog read values are sent raw to the gateway. The gateway is responsible for the Modbus message parsing and data conversion, as it holds a table for conversion of all measurements from the mobile nodes. Figure 12 shows a picture of the power outlets with EPOSMoteIII inside.

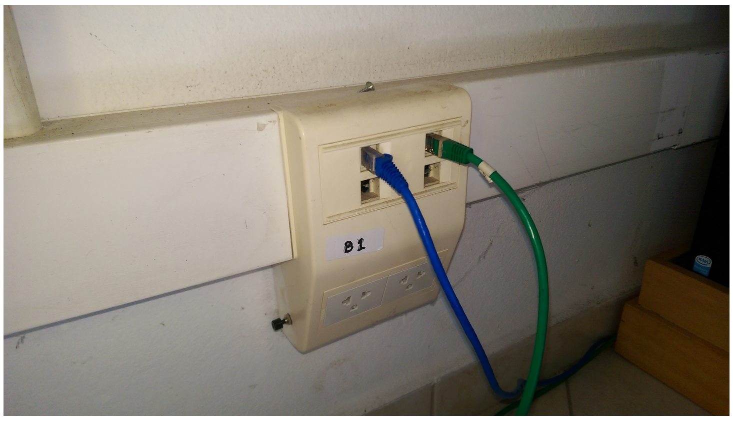


Figure 12 – Power Outlet with Smart Plug.

5.2.2 Lights

The lights inside the smart room were the only fully-functional devices already installed. They did not require any additional reprogramming or installation. Similar to the power outlets, the lights in the smart room also use a smart-plug for measuring energy consumption and turning them on/off. The firmware used for the lights is a little bit more complex, as it also handles the dimerization of the lights.

5.2.3 External Sensing Node

Since the intelligent system relies on contextual data to achieve better performance, it was speculated that the system could benefit from data outside the room such as luminosity, temperature, and humidity. For that reason, an extra sensing node was deployed outside the room. Two additional sensors were connected to the EPOSMoteIII. One measuring luminosity and other temperature and humidity. Each component is better explained in the following sections.

5.2.3.1 Luminosity

To measure luminosity a Light-Dependent Resistor (LDR) of 5mm was chosen. Both the LDR and the circuit schematic, which was used for the measurements connected to an analog input of EPOSMote III, are illustrated by Figure 13.

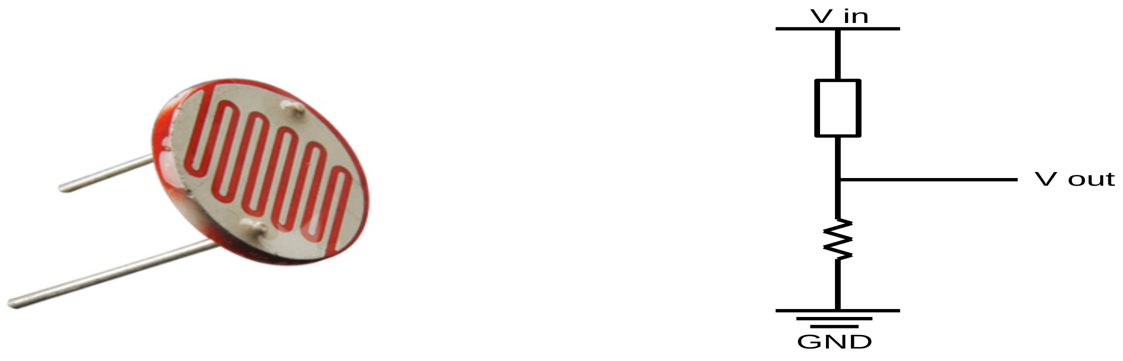


Figure 13 – Light-Dependent Resistor and schematic for connection to EPOSMoteIII.

Source: ProjectShop DB

The EPOSMote III has a 12-bit AD converter for the analog inputs. The value read from the analog input in the mote, to which the LDR is connected to is sent to the gateway and mapped to values ranging from 0% to 100%.

5.2.3.2 Temperature and Humidity

The measurements of external temperature and humidity were made using the DHT11 sensor. Different from the LDR, that is analog, the DHT11 is a digital transducer. The output pin of the DHT11 is connected to a digital GPIO pin of the EPOSMoteIII. The MCU must send a start signal to ask the transducer for the information. The transducer answers with a bit stream that differentiate 1s and 0s by the low and high signal periods.

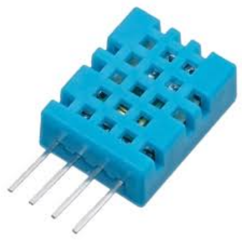


Figure 14 – DHT11 Temperature and Humidity sensor.

Source: Arduino Forum

When EPOSMote III sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for the MCU to complete the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include relative humidity and temperature information to the MCU. Without the start signal from MCU, DHT11 will not give back the response signal. Once data is collected, DHT11 will change to low power-consumption mode until it receives a start signal from MCU again. The overall communication process between the transducer and the micro-controller is shown on Figure 15.

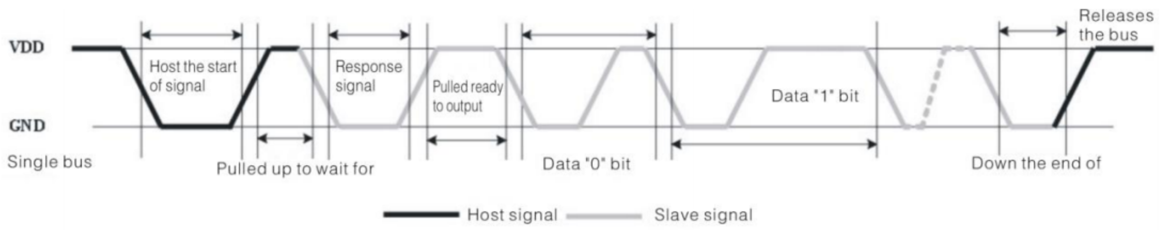


Figure 15 – Overall communication process between EPOSMoteIII and DHT11.

Source: Micropik DHT11 Datasheet

5.2.3.3 External Sensing Node Implementation

Two additional classes were developed to handle the LDR and DHT11 sensors. The microcontroller firmware was rewritten to incorporate these two new classes and report the data from the sensors via Modbus messages.

The above described components were brought together in a breadboard and connected to EPOSMote III. The resulting prototype and respective connection schematic is shown in Figure 16.

To protect the circuit and the microcontroller, both were placed in a case. In figure 17 the node is shown placed outside the window and connected to a power-bank as its power source.

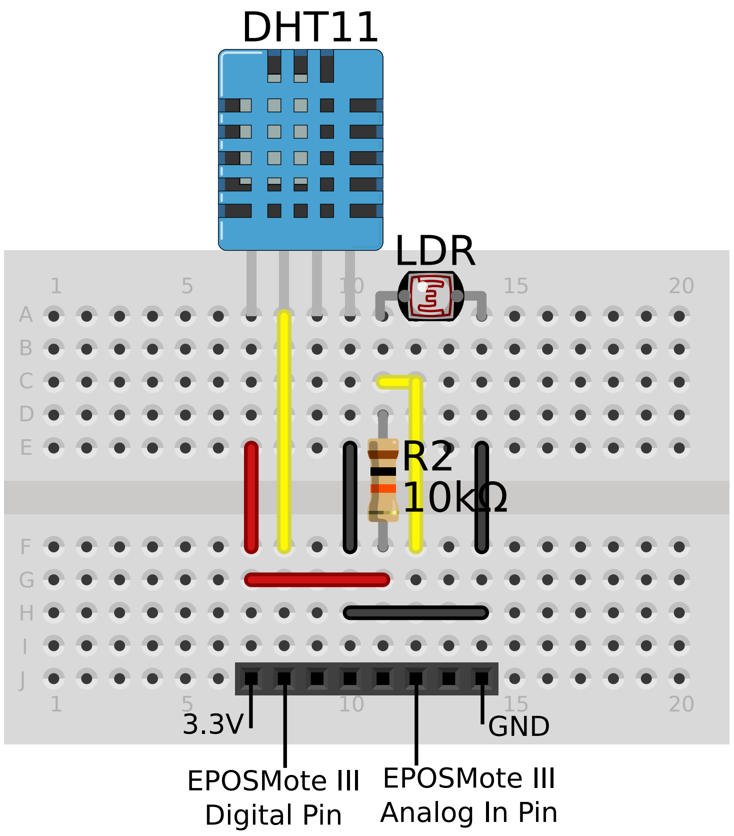


Figure 16 – EPOSMote III Circuit prototype Schematic.

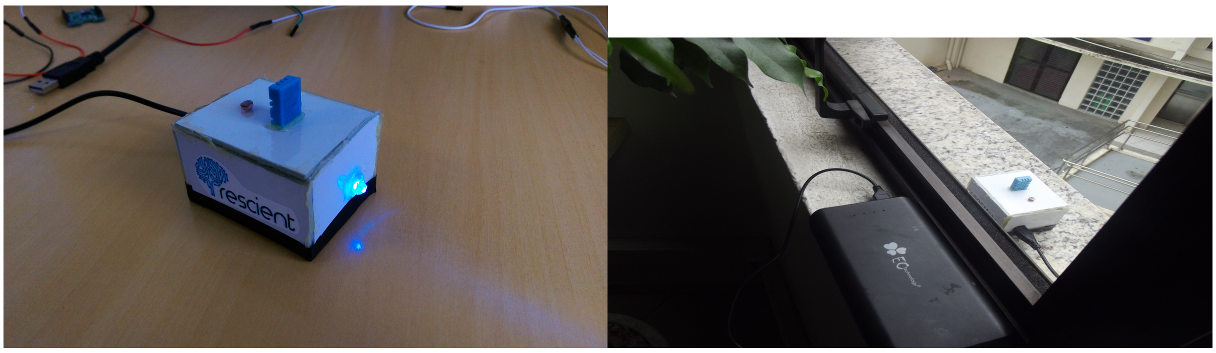


Figure 17 – External sensing node.

5.3 Intelligent Internet of Things Gateway

An embedded control gateway for the Internet of Things extends the functionality of a simple gateway by providing processing resources and intelligence for handling local applications. This can take the form of shared processing resources where the gateway performs tasks that would otherwise occur on nodes. An embedded control gateway can evaluate and filter sensor data as well as implement high-level management tasks. After evaluating and filtering sensor data, a gateway can trigger certain events, according to some thresholds or rules.

Having an intelligent embedded control IoT gateway can reduce the complexity and cost of end points. Depending upon the application, this can result in significant system savings in an application such as a Smart Home, where the gateway connects to an array of sensors. Consolidating processing in the gateway, such as sensor data filtering, enables nodes to leverage a shared resource, making each node simpler and lower in cost.

The same holds for enabling connectivity. IP is a complex protocol to implement with relatively high overhead for simpler IoT nodes. Instead, simple nodes can connect to each other and to the gateway via IEEE 8002.15.4. The gateway then bridges connection to an IP-based WAN interface like Wi-Fi or Ethernet. In both of these cases, savings include lower processing, memory, and power requirements. Therefore, nodes can be less expensive as well as more efficient.

Enabling intelligence in a gateway addresses interoperability issues on a local level while minimizing the changes required to connect appliances. Rather than require full intelligence in each device, the gateway can provide the base intelligence for all devices. An intelligent gateway also better addresses the issues that arise from connecting disparate nodes, compared to users having to connect each device or appliance to the Internet manually.

5.3.1 Sensors and Actuators

A series of sensors and actuators were connected to the Galileo board. The sensors were responsible for gathering the contextual data inside the room, and the actuators for providing user feedback. These sensors require different electronic connections and different code implementations. All of them were placed inside a case developed by the group. Figure 18 shows the resulting gateway case. Each sensor, electronic connection and python implementation are shown in the following sections.

5.3.1.1 LM35

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The read of the temperature can be easily done in the Galileo board by connecting the transducer signal pin to an analog input from Galileo. The python class created to handle this transducer is shown below and is easily imported as a module by the main daemon. The first measurements proved to have a little bit of noise from the power grid. For this reason we used a 1uF capacitor along with the LM35 transducer. It delayed the dynamic of the sensor but improved the readings and, since temperature dynamic is slow, this delay caused no problem to the system. The LM35 transducer and the electronic schematic are shown in Figure 19.

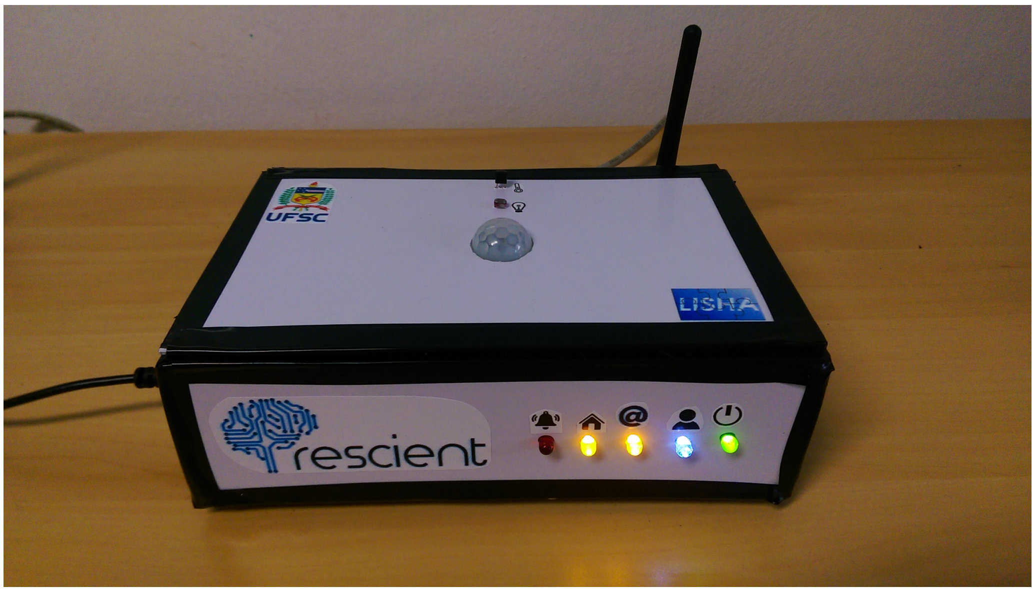


Figure 18 – Galileo gateway case with sensors and actuators.

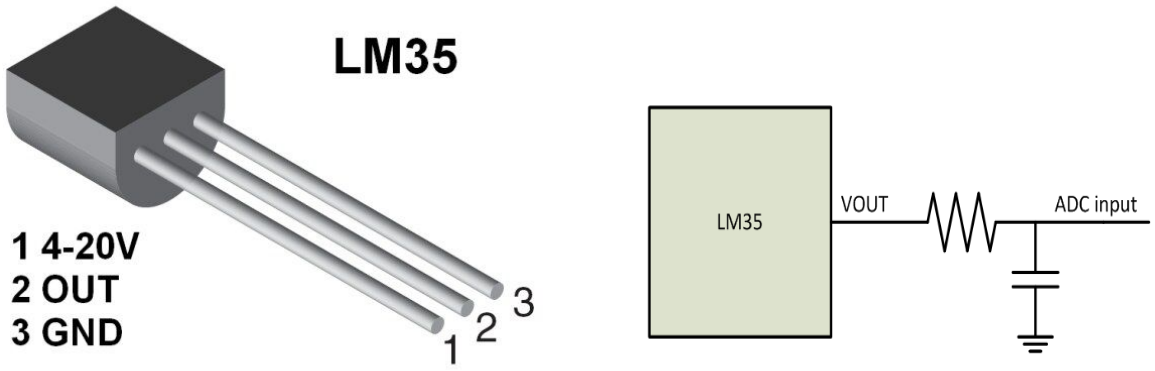


Figure 19 – LM35 transducer and connection schematic.

Source: Texas Instruments.

The python class implementation used by the gateway daemon is displayed in Figure 20. The class is accessible by including it in the Daemon.

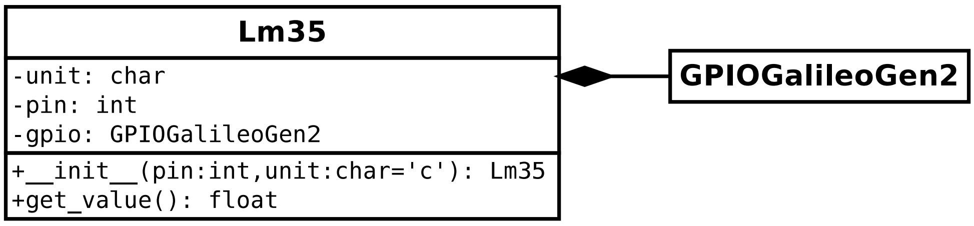


Figure 20 – UML lm35 sensor class diagram.

Heading

```
from sensors.temperature import Lm35
```

Attributes

- `unit`: Indicates the desired unit of measurement, where 'c' is Celsius 'f' is Fahrenheit and k is Kelvin. If none provided takes Celsius as default.
- `pin`: Valid analog input pin. From 14 to 19.
- `gpio`: GPIOGalileoGen2 object to provide access to the I/O pins.

Methods

- `__init__(pin, unit)`: Object initializer. Takes pin and unit as parameters.
- `get_value()`: Reads the current value of the temperature and converts it to the respective unit indicated in the attribute.

5.3.1.2 Light-Dependant Resistor

In the same way as the mobile sensing node developed with EPOSMote, we used an LDR sensor to measure luminosity inside the room. The sensors were almost identical to be able to compare results.

Differently from the LDR class implemented for the EPOSMote III, which was in C++ and required different libraries, Figure 21 shows the one implemented to be used in the gateway.

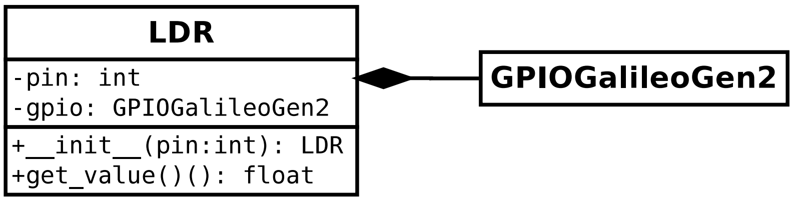


Figure 21 – UML LDR class diagram for Galileo.

Heading

`from sensors.light import LDR`

Attributes

- `pin`: Valid analog input pin. From 14 to 19.
- `gpio`: GPIOGalileoGen2 object to provide access to the I/O pins.

Methods

- `__init__(pin)`: Object initializer. Takes pin number as parameter.
- `get_value()`: Reads the current value of the indicated pin and return in percent.

5.3.1.3 Passive Infrared Sensor

The presence sensor used is shown in Figure 22. The connection of such sensor to the Galileo board is very straight-forward. The Galileo board just needs to do a digital read of the GPIO port to which the sensor is connected. If the read is 1 the sensor is detecting something and, if the read is 0, the sensor is not detecting anything.

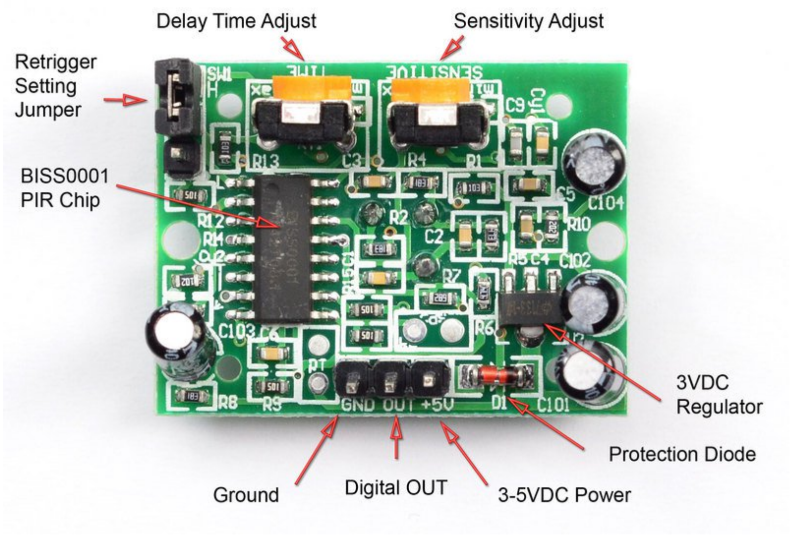


Figure 22 – Presence Sensor.

Source: Adafruit.

The most trivial implementation of all the sensors is the PIR sensor. The python implementation is depicted in Figure 23.

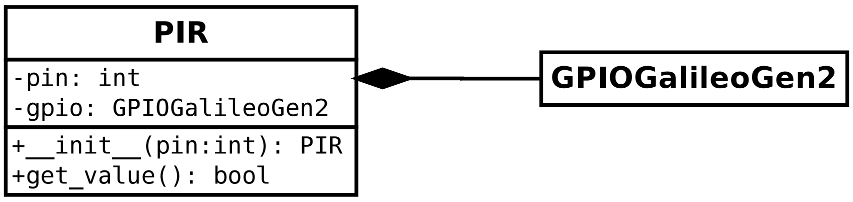


Figure 23 – UML PIR class diagram.

Heading

```
from sensors.presence import PIR
```

Attributes

- `pin`: Valid digital pin. From 0 to 13.
- `gpio`: GPIOGalileoGen2 object to provide access to the I/O pins.

Methods

- `__init__(pin)`: Object initializer. Takes pin number as parameter.
- `get_value()`: Reads the current value of the indicated pin and return the corresponding boolean value.

5.3.1.4 Feedback LEDs

The gateway case also features 5 color LEDs that are used to provide feedback to the user. Every LED has its own meaning, as shown below:

- **Green:** Galileo is turned on.
- **White:** User is present in the network (user_pin).
- **Yellow 1:** The Galileo board is connected to the Internet (internet_pin).
- **Yellow 2:** Blinks when parsing data received from other EPOSMote III nodes (data_pin).
- **Red:** Indicates that the Board is currently not using the context aware engine to predict how to adjust the room (alarm_pin).

The most trivial implementation of all the sensors is the PIR sensor. The python implementation is depicted in Figure 24.

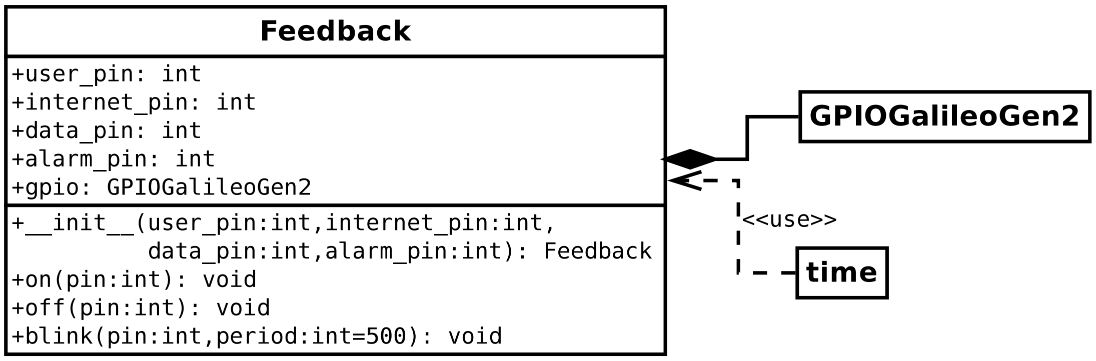


Figure 24 – UML Feedback class diagram.

Heading

`from actuators import Feedback`

Attributes

- `user_pin`: Valid digital pin. From 0 to 13.
- `internet_pin`: Valid digital pin. From 0 to 13.
- `data_pin`: Valid digital pin. From 0 to 13.

- `alarm_pin`: Valid digital pin. From 0 to 13.
- `gpio`: GPIOGalileoGen2 object to provide access to the I/O pins.

Methods

- `__init__(pin)`: Object initializer. Takes pin number as parameter.
- `on(pin)`: Turns the LED indicated by the pin attribute on.
- `off(pin)`: Turns the LED indicated by the pin attribute off.
- `blink(pin, period)`: Turns LED indicated by the pin attribute on/off in periods of milliseconds.

A circuit was also developed to enable connection of all the devices to the GPIO ports of Galileo. Figure 25 illustrates the electronic connections of the sensors to the board while Figure 26 shows the inside of the case developed for our gateway.

5.3.2 User Detection

In this work, the adaptive context-aware system makes predictions based on previous user preferences. In order to detect a user in the room and make the adequate predictions for him, there must be a way of detecting when and which user is inside the room. Merely by using a single presence sensor, it is possible to detect a user inside the room, but not differentiate between them. For this reason, the MAC address of smartphones were chosen to be used as user identifiers. Every person nowadays has a smartphone and every smartphone has a unique MAC address which is accessible through the network.

The Galileo board was connected to a wireless router through an ethernet cable to put both the board and the connected user in the same subnet. If both the user and the board are connected to the same subnet, Galileo can search the network for the MACs which the registered users are assigned. If the MAC address is found on the network sweep done by the gateway daemon, the system knows that it should make predictions based on that user preferences.

User detection is implemented with the help of Linux utilities. The developed python script makes calls to both BASH scripts and shell commands to access the network and verify if a given MAC address is connected. To do that the Nmap tool is used to sweep the network for the user's MAC. Once a user is found, checking if the device is still online is achieved by pinging the assigned IP address. While there are successful pings within a 5-minute interval the user is considered to be online, else wise offline. Since most routers work with IP lease times of close to a week, it possible to optimize

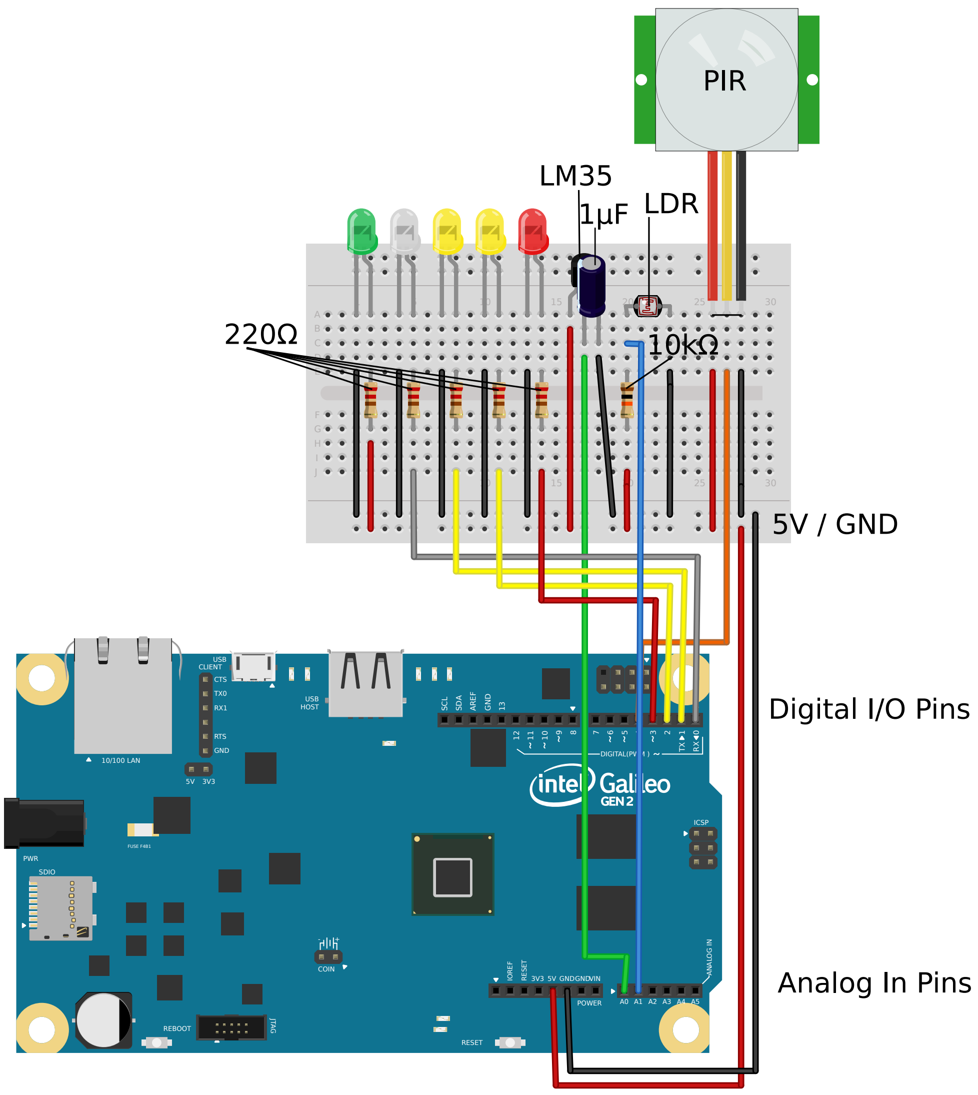


Figure 25 – Galileo Circuit prototype Schematic.

the performance of the script accessing Linux ARP tables. Before sweeping the network, the script pings the last IP address to which the user was connected, if the packet reaches the destination (something is connected to the IP), Linux ARP table entries are checked to verify if the IP matches the MAC address. The class implementation for searching users is illustrated in Figure 27. The flow of user detection process is depicted in Figure 28.

Heading

`from scripts.user_detection import User`

Attributes

- `connected`: Whether the user is currently connected to the network or not.
- `script_path`: Path to the bash scripts used by the `operation`.
- `last_connected`: Last instant that the user was connected to the network.

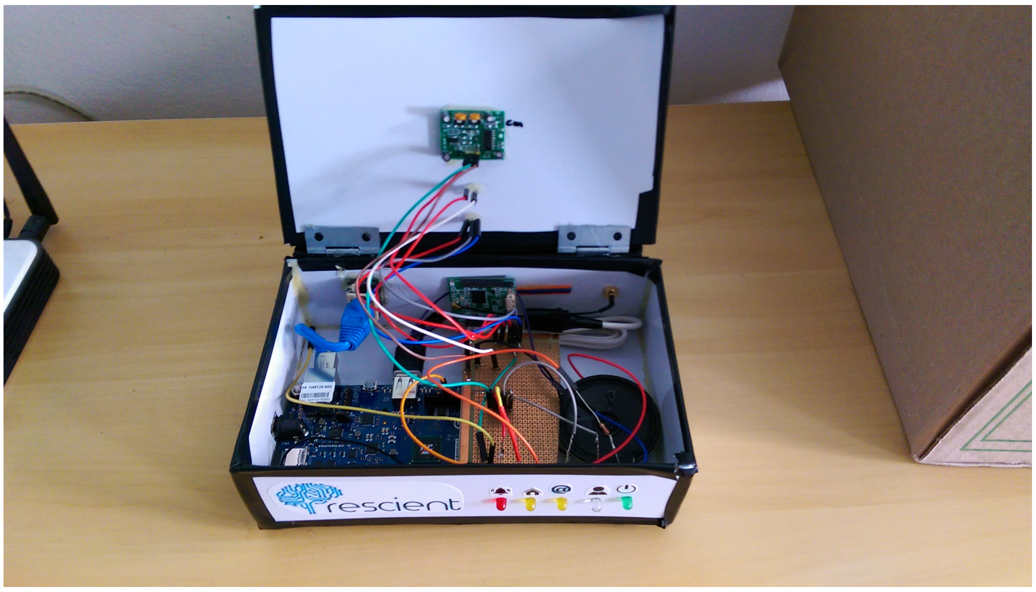


Figure 26 – Inside Galileo Case

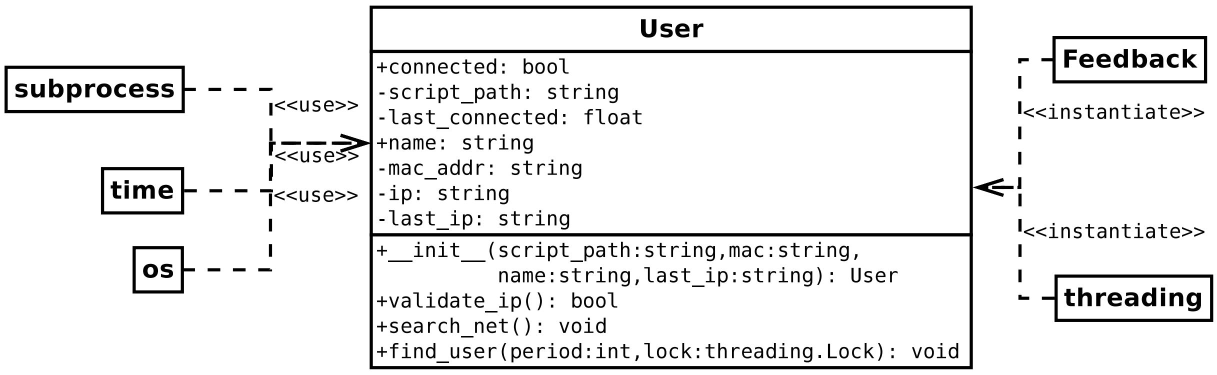


Figure 27 – UML User class diagram.

- `name`: Name of the user.
- `mac_addr`: MAC address of the user's smartphone.
- `ip`: IP address to which the user is currently connected.
- `last_ip`: Last known IP to which the user was connected.

Methods

- `__init__(script_path, mac, name, last_ip)`: Object initializer.
- `validate_ip()`: Checks if the IPs assigned to the user are valid addresses. Returns 1 if IP is valid and 0 if IP is not valid.
- `search_net`: Calls bash script that pings the last known IP and verify if Linuxs ARP table's MAC matches or/and scan the network to verify if the object's MAC

address is found in the up hosts. If the ping is successful but the MAC in the ARP table is different, reset `last_ip`. If the MAC address is found updates both `ip` and `last_ip` variables.

- `find_user(period, lock)`: If the smartphone is not connected, searches the network by calling `search_net`. If the user is connected, pings IP, if no pings are successful in a 5 minute interval set user as not connected. The lock attribute is used to prevent that 2 User objects search the network at the same time due to the memory usage of the Nmap tool.

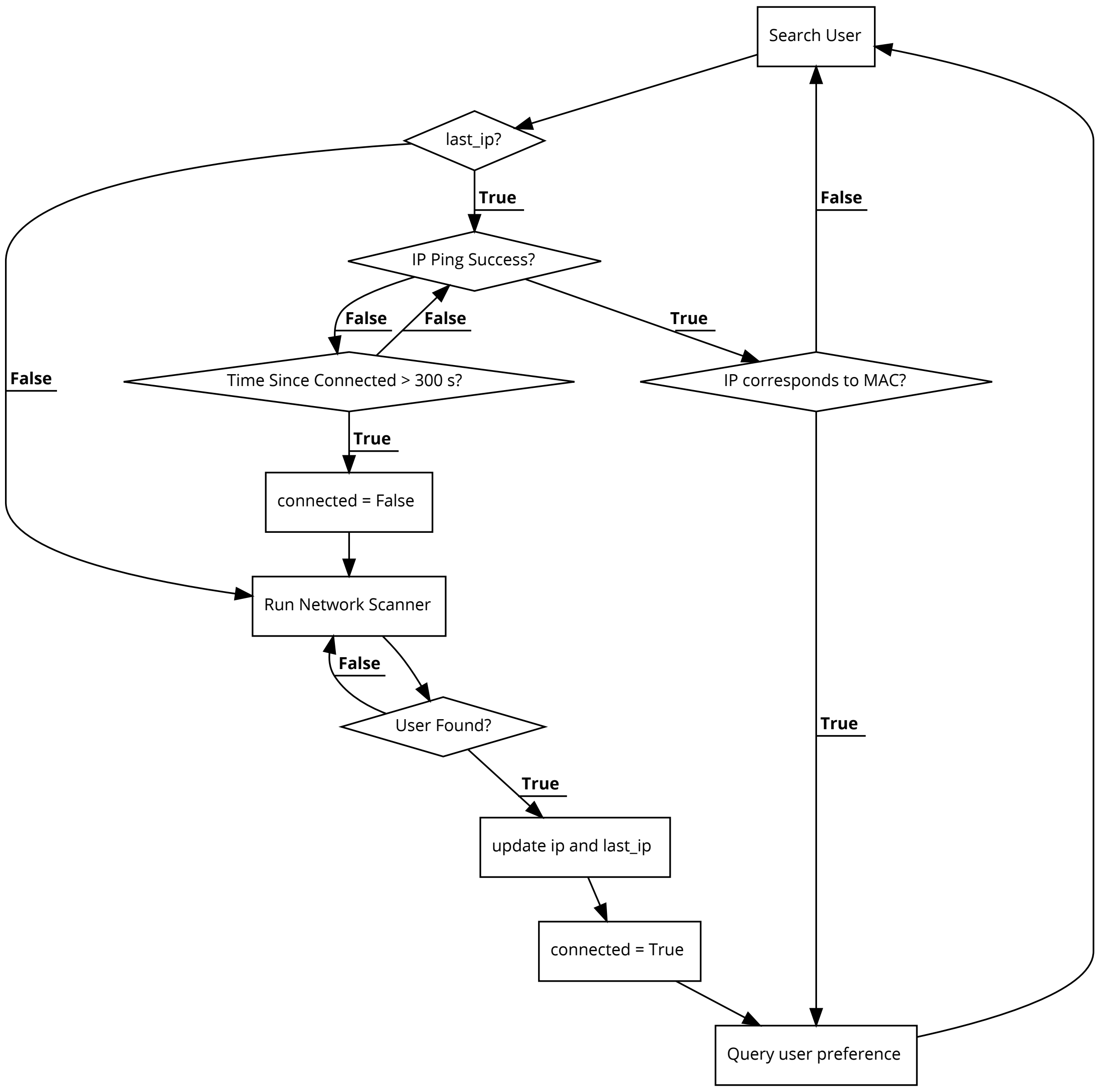


Figure 28 – User Detection Flow Chart

5.3.3 Internet Connectivity Checking

Checking internet connectivity is much easier than searching for users in the network. We just try to ping http://www.google.com/. When ping occurs with success, the gateway is connected to the internet. The connectivity verification is very useful. If the system is not connected to the web, then the data points that were previously stored in the server, must be stored locally until connection to the server is restored. The same principle holds for querying predictions for the user. The Gateway maintains a local copy of the neural network weights that were trained in the server[1], so when the internet connectivity is out, it processes the predictions on the Galileo board itself. The python implementation is depicted in Figure 29.

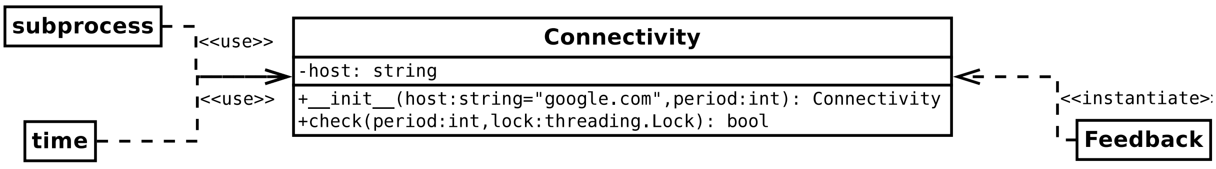

subprocess
<<use>>
Connectivity
-host: string
+__init__(host:string="google.com",period:int): Connectivity
+check(period:int,lock:threading.Lock): bool
<<use>>
time
<<instantiate>>
Feedback

Figure 29 – UML Connectivity class diagram.

Heading

`from scripts.connectivity import Connectivity`

Attributes

- `host`: Desired host address. Set to "google.com" by default.

Methods

- `__init__(period, host)`: Object initializer.
- `check()`: Issues a ping per second to the host to verify if it is online, if no pings are successful within 5 seconds connectivity to host is down. Returns 0 if connectivity to host is down and 1 if up.

5.3.4 Integration with EPOSMote III

To integrate the Internet with the Network of things (IEEE 802.15.4) an EPOSMote III was connected to the Galileo board with a USB cable. This mote is the central node of the network. This EPOSMote III is programmed to forward every packet received from other nodes to the gateway via Serial USB connection.

[1] The Galileo board does not have enough processing resources to train a neural network, but it can process the feed forward to get an output prediction on account of a little system overhead if connectivity is out.

On the Galileo side of the Gateway, there is a thread running that is responsible for parsing the Modbus messages and converting it to the adequate units according to the Modbus ID in the message received. The serial connection is kept open, between the mote and the Galileo board, and for that reason it is necessary to always issue a serial buffer flush before reading the messages. Otherwise, depending on the period between the serial reads, the Galileo board is going to parse messages received a long time ago. A Gateway serial read is considered complete when all the IDs from the registered devices are read.

The class developed to handle the EPOSMote III connection in the gateway is depicted by the diagram in Figure 30.

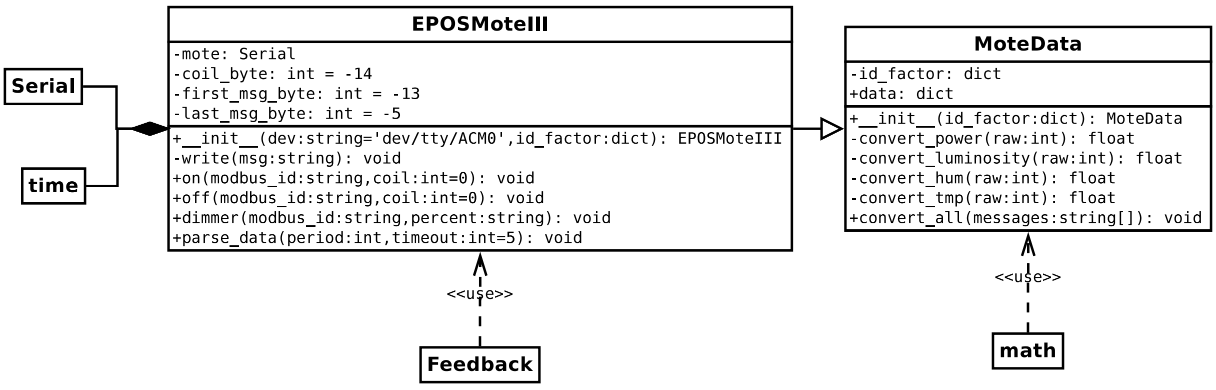

EPOSMoteIII
-mote: Serial
-coil_byte: int = -14
-first_msg_byte: int = -13
-last_msg_byte: int = -5
+__init__(dev:string='dev/tty/ACM0',id_factor:dict): EPOSMoteIII
-write(msg:string): void
+on(modbus_id:string,coil:int=0): void
+off(modbus_id:string,coil:int=0): void
+dimmer(modbus_id:string,percent:string): void
+parse_data(period:int,timeout:int=5): void
MoteData
-id_factor: dict
+data: dict
+__init__(id_factor:dict): MoteData
-convert_power(raw:int): float
-convert_luminosity(raw:int): float
-convert_hum(raw:int): float
-convert_tmp(raw:int): float
+convert_all(messages:string[]): void
Serial
time
<<use>>
Feedback
<<use>>
math

Figure 30 – UML EPOSMoteIII class diagram.

Heading

`from sensors.eposmote import EPOSMoteIII`

Attributes

- `mote`: Serial object responsible for the connection with the microcontroller.
- `coil_byte`: Modbus message positional byte which corresponds to the coil number. Varies depending on dev.
- `first_msg_byte`: Modbus message positional byte which corresponds to the beginning of the Modbus messsage. Varies depending on dev.
- `last_msg_byte`: Modbus message positional byte which corresponds to the last message byte. Varies depending on dev.
- `id_factor`: Key:value pairs with device ID and conversion factor.
- `data`: Stores the resulting key:value pairs in the data dictionary.

Methods

- `__init__(dev)`: Object initializer. Takes the device port as parameter, set to '/dev/tty/ACM0' by default, and devices' IDs and conversion factors.
- `write(msg)`: Writes a Modbus message and sends it.
- `on(modbus_id, coil)`: Composes and writes the Modbus message to turn on the device with `modbus_id` at the given coil.
- `off(modbus_id, coil)`: Composes and writes the Modbus message to turn off the device with `modbus_id` at the given coil.
- `dimmer(modbus_id, percentage)`: Composes the Modbus message to dimmer the device with `modbus_id` by the percetage. 0 dimerization means maximum power and 99 corresponds to the minimum.
- `convert_power(raw)`: Takes the raw value from the Modbus message, verifies if it is between the possible thresholds and uses the conversion factor to turn the raw value into Wh.
- `convert_luminosity(raw)`: Takes the raw value from the Modbus message, verifies if it is between the possible thresholds and uses the conversion factor to turn the raw value into %.
- `convert_hum(raw)`: Takes the raw value from the Modbus message, verifies if it is between the possible thresholds and uses the conversion factor to turn the raw value into %.
- `convert_tmp(raw)`: Takes the raw value from the Modbus message, verifies if it is between the possible thresholds and uses the conversion factor to turn the raw value into degree Celsius.
- `convert_all(messages)`: Compares the array of Modbus messages with IDs and factors from `id_factor` and issues the appropriate unit conversions. Results from the conversions are stored in data.
- `parse_data(period)`: Runs as a thread in the Daemon. Periodically flushes the serial buffer, waits until all the Modbus messages with IDs listed in `id_factor` are received or uses all the messages received before the timeout.

From times to times occurred segmentation faults in the daemon and it crashed. The problems was identified as a problem with the serial connection between the Galileo board and EPOSMote III. For some reason, the serial connection at some point stopped working and it implied in a future segmentation fault. To solve this issue, along with the parser timeout was added a USB reset code developed in C. If at any point a timeout occurred during data parsing due to unavailability of the serial connection (e.g. no data received in the whole period) the python program made a call to the systems shell and executed the USB reset program.

5.3.5 Server Communication

Communication with the cloud server is straightforward. If the gateway is connected to the internet, it sends data to the cloud whenever possible. However, if internet connectivity is not available, all communication with the server is stopped, and the data is stored in the gateway until connectivity is restored.

The gateway uses an HTTP REST API to enable communication with the cloud server. Everything is done in python using the requests module. Data from the Daemon is POSTed to the cloud server in JSON format to be stored and, when prediction queries are done to the server, GET requests are used. The sequence diagram shown in Figure 31 exemplifies the process. A mote reads the value from the sensor and reports it to the relay mote, connected to the gateway, via Modbus message. The relay then forwards the message to the gateway, which parses the message (Modbus decoding, data filtering and unit conversion) and issues an HTTP Post to the Server.

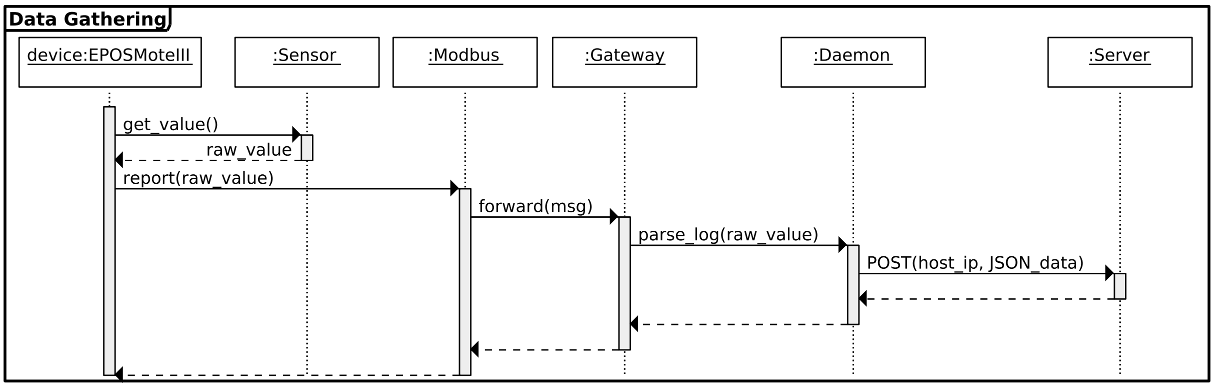


Figure 31 – Sequence Diagram of Data Gathering and Server Storage.

The other part of the server communication consists of answering queries from the mobile application for current data or remote device control. The handling of HTTP requests is done with Flask, a python microframework for web development.

5.3.6 Daemon

The Daemon is the core of the gateway. It integrates all the above described components into a parallel and synchronized environment. The implementation is illustrated by the diagram in Figure 32. The attributes and methods are described below.

Attributes

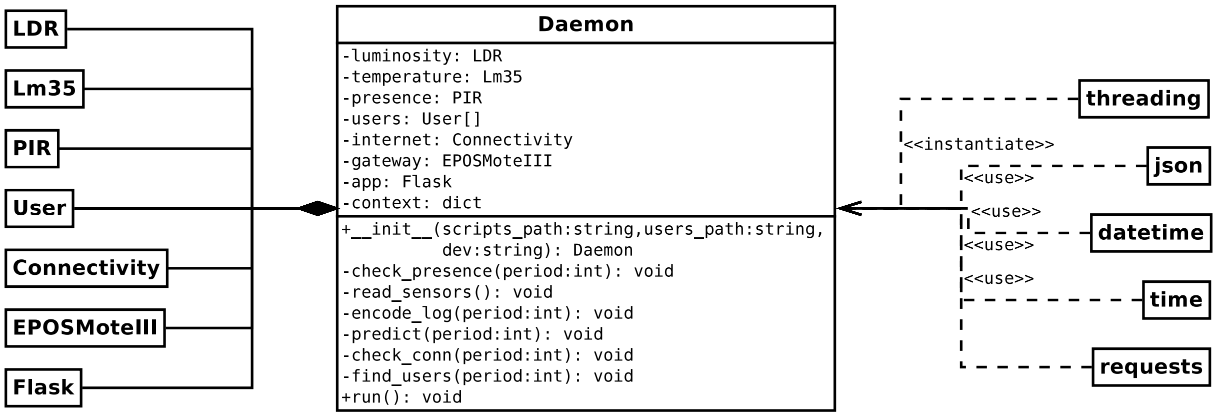


Figure 32 – UML Daemon class diagram.

- `luminosity:` Object of Class LDR. Deals with the luminosity sensor from the Galileo Board.
- `temperature:` Object of Class Lm35. Deals with the temperature sensor from the Galileo Board.
- `presence:` Object of Class PIR. Instantiates the PIR sensor connected to the Galileo board.
- `users:` Array of User objects. Objects are created accordingly to contents of the file referred to by `users_path`.
- `internet:` Connectivity object. Used to verify web connectivity availability status.
- `gateway:` Instance of the EPOSMote III connection to the Galileo board.
- `app:` Flask object that instantiates the REST request handlers.
- `context:` Key:Value pairs representing the most recent data collected.

Methods

- `__init__(scripts_path, users_path, dev):` Daemon initializer. Takes paths to important files required by the objects and the device port to which the mote is connected.
- `check_presence(period):` Loop function that every `period` of seconds checks the presence sensor and alters the context variable. Additionally, turns the lights on/off if no registered user is connected but a person in the room is detected.
- `read_sensors():` Gets the values from the luminosity and temperature every `period` seconds and updates the respective entry in the context variable. Also copies the `gateway.data` key:value pairs to the context variable.

- `encode_log(period)`: Every `period` seconds, calls `read_sensors` to refresh the context variable with current data, and encodes the context dictionary into a JSON string and issues a PUT request to the server.

- `predict(period)`: Periodically makes device control predictions for the User who was first detected. If internet connection is available queries the predictions from the server. If connection to the internet is not available, but the user file contains a copy of the ANN weights calculates the output locally, else does nothing.

- `check_conn(period)`: Updates periodically the status of the internet connectivity.

- `find_users(period)`: Reads the user file in `user_path` and for every user creates an object and a `users[].find_user(period, lock)` thread that periodically search the network for connected users or verifies if they are still connected.

- `run()`: Creates all the threads (`gateway.parse_data(period)`; `app.run(host, port)`; `check_presence(period)`; `encode_log(period)`; `predict(period)`; `check_conn()`; `users[].find_user(period, lock)`) and start the threads.

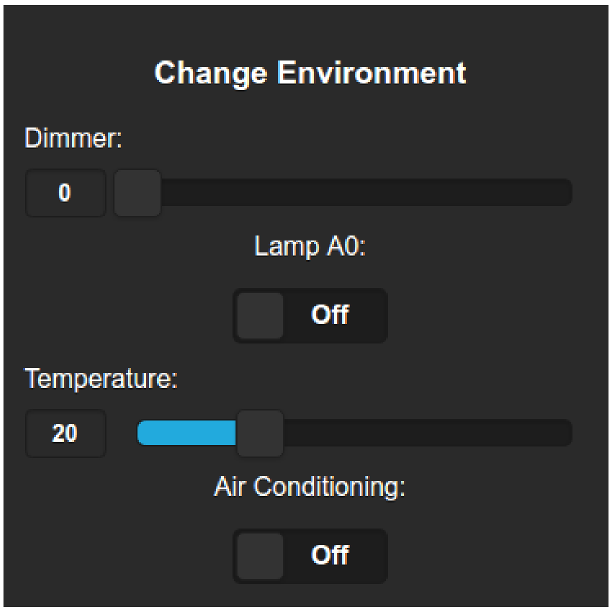
5.4 Hybrid Application

Hybrid mobile applications are built in a similar manner as websites. Both use a combination of technologies like HTML, CSS, and JavaScript. However, instead of targeting a mobile browser, hybrid applications target a WebView hosted inside a native container. This enables them to do things like access hardware capabilities of the mobile device. This enables the generation of cross-platform applications, yielding many advantages such as, if bug is found in a common codebase it needs to be fixed only once, it is possible to use existing programming talent rather than learning platform specific development language. This results in faster development and reduced costs.

5.4.1 Remote Device Control

One of the features that cannot be missing in any new Internet of Things device is remote control. A user should be able to remotely turn devices on and off, independently of where he is. If the user left the room and only later realized that he forgot the lights on, instead of going back to the place to turn the lights off, he must be able to do that from where he is.

In LISHA's Smart Room, as shown in Section 5.2, there are devices capable of being remotely controlled. One of the features included in the app, is a set of controls that, allow the user to select his preferred settings. The person can choose both turning



Change Environment

Dimmer: 0

Lamp A0: Off

Temperature: 20

Air Conditioning: Off

Figure 33 – Available options for the user in the application.

on and off the devices as well as adjusting the power consumption (light power is controlled through dimerization).

To do that, a graphical user interface (GUI) was developed where the user can chose the appropriate settings. This interface is shown in Figure 33. The event handlers for these buttons were implemented using JQuery.

Communication between the application and the gateway is achieved through an HTTP REST API from Jquery called AJAX. Whenever a button is pressed or the slider is set to a new value in the user interface, the application sends a GET request to the server. The server then relays this command to the gateway. The command is received by the Flask request handler that accesses the EPOSMoteIII class and writes the Modbus message to control the physical device. This process is illustrated by the sequence diagram in Figure 34.

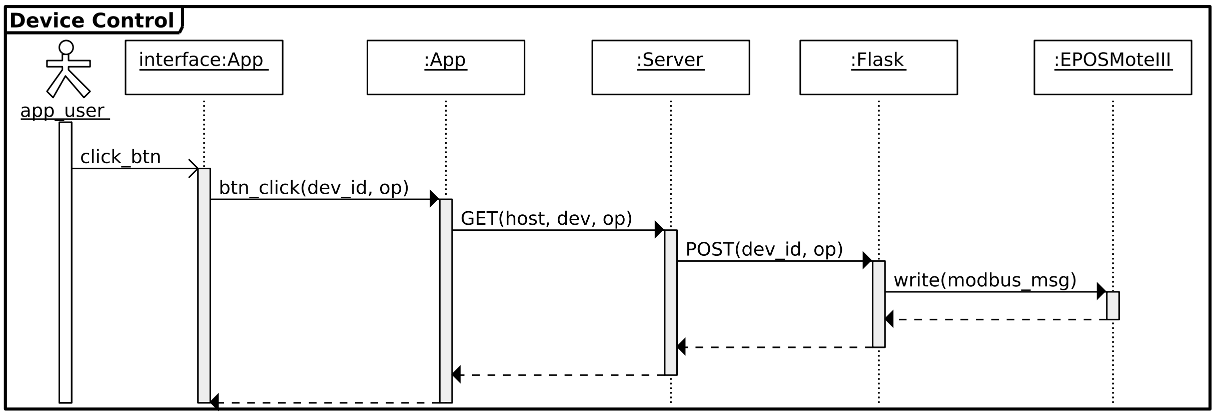


Figure 34 – Sequence Diagram of remote commands from app to gateway.

5.4.2 Real-time Data Visualization

Another very important feature that an application for the Internet of Things must have is real-time data visualization. A user must be capable of checking the current power consumption of the devices, temperature and every other information available about the smart things.

Observing data such as energy consumption has a very important influence on user behavior. The user may notice that an appliance connected to a smart power outlet is damaged and consuming double the power it should. The user can also observe some behaviors that cause high energy consumption and change the way he behaves in order to reduce his energy consumption footprint.

Data update, the same way as the remote control feature, is also done through HTTP GETs to the server. A page named "Context" was implemented in the application to show every available environment data from the Room. Another page named "Energy" was developed to focus on energy consumption data. We also chose to use a JavaScript charting module (ZingChart), that allows the user to see a time series of the consumption in a 2 minute window. Both pages show the current value for every measurement. Figure 35 shows the resulting application interface running on an Android smartphone.

Queries to the server for current data were also done using HTTP GET requests periodically. The message from the server, in this case, is a JSON object with values for all the devices available.

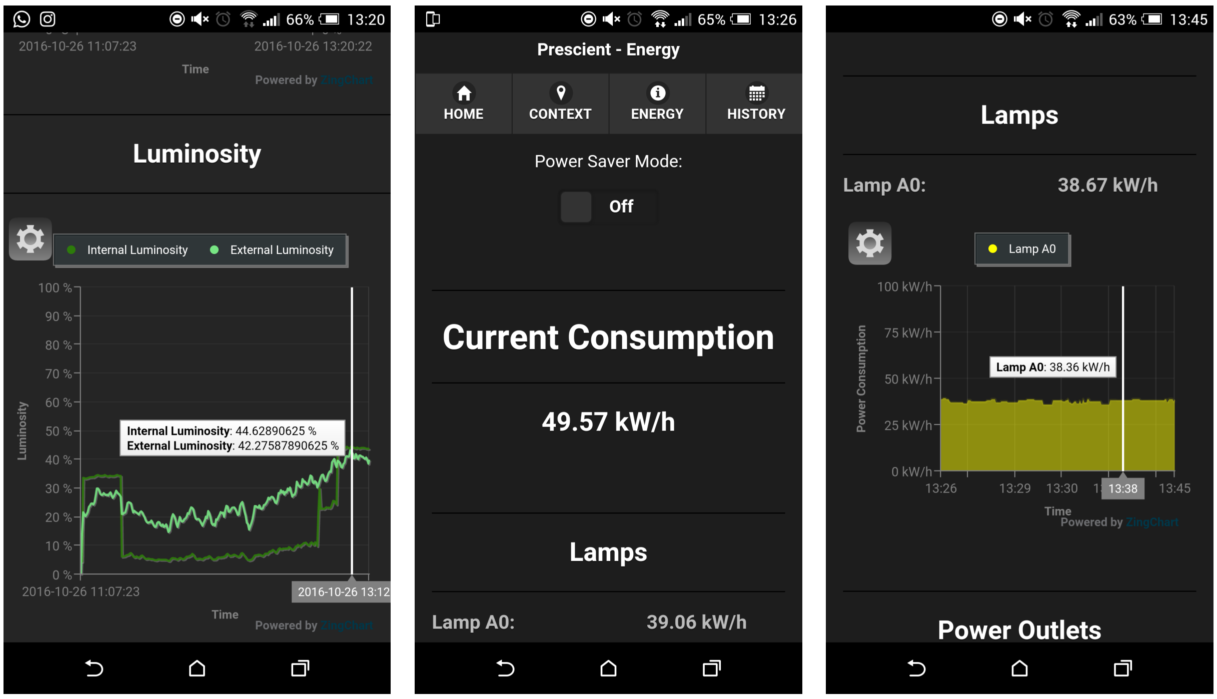


Figure 35 – Application Context and Energy Pages on Smartphone Browser.

5.4.3 Data History Visualization

Data history visualization allows the identification of periodic patterns and correlation between the devices. Therefore we've developed another page in the application named "History", that shows data over time for the wanted period. Figure 36 shows the result of the implementation for the query of the last 2000 data points.

Similarly to the previous described features, history is queried using HTTP GETs to the server. The application receives a list with the defined number of data points for the variable.

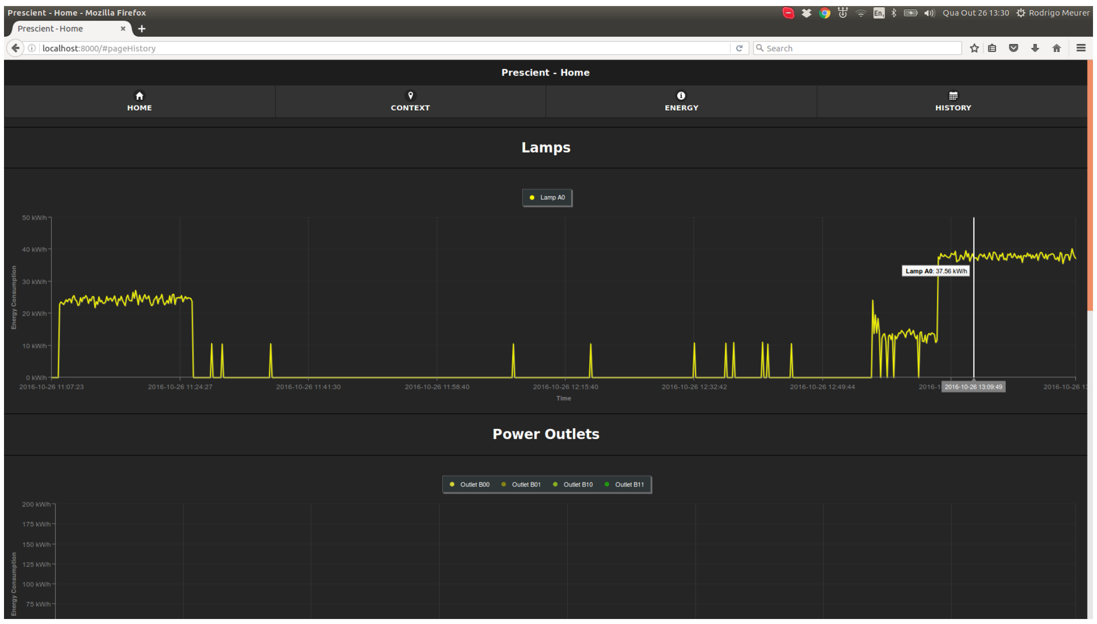


Figure 36 – Light power consumption over time (Last 2000 datapoints).

5.5 Adaptive Context-Aware Algorithm

A context-aware algorithm must be able to know how to control every device available according to current room context. In this project, the control predictions are made through a combination of machine learning and data mining techniques. Data mining technique, Random Tree Classifier, is applied to the dataset to identify the most relevant data relations. The DNNs (MLPs) are used to classify the output of a certain device according to relevant contextual data. Finally, fuzzy logic is applied based on heuristics to optimize the usage (energy consumption, comfort, etc.) of the device. Predictions are individually tailored for every user. Profiles are distinguished by the smartphone's MAC address connected to the gateway network.

A python class was developed to use DNNs to classify a device's output according to relevant data from the current context. An object of this class should be initialized with the name of the device, the path for a simple training set containing data for basic configuration, the path for the test set, the file path of the user's corrections to the engine decisions and a number of inputs and outputs for the files. With this information, a Brain object is created for predicting classification of the output in the current context, while relevant data is inferred from the datasets.

5.5.1 Contextual Data

As stated in Sections 5.2 and 5.3, a series of sensors were deployed in the room to get as many information as possible. In the scope of this project, the context of the smart room is defined by the following set of measurements:

- `A0`: Whether light A0 is turned on or off.
- `A0_pw`: light A0 power consumption.
- `AC`: Whether the Air Conditioner is turned on or off.
- `B00_pw`: Power consumption for the first plug in the power outlet B0.
- `B01_pw`: Power consumption for the second plug in the power outlet B0.
- `B10_pw`: Power consumption for the first plug in the power outlet B1.
- `B11_pw`: Power consumption for the second plug in the power outlet B1.
- `year`: Year of the measurement.
- `month`: Month of the measurement.
- `day`: Day of the month of the measurement.
- `weekday`: Day of the week of the measurement.
- `hour`: Hour in which the measurement was made.
- `minute`: Minute in which the measurement was made.
- `external_lum`: Luminosity outside the smart room.
- `internal_lum`: Luminosity inside the smart room.
- `external_temp`: Temperature outside the smart room.
- `internal_temp`: Temperature inside the smart room.
- `smartphone`: Whether user smartphone is connected or not.
- `present`: Whether presence sensor is detecting user or not.

- `humidity:` Air Humidity.

This data composes four sources of information for the context-aware decision engine:

1. **Prior Knowledge Data (PKD):** Data that may or may not be introduced by the user (or factory) to initialize the decision engine with pre-defined rules. Prior knowledge may come from observations of the control rules from SCADA systems. The more possibilities this data covers, the better the results obtained from the output of the decision engine training. Data from rules can be a subset of the whole contextual dataset and are general for every user. PKD can be interpreted as the factory control settings for the device.

2. **Room Data (RD):** For the great part of the day (night) there is no user present in the room. Data stored from that period is what is called the room data. For itself, the room data does not provide much relevant information regarding the specific device control for the users. However, when used along with User data, the decision engine can learn the impact of a certain user being detected in the room.

3. **User Room Data (URD):** User room data is all the contextual information for the period that a given user has spent inside the smart room.

4. **User Commands Data (UCD):** The decision engine periodically predicts device adjustments for the user. There may come a time when the user does not agree with the changes made by the decision engine. When that happens, the user uses the hybrid application to adjust the devices to his preferred settings. The new configuration chosen by the user are stored separately. Predictions from the decision engine that are successful (without negative feedback from the user in 10 minutes) are also incorporated by the user commands data.

5.5.2 Context-Aware Decision Engine Training

Training machine learning techniques is not a trivial task. Many aspects must be taken into account when designing the data mining and machine learning algorithm structures. Otherwise results may be invalid or misinterpreted. In ANNs, for example, the usage of irrelevant or noisy data may affect the accuracy of the training and model the system incorrectly. The usage of insufficient data points to represent a given behavior may yield in the same result. In opposition, the use of too complex ANN topologies or/and strict parameters in data mining algorithms shall over fit the model and incapacitate it from generalizing the output well enough.

That is why, in this work, a novel approach is developed aiming at the control of devices using contextual data. Figure 37 depicts how the distinct datasets are used to model user's preferences and control devices. Each step is further discussed in the following sections.

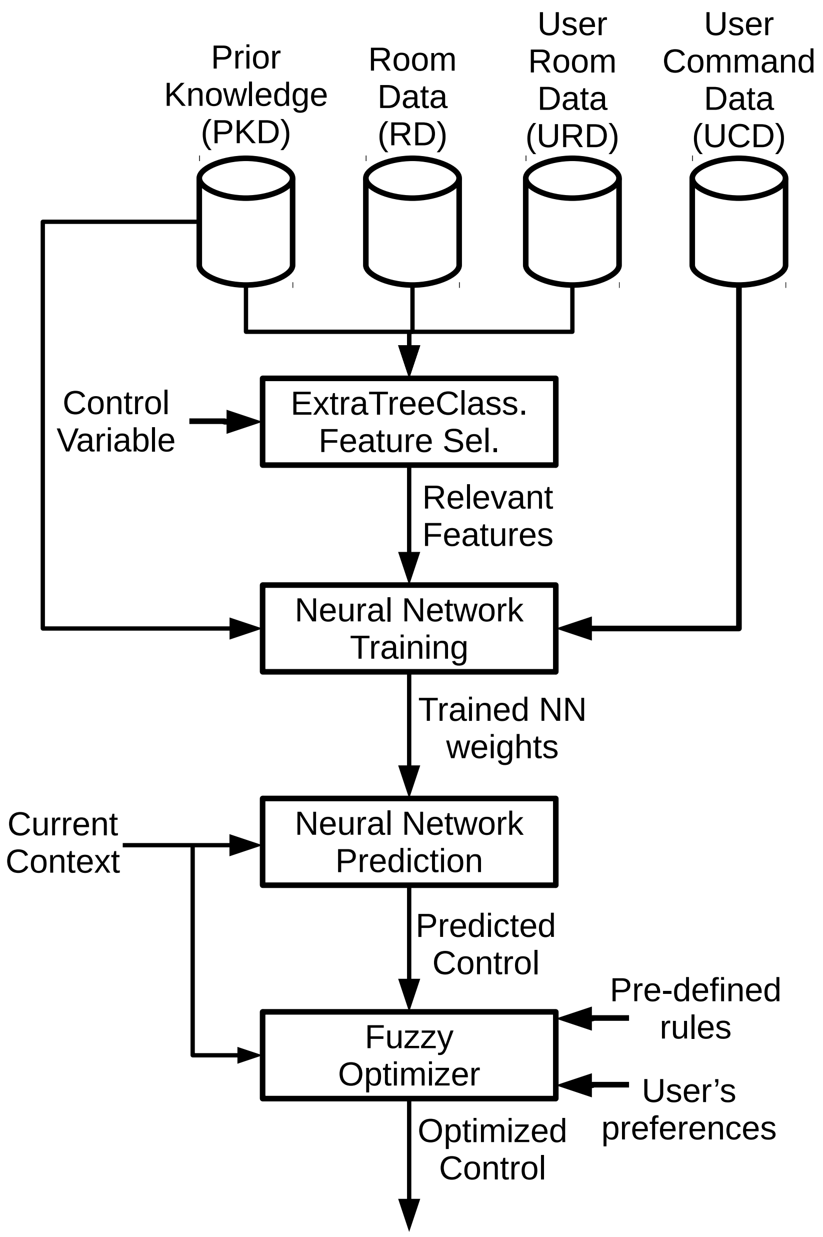


Figure 37 – Neural Network relevant feature selection and training.

5.5.2.1 Automatic Selection of Relevant Features

In this work, context is defined by a set of 20 variables. Training a Neural Network with 20 inputs would be a cumbersome process. Not only it would take a very long time to process, but the data that would be needed to train such network would have to cover almost every possible scenario. Else wise the accuracy of the neural network predictions would not be trustful. Therefore, reduction of the data space is needed. This reduction yet must account for the relevance and representativity of a given variable to the control of an individual device.

Selecting data relevance is no trivial task, however, especially when data variables have different meanings and limits. Temperature, for example, may assume a broad range of positive and negative values, while the luminosity is measured in percent from 0 to 100. The selection of a data mining method able to identify the adequate features to train a neural network is substantial for the success of the training.

An algorithm that can attend to that purpose is the Extra Trees Classifier. It implements a meta-estimator that fits some randomized decision trees on various subsamples of the dataset. This method also uses averaging alongside with other statistical tools to improve the accuracy of feature importances.

This algorithm is fed with three of the data source previously mentioned. Prior knowledge data, room data, and user room data. Additionally, the target output variable is informed so that the algorithm can rank the features accordingly. There are data sources that may be directly affected by the output of the controlled device, which in the algorithm would mean false relevances. Therefore, these variables are marked and taken out of consideration when selecting the relevant features. The feature importance ranking is expressed in percentage, where the sum of all importances for the control accounts for 100%. Relevant features for a given output are the top N variables with relevance above a given threshold.

5.5.2.2 Context-Based Adaptive Learning

Contextual data is used to classify how devices should operate. This classification processes in this work is done through trained neural networks that learn from user behavior how to classify the output or mode of operation of an individual device according to previous data from the user and environment interaction.

Prior knowledge (e.g. rules already developed for other systems such as SCADA) can be incorporated into the neural network to avoid the cold-start phase[2] of machine learning systems. When no previous knowledge is provided to the system, it causes a trade-off. The sooner the system begins to control the room, the less representative data it has, and, therefore, the worst will be the predictions that it will make. On the other side, the longer the system gathers representative data, better the accuracy of the predictions, but longer the intelligent control system idle time. The features provided by the prior knowledge data are considered relevant.

[2] Cold-start is a potential problem in computer-based information systems which involve a degree of automated data modeling. Specifically, it concerns the issue that the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information.

Prior knowledge only, however, not necessarily best represents the user's preferences. The user may disagree with a prediction that was made by the system based solely on prior knowledge (e.g. factory rules). In this case, the person would adjust the settings of the room accordingly to his preferences. This sort of scenario is approached with reinforcement-learning-like[3] feedback technique. When the decision engine makes a correct device control prediction, and the user does not correct it (e.g. does not change any device) within a period of 10 minutes, this prediction is appended to the user command data. After that occurs, no other prediction is added to the dataset while the predictions values do not change. A correct prediction is considered to have the same weight of a user command. When the user issues a command after a prediction from the system, it represents a negative feedback to the decision engine, which means that the predicted device control was not ideal. This feedback triggers a back-off for the retraining of the control ANN of every device affected by the user feedback. During the back-off period, the system does not query any predictions. The predictions are resumed either after the ANN is completely trained and a fixed time has elapsed, or a relevant change in context occurs. The feedback context (User Command Data) is joined with the prior knowledge data to compose the ANN's new training set.

As stated previously, the topology of the ANN may vary in the number of layers, the number of neurons per layer, inputs, outputs, activation functions, error function, optimizer, batch sizes, etc. There are no known heuristics or formulas for determining the best topology of an ANN. This process is usually done through trial and error. Thus, the decision engine should be capable of automatically generating diverse network topologies and cross-evaluating the accuracy achieved by each one to chose the best. The cross-evaluation is done by shuffling the User Room Data and taking 1/3 of it to check the accuracy of the predictions. The ANN topology with highest accuracy score is implemented to make the predictions.

This composition of data mining and machine learning techniques provides the context-aware decision engine with an adaptive capacity. The system can learn along with user experience inside the given environment which are the relevant environment variables to control a given device and how they were controlled previously by the user.

5.5.3 Preference Prediction

Preference prediction is very straight-forward. After the ANNs have already been trained, the calculations to obtain the output are simple matrix multiplications. When the engine receives a query to predict the control of a given device, it extracts the

[3] Reinforcement Learning is a type of Machine Learning, and thereby also a branch of Artificial Intelligence. It allows machines and software agents to automatically determine the ideal behavior within a specific context, to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

relevant data from the current context and feeds it to the ANN. The ANN makes the multiplications and generates the classification of the device control.

5.5.4 Fuzzy Optimizer

Users are not used to considering energy efficiency when controlling room devices. Lights are usually just turned on, and not dimerized accordingly to the sunlight or any other factor along the day aiming to save energy. For that reason, trying to impose this kind of behavior into a neural network may require the user to be continuously aware of the power consumption of the devices An intelligent gateway, however, must be able to optimize energy consumption.

Fuzzy logic controllers are an excellent approach to solving the power consumption optimization problem. Most people do not know how to model a mathematical model that would control something according to his preferences. However, most people can describe their preferences qualitatively, or even, how to optimize energy with little impact on comfort, based on IF-THEN statements.

For example. Describing a control equation that adjusts the temperature of the room to optimize the air conditioning system's power consumption without considering human comfort is a very complicated thing to do. In counterpart, it is very easy for a person to describe how to improve the power consumption based on qualitative terms such as cold, cool and warm. Rules defined by this person would be something like *"If the temperature inside is COLD and the temperature outside is HOT, THEN AC power should be set to LOW"*. Figure 38 depicts an example of a fuzzy set with triangular membership functions. From the rules and fuzzy sets provided, the controller calculates the power of the AC.

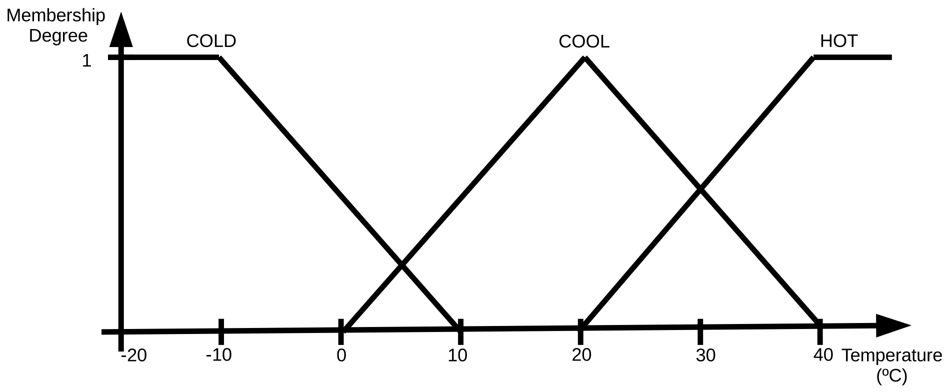


Figure 38 – Fuzzy temperature set example.

This sort of energy efficiency heuristics are common knowledge and, therefore, easily generalizable, which makes the system even better for the user since he does not need to think of the rules. The user may adjust the intervals (e.g. COOL temperature ranges from 10 to 30 instead of 0 to 40) to his preferences improving his comfort while optimizing energy efficiency. For these reasons, fuzzy controllers were chosen to be used as optimizers after the output of the ANNs.

In this project, a fuzzy controller that uses internal and external luminosity was implemented to adjust the devices in a way that power is saved and user comfort is not much affected. This optimizer can be used by activating the power-saving mode in the user application. If the user feels that the adjustments are not to his liking, he can turn it off.

Figures 39, 40 and 41 illustrate how the luminosity variables are split in fuzzy sets to be used in the optimizer.

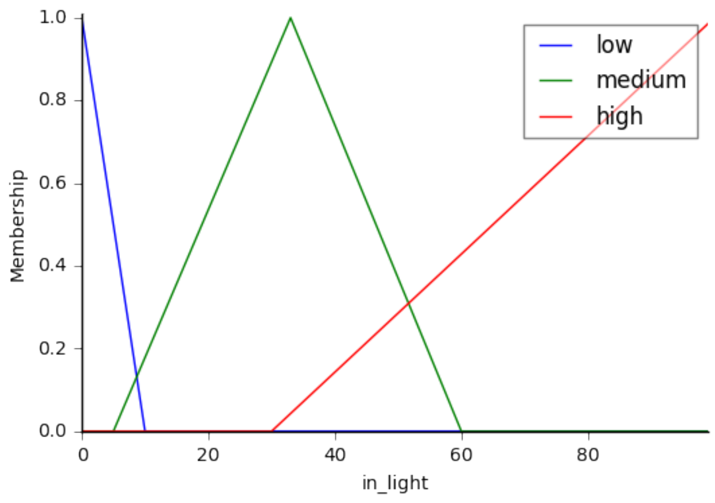


Figure 39 – Fuzzy set representation for internal luminosity.

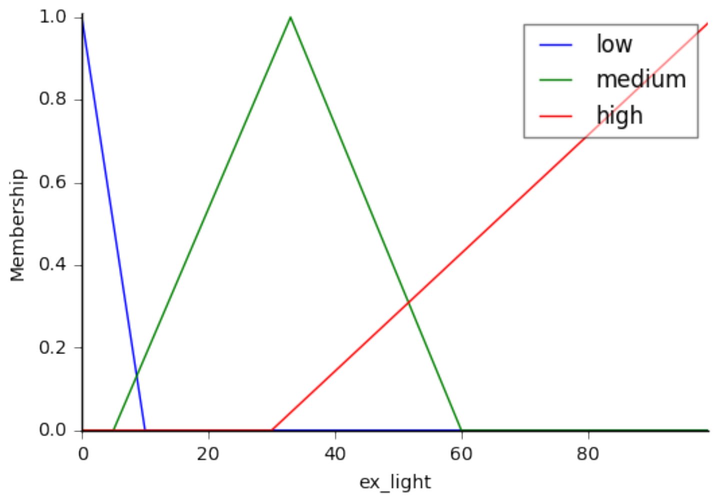


Figure 40 – Fuzzy set representation for external luminosity.

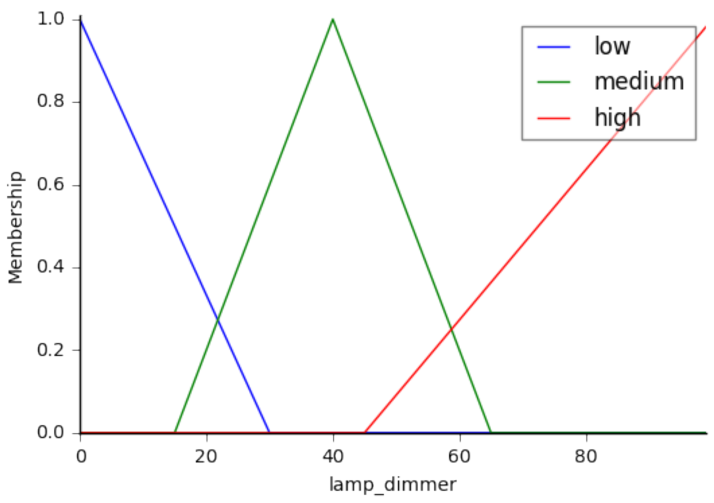


Figure 41 – Fuzzy set representation for light dimmerization.

These fuzzy sets are brought together through fuzzy rules that consider energy optimization in their core. Take one of the rules for example:

```
IF (in_light is LOW && ex_light is HIGH) THEN (set lamp_dimmer HIGH)
```

It states that, if internal luminosity is low but external is high, there is no need to turn the lights on at full brightness, high dimerization would save energy.

All rules described by an optimizer are processed, and an output value is calculated. This output is then passed on to the gateway so that the device is adjusted for lower power consumption.

5.5.5 Adaptive Context-Aware Decision Engine Implementation

The decision engine, which was named Prescient, runs on a server and is composed of two parts. The intelligence encapsulates the complete implementation for the data mining and machine learning algorithms. The services encapsulate the HTTP Server and handle the REST requests from the application and the gateway.

The intelligence is encapsulated in a Class called Brain. Brain objects are created to learn how to control a particular device. They implement the whole learning and predicting process described in section 5.5 and may or may not use optimizer objects that extend fuzzy controllers. Figure 42 presents a UML class diagram that depicts the implementation of the Classes that compose the intelligence of the decision engine.

Heading

```
from NN import Brain
```

Attributes

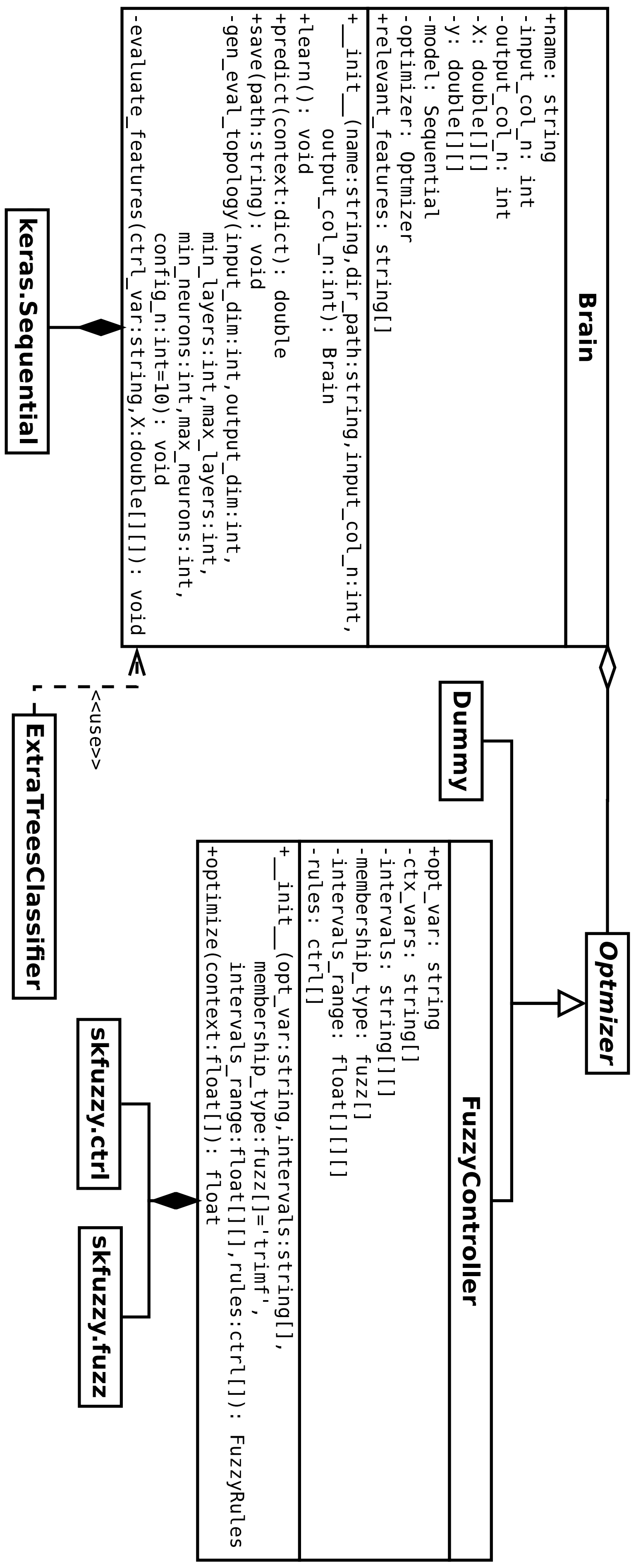

Brain
+name: string
-input_col_n: int
-output_col_n: int
-X: double[][]
-y: double[][]
-model: Sequential
-optimizer: Optmizer
+relevant_features: string[]
+ __init__(name:string,dir_path:string,input_col_n:int,
output_col_n:int): Brain
+learn(): void
+predict(context:dict): double
+save(path:string): void
-gen_eval_topology(input_dim:int,output_dim:int,
min_layers:int,max_layers:int,
min_neurons:int,max_neurons:int,
config_n:int=10): void
-evaluate_features(ctrl_var:string,X:double[][]): void
keras.Sequential
<<use>>
ExtraTreesClassifier
Dummy
Optmizer
FuzzyController
+opt_var: string
-ctx_vars: string[]
-intervals: string[][]
-membership_type: fuzz[]
-intervals_range: float[][][]
-rules: ctrl[]
+ __init__(opt_var:string,intervals:string[],
membership_type:fuzz[]='trimf',
intervals_range:float[][],rules:ctrl[]): FuzzyRules
+optimize(context:float[]): float
skfuzzy.ctrl
skfuzzy.fuzz

Figure 42 – UML Brain Class Diagram.

- `name`: ID of the device that the Brain object will learn to controll.
- `input_col_n`: Number of input collumns of the prior knowledge Dataset.
- `output_col_n`: Number of ouput collumns of the prior knowledge Dataset.
- `X`: Matrix with the inputs from the dataset.
- `y`: Matrix with the outputs from the dataset.
- `model`: Most recent ANN trained and compiled model. The model is implemented by Keras Sequential Class.
- `optimizer`: The optmizer object. Applies heuristics to optimize the device energy consumption.
- `relevant_features`: Holds a list of the relevant features from the dataset that were calculated by the ExtraTreesClassifier.

Methods

- `__init__(name, dir_path, input_col_n, output_col_n)`: Object initializer. Takes the ID of the device to assign to the Brain object, the directory path where the user's datasets are stored, the number of input and columns in the prior knowledge dataset. Additionally, the object initializer assigns the datapoints to respective matrices, creates the ANN model, if more data is already available, trains the network.
- `learn()`: Function triggered by adjustments made by the user when he does not agree with the predictions of the decision engine. It issues the whole retraining of the ANN. If there are enough new entries since the last feature evaluation (10% more) it reevaluates the features. The ANN topology is automatically evaluated and the best model is assigned to the `model` attribute.
- `predict(context)`: Takes the current context dictionary and predicts how the device should be controlled. If an optimizer exists for the device it also handles the optimization computation.
- `save(path)`: Saves the ANN weights and topology.
- `gen_eval_topology(input_dim, output_dim, min_layers, max_layers, min_neurons, max_neurons, config_n)`: Generates multiple topology configurations according to the parameters trains all and compare the scores of accuracy to select the best one. The chosen one is assigned to model.

- `evaluate_features(ctrl_var, X):` Issues an ExtraTreesClassifier to select the most relevant features in the dataset for the control of the device.

Optimizer objects may extend either Dummy or FuzzyController classes. The dummy is a blank class that does not optimize anything. The FuzzyControler class instantiates a fuzzy controller based on the parameters passed to the object initializer.
Heading
`from Optimizer import FuzzyController`
Attributes

- `opt_var:` Variable that will be optimized. Usually the ID assigned to `name` of the Brain object.
- `ctx_vars:` String with the IDs of context variables useful to the optimization.
- `intervals:` Matrix with the names of the intervals. Each line represents a fuzzy set's interval's names.
- `membership_type:` The membership function to be used in the fuzzy controller. It is set to `trimf` (Triangular Membership Function) by Default.
- `intervals_range:` Matrix with the points to be used to define the intervals
- `rules:` Set of fuzzy control rules based in power optimization heuristics.

Methods

- `__init__(opt_var, intervals, membership_type, intervals_range, rules):` Object initializer. Creates the fuzzy controller with information from the parameters.
- `optimize(context):` Computes the result of the fuzzy controller based on the current context.

The Brain class along with the Flask server compose the Prescient Class. Prescient contains the full implementation of the server. It can respond to the gateway as well as application REST requests and relay commands or messages from one to another. The prescient class is illustrated in the diagram of figure 43.
Attributes

- `server:` Flask object that instantiates the HTTP server and REST request handler.
- `cluster:` String of Brain objects.

Methods

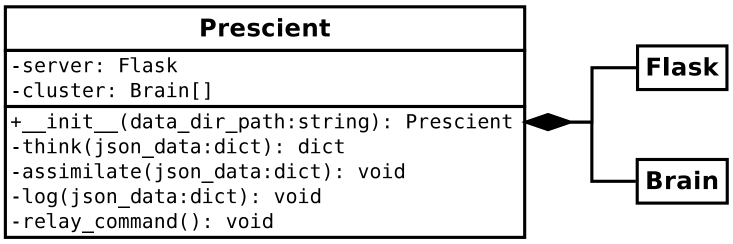

Prescient
-server: Flask
-cluster: Brain[]
+__init__(data_dir_path:string): Prescient
-think(json_data:dict): dict
-assimilate(json_data:dict): void
-log(json_data:dict): void
-relay_command(): void
Flask
Brain

Figure 43 – Prescient UML Class diagram.

- `__init__(data_dir_path)`: Object initializer. Takes a path to where the data is stored. From the data it creates all the Brain Objects to control the devices. In the data directory the server keeps a list of the controllable devices to instantiate the objects. Finally it ends by running the Flask HTTP server.
- `think(json_data)`: Receives a JSON object with current context and for every Brain object predicts how to control (and optimize) it. If during a Back-off does nothing. Returns a dictionary of key:value pairs with the predicted adjustments that is forwarded to the gateway.
- `assimilate(json_data)`: Receives the changes made manually by the user to the devices. Starts a back-off period and verifies which Brain objects are affected by this changes and when necessary creates multiple threads to re-train the ANNs.
- `log(json_data)`: Receives a JSON object from the gateway to be logged on the server.
- `relay_command()`: Bridges the commands to devices from the mobile application.

6 Results and Analysis

In this chapter, the whole system implementation is validated. Both the experiments and results for each component of the system are detailed and discussed. The results hereby presented are based on the developed system functioning during a period of approximately two weeks in LISHA's Smart Room. The system was configured to search for only one user's smartphone and control the light A0 status (On/Off) and power[1]. Both the gateway and the server use threads to run periodic, constant or on demand functions. The periods and timeout limits of the threaded functions are presented in Table 6. During this time a total of 307500 contextual-datasets were collected.

Gateway Threaded Function	**Period**	**Timeout Limit**
Gateway Request Handler	Constant	None
Data Parser	5 Seconds	4.8 Seconds
Logger Period	15 Seconds	None
User Finder	15 Seconds	None
Connectivity Checker	10 Seconds	5 Seconds
Control Prediction Period	20 Seconds	None
Server Threaded Function	**Period**	**Timeout Limit**
Server Request Handler	Constant	None
ANN Training	On Demand	None
ANN Topology Evaluation	On Demand	None

Table 3 – Threaded function default periods and timeout limits.

6.1 Data Parsing

The Smart Room implementation was supposed to gather data from devices and send them through Modbus messages over the IEEE 802.15.4 wireless protocol to a central network node. As stated in the previous chapter, the motes were already physically installed in the room lights and power outlets. The firmware running on these devices were updated to send data every second with the correct IDs expected by the gateway. The external sensing unit was implemented from scratch, and a new firmware was developed to handle the DHT11 sensor. All the data was sent to the Gateway mote that received the messages and forwarded them via Serial USB connection to the Galileo board where they were parsed. To verify if the messages were being correctly delivered to the gateway Linux minicom tool was used.

[1] During the development of the project, two of the three lights installed in LISHA's room presented malfunction due to electrical problems in the building and could not be further used.

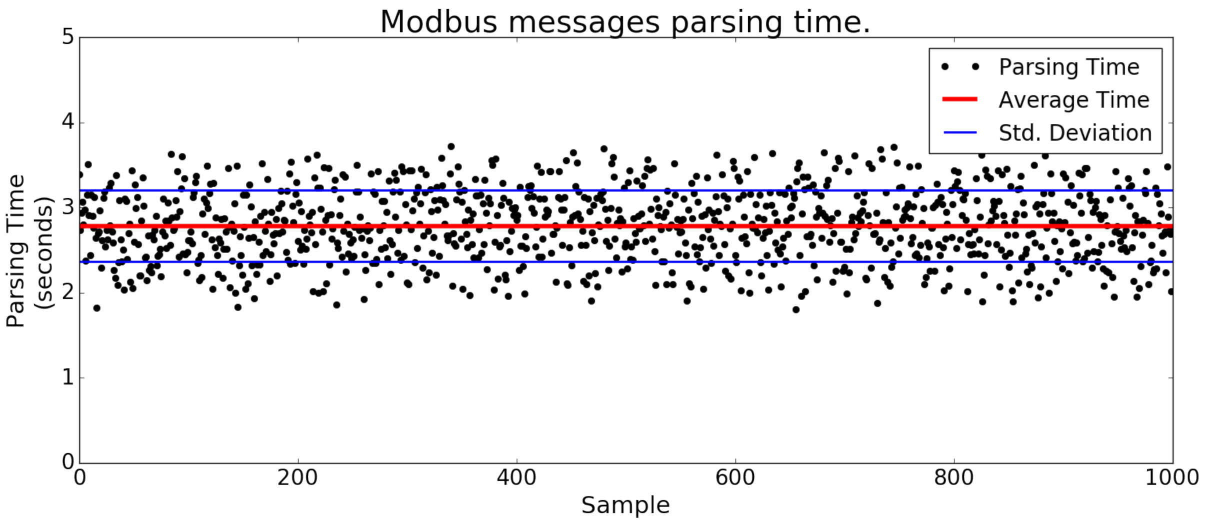


Figure 44 – Parsing time plot of 1000 data parses.

Total Samples	1000
Average Parse Time	2.781 Seconds
Parse Time Std. Deviation	0.416 Seconds
Parsing Maximum Time	3.718 Seconds
Parsing Minimum Time	1.800 Seconds

Table 4 – Wireless message parsing time statistics.

In our implementation, every mote sends one message per second. That accounts for roughly eight messages. The average message parsing and conversion time for messages received from the EPOSMote III were measured to verify if all the messages could be received in a period shorter than the threads. The Information is logged to the server every 15 seconds. The parsing thread period chosen was 5 seconds, to avoid receiving and logging invalid data (the latest valid dataset is committed to the server). A set of 1000 parsing time measurements were made to verify if it was possible to achieve under the 5 seconds period. The average receiving and parsing time of all the messages from the devices was calculated as 2.758 seconds (running multi-threaded along with the other threads on the daemon). A table with the complete statistics for the parsing time is shown in Table 4. The set of measurements and average parsing period are shown in Figure 44.

To verify the need for data filtering in the gateway, an additional counter was implemented in the parser to test the number of invalid measurements. During the period were parsed approximately 2.5 million messages. Of these individual measures, the parses considered 322 to be invalid measurements. That represents approximately 0.013% error rate. Later, it was discovered that the DHT11 sensor used in the external sensing unit presented a defect and sometimes gave an incorrect measurement. Nevertheless, the error in the sensor justifies the data filters implemented by the parses. If an erroneous data point is incorporated into the training of the ANNs, it may corrupt the entire decision engine.

6.2 User Detection

The user detection thread is supposed to scan the network for the user's smartphone MAC ID in the format "XX:XX:XX:XX:XX" where every X represents a hexadecimal symbol. To enable this to be done, the Galileo board was connected to the wireless router, which provided it with an IP address in the same subnet as the smartphone and internet connectivity.

The user detection implementation approaches the problem in two different ways, when the user has not connected to the network, and when the user has already connected to an IP in the network. If the user has never connected to the network, the DHCP server has not yet leased any IP address to the smartphone's MAC, which means that it will lease a new one. To discover which IP was leased to the smartphone, the Nmap tool is used. A set of 150 measures of the Nmap network scan were made. The measurements were made with all the threads of the gateway running. The plot of the measurements is shown in Figure 45 and the statistical values for the measurements in Table 5.

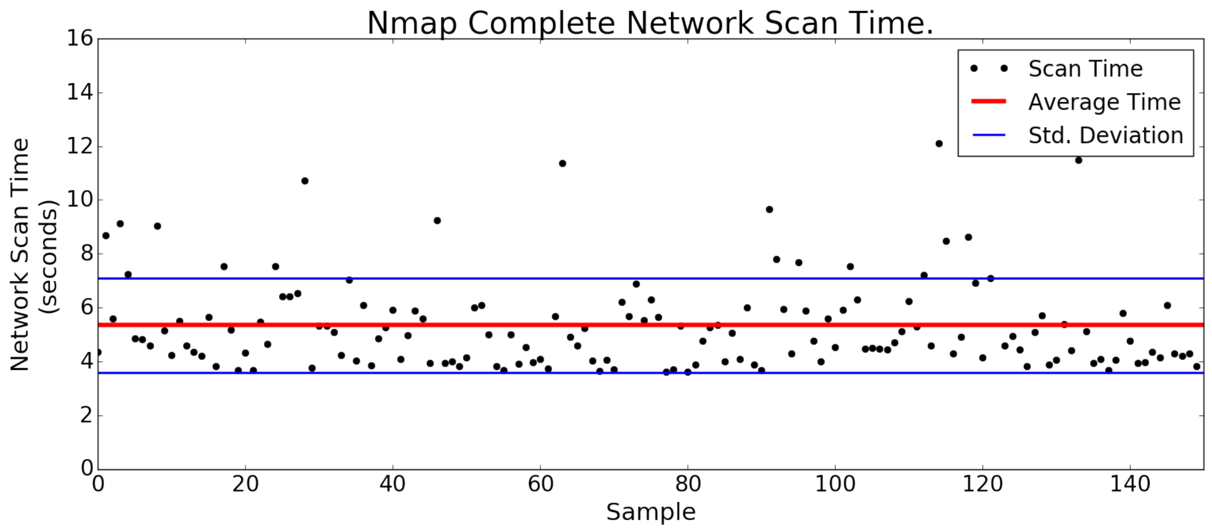


Figure 45 – Nmap Scan time plot of 150 network scans.

Nmap is not, however, the best way to verify if the user is connected if it already had an IP in the network associated with its MAC address. When the user already has an IP address leased to it's device, checking that IP with pings before issuing an Nmap scan is very useful. Pings to an IP are much faster than network scans. To analyze and confirm this hypothesis, the same set of measurements were made with ping configured

Total Samples	150
Average Nmap Scan Time	5.343 Seconds
Nmap Scan Time Std. Deviation	1.749 Seconds
Nmap Scan Maximum Time	12.118 Seconds
Nmap Scan Minimum Time	3.602 Seconds

Table 5 – Network Scan with Nmap time statistics.

to make three echo request tries. Figure 46 presents a scatter plot of the measurements obtained and Table 6 presents the statistics.

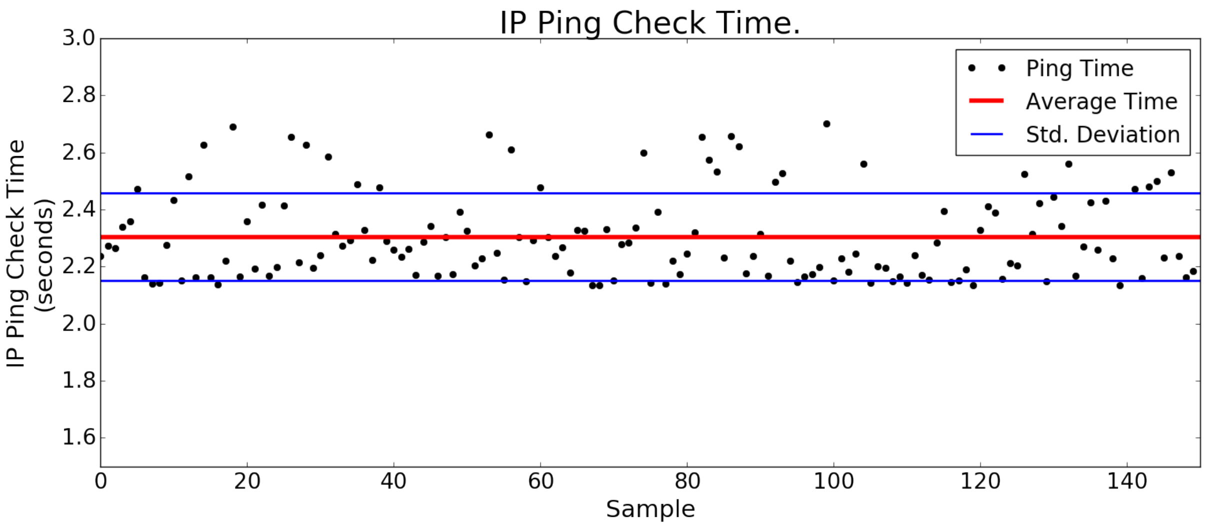


Figure 46 – IP ping check plot of 150 pings of 3 echo request tries.

Total Samples	150
Average Ping Check Time	2.304 Seconds
Ping Check Time Std. Deviation	0.154 Seconds
Ping Check Maximum Time	2.702 Seconds
Ping Check Minimum Time	2.133 Seconds

Table 6 – IP ping check time of 3 echo request tries statistics.

From the data presented it is simple to note that pings are much faster than Nmap scans when trying to verify if a known user is present in the network. Nmap, though very useful when trying to discover to which IP address a MAC address is connected, is too sophisticated and requires too many resources from a constrained embedded platform such as Galileo. It takes up to 12 seconds to make one verification if the MAC is connected. Especially when considering that a smartphone may be in power-saving mode, where it disables the connection for periods of time, the Nmap scans may take too long to detect it. The usage of such tool, however, is necessary when a MAC address has never been connected to the network, or there is doubt to whether it may have switched IP addresses.

Pings, on the other hand, have the advantage of well-behaved times with small standard deviation. A ping worst-case scenario is still faster than Nmap's best scan time. Because of that, it is possible to check more frequently for the connection without any significant impacts on the gateway performance. Provided that arp table lookups are issued to confirm the MAC addresses when needed. Internet connectivity checking is achieved in the same manner, though pings may take longer to reach remote servers and, therefore, perform worse than inside a subnetwork.

The association of both scanning techniques proved to be very successful in detecting the user's smartphone, which indicates that smartphones may be used as IDs for the context-aware decision engine profiles.

6.3 Adaptive Context-Aware Decision Engine

An ideal context-aware decision engine for controlling IoT devices should be sensitive and responsive to the presence of people and all information that surrounds. This decision engine makes devices work in concert to support people in carrying out their everyday life activities, tasks and rituals in an easy, natural way using information and intelligence that is hidden in the environment's context to automate them, emphasizing user experience. These systems must be able to accomplish some key aspects to make use of information and yield practical actions to the user.

The core components of the system described in this work are evaluated according to their metrics to verify the applicability of this approach to solving the context-aware control problem of smart environments.

6.3.1 Topology Evaluation

The topology of a neural network refers to the way the Neurons are connected, and it is a major factor in the system's functioning and learning. The most common topology in supervised learning is the fully connected network, also known as a dense MLP network. All input values to the network are connected to all neurons in the hidden layer (hidden because they are not visible in the input or the output). The outputs of the hidden neurons are connected to all neurons in the output layer, and the activations of the output neurons constitute the output of the whole network. Such networks are popular partly because theoretically they are known to be universal function approximators (with, e.g., a sigmoid or Gaussian nonlinearity in the hidden layer neurons) [26].

Nevertheless, the choice of an optimal topology brings better predictions to the system. The developed topology evaluation method can vary a series of settings of the neural networks generating as many models as needed to compare and chose the one with the best accuracy. In the case of two models providing the same precision, it selects the model with lesser complexity for two reasons: processing the calculations of a less complex model is faster and due to it be less likely to overfit the data.

The main problem of evaluating such topologies is the time required to process these evaluations.

To verify this question, a series of assessments, generating 20 different models, were run. These generated topologies were equal in the activation function (sigmoid), input and output dimensions, optimizer (adam) and loss functions (mean squared error). The same configuration used during the experiments. The dataset was based on the one obtained at the end of the two-week experimental period but extrapolated to show how the number of training examples and topology configuration may affect the evaluation time of the best topology. All the measurements were made with python time module, and ran on an Intel i7-4700MQ CPU @ 2.40 GHz x 8 with 12GB DDR3 @ 1600 MHz memory on Ubuntu 16.04. Figure 47 shows the ANN topology evaluation time for different configurations of hidden layer number and hidden neurons number according to the number of lines in the training set.

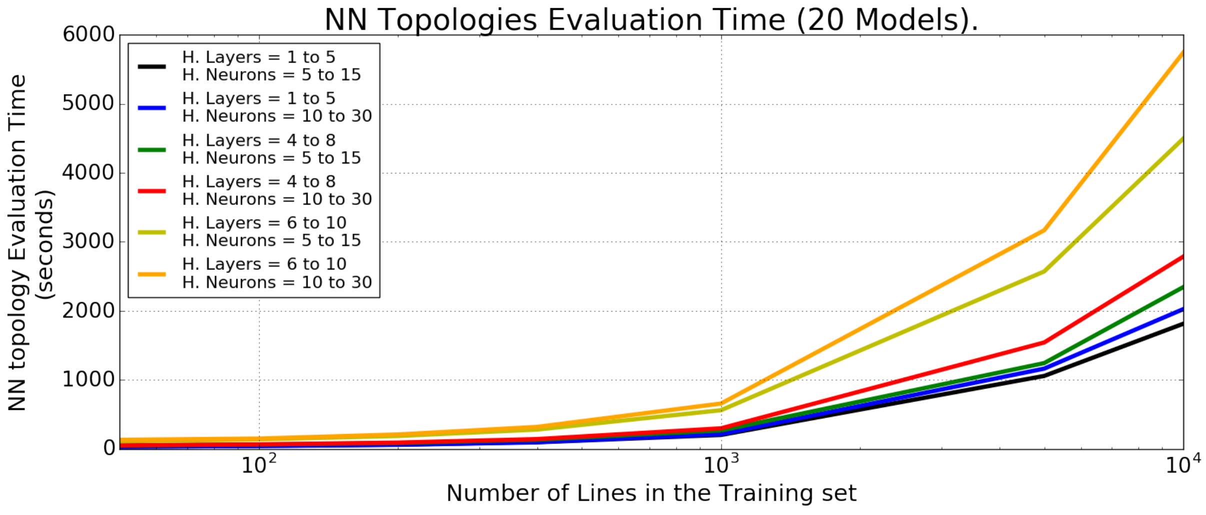


Figure 47 – Increase of ANN traning time according to topology complexity and size of the training set.

In this project, due to the system design that divides the training data into a separate dataset (UCD), it was not possible to reach a number of data points for the ANN topology evaluation to impact the decision engine or cause any complications. This point, however, should be considered when designing a system that may come to deal with too complex ANN topologies. If this gets to a stage, where the topologies become too complicated to be processed in standard CPUs, and dedicated GPUs would need to be instantiated as the backend of the ANN abstraction modules to provide high-performance processing. The use of higher processing power would only remediate the problem, as a greater number of data entries and ANN complexity would soon reach the GPU's maximum processing power.

The way the system proposed by this work was modeled, a good solution would be to use only the latest K entries of the user command dataset. Since correct predictions are re-included in the UCD, current behaviors are preserved in the most recent data entries. Another good impact of using only the latest points is that old habits and behaviors learned by the network that no longer occurred can be forgotten. Moreover, new patterns are more easily absorbed by the neural network since they are more representative to the K entries than to the whole dataset.

6.3.2 Incorporation of Prior Knowledge

In the system proposed by this work, the main objective is for the system to learn how to control the light. The previous knowledge considered in this case was simply the rule: If a user is present in the room; Then the light should be on. PDK is introduced in the same format that the data is stored in the other datasets. Every column is representing an environment variable and every row an observation at a particular time. The main reason to incorporate prior knowledge in the ANN is to increase ANN prediction accuracy, by stating beforehand relevant features to control a device and providing enough data to avoid erroneous predictions due to lack of representative examples (cold start).

Prior knowledge incorporation to the decision engine may be very useful. Many times there is a lack of proper (experimental) training data, and it is compensated for by the introduction of prior knowledge about the process. Not only that, it augments the training data and brings better support to the experimental data. It can be obtained either from other well-known and validated processes or may be defined by the users according to their preferences.

6.3.3 Relevant Feature Selection

In a future where close to 30% of billions of intelligent devices will be installed inside rooms, homes and buildings, programming devices with rules that consider all data collected from the environment is a cumbersome task. The evaluation of the relevant feature selection was done after the 2-week period where the system had already learned how to control the light. This was done to verify the dataset's impact on the feature selection. Firstly we analyze only the prior knowledge. Secondly, the effect of the inclusion of the Room Data (RD) (where there is no profiled[2] user in the room) on the feature importance. Lastly, the User Room Data (URD), to check, which of the context variables were particular to the user.

[2] Even though only one user was profiled, more people did use the room.

Figure 48 presents the three observations of feature relevance. The sum of all relevances accounts to 1. Features are considered relevant and used by the ANN if they achieve a minimum of 10% of relevance threshold. For the control of the light, the internal luminosity and light power consumption were marked as irrelevant because they are causes of the light control and not reasons to control it.

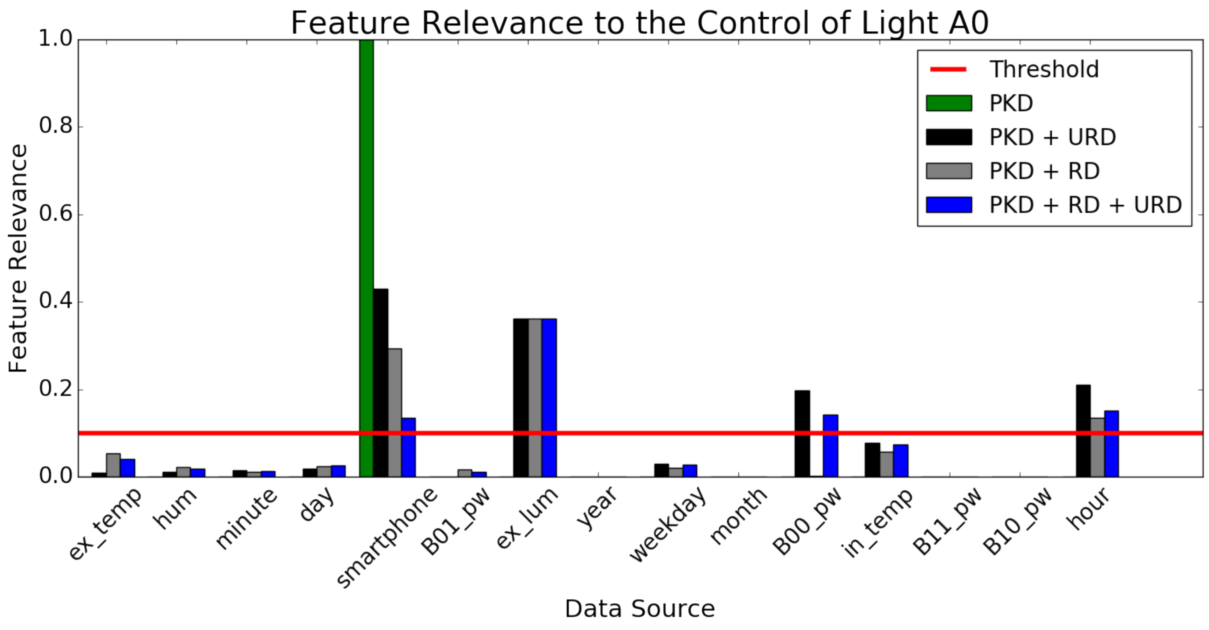


Figure 48 – Feature relevances evolution with PKD, PKD + URD, PKD + RD and PKD + RD + URD.

As expected, for the previous knowledge data, the algorithm detected only the smartphone as relevant for the control of the light. In the ANN terms, it meant that the neural network would be trained only with the smartphone feature of all datasets.

When user room data was appended to the prior knowledge data, it is possible to note a shift in the data source importances. From this change, we can infer that the environment variables that matter the most are the external luminosity (e.g. day or night) and the time of day. It makes a lot of sense that both are relevant since they are closely related. The higher relevance still for the luminosity is justified by the fact that even during the day, there may be cloudy days where the luminosity is low, and there is need to turn on the lights. The presence of the B00 power consumption in the relevant features may seem strange at first sight. However, during the period of data collection and experiments, a projector was connected to the B00 power outlet, and whenever it was turned on, the lights were turned off by the user.

The association of prior knowledge with room data proved that for other users also the external luminosity and time of day are closely connected. Lastly, the association of all datasets exemplifies the generalization for the room control.

The four examples show four possible scenarios of application for the usage of feature detection before the neural network training. Respectively, when someone might want to limit or program rules, when a single profiled user uses room (e.g. home office), when no profiled user uses the room (e.g. building hall or shared space) and, lastly, when there is a mix of profiled and non-profiled users that have access to the environment. This last scenario represents the one utilized in this work.

If the datasets become too sparse with time and the algorithm is no longer able to find consistent correlations between data sources, a similar approach to the one described in the model evaluation could be adopted, where a set of the latest K data entries are selected, to verify the most recent correlations.

6.3.4 Device Control Prediction

The gateway was configured to only query for predictions when a profiled user was detected in the room. Even though more people used the room, it would be hard for the author of this work to interact with the room to correct the controls predicted if required. Figure 49 shows two side by side snapshots of the ANN outputs. The first one represents the control prediction for light A0 for the ANN trained only with PKD. The PKD trained the ANN to turn the lights on according to the presence of the user inside the room. At this point, the only relevant feature of the data for the control of the light was the smartphone being connected to the network. This behavior is shown in the left plot. No matter the power consumption measured in the B00 outlet, the lights are turned on if the user is present in the network and off if it is not.

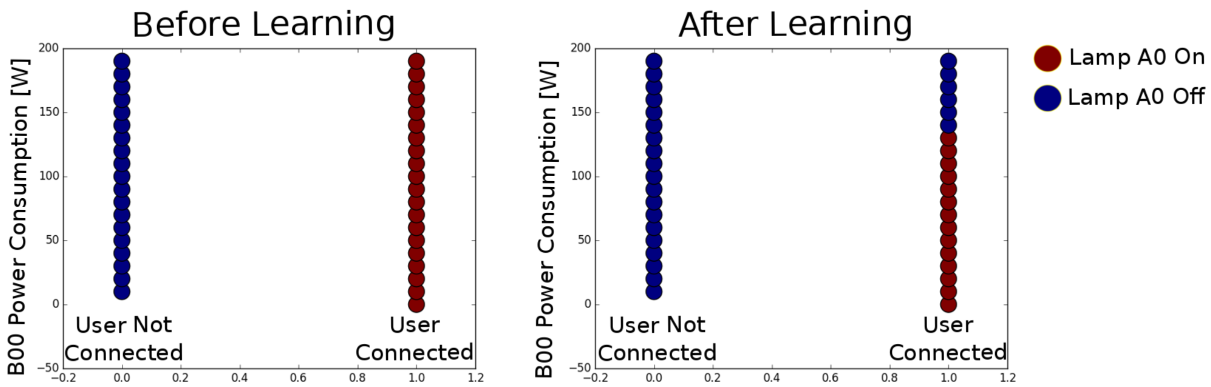


Figure 49 – Device control prediction before and after the inclusion of B00 power consumption in relevant features.

As shown in the Feature Selection, along the 2-week period, the power consumption(B00_pw), hour and external luminosity were also selected as relevant features for the light control. The ANN was continuously trained when a prediction was corrected by the user. The snapshot of the right of Figure 49, represents the control prediction of the

ANN after two weeks of training. Here is possible to note that with the inclusion of the B00_pw in the list of relevant features for the control of light along with the adjustments made by the user stored in the UCD the ANN learned that whenever there is a surge in the values of B00_pw (Meaning that the projector was turned on and the power consumption is rising) the light A0 should be turned off.

Figure 50 shows the instant execution timeline where the projector is turned on. As the projector heats energy consumption rises. The first moment when the gateway queries a prediction and the power consumption of the projector, the ANN predicts that the light should be turned off (indicated by the blue vertical line). When the projector is turned off, a little while later, the power consumption drops radically about 30W (the fan keeps on a little while longer to cool the projector). After that, at the first moment, a prediction is queried with the new context (indicated by the vertical red line), the ANN output is to turn on the light.

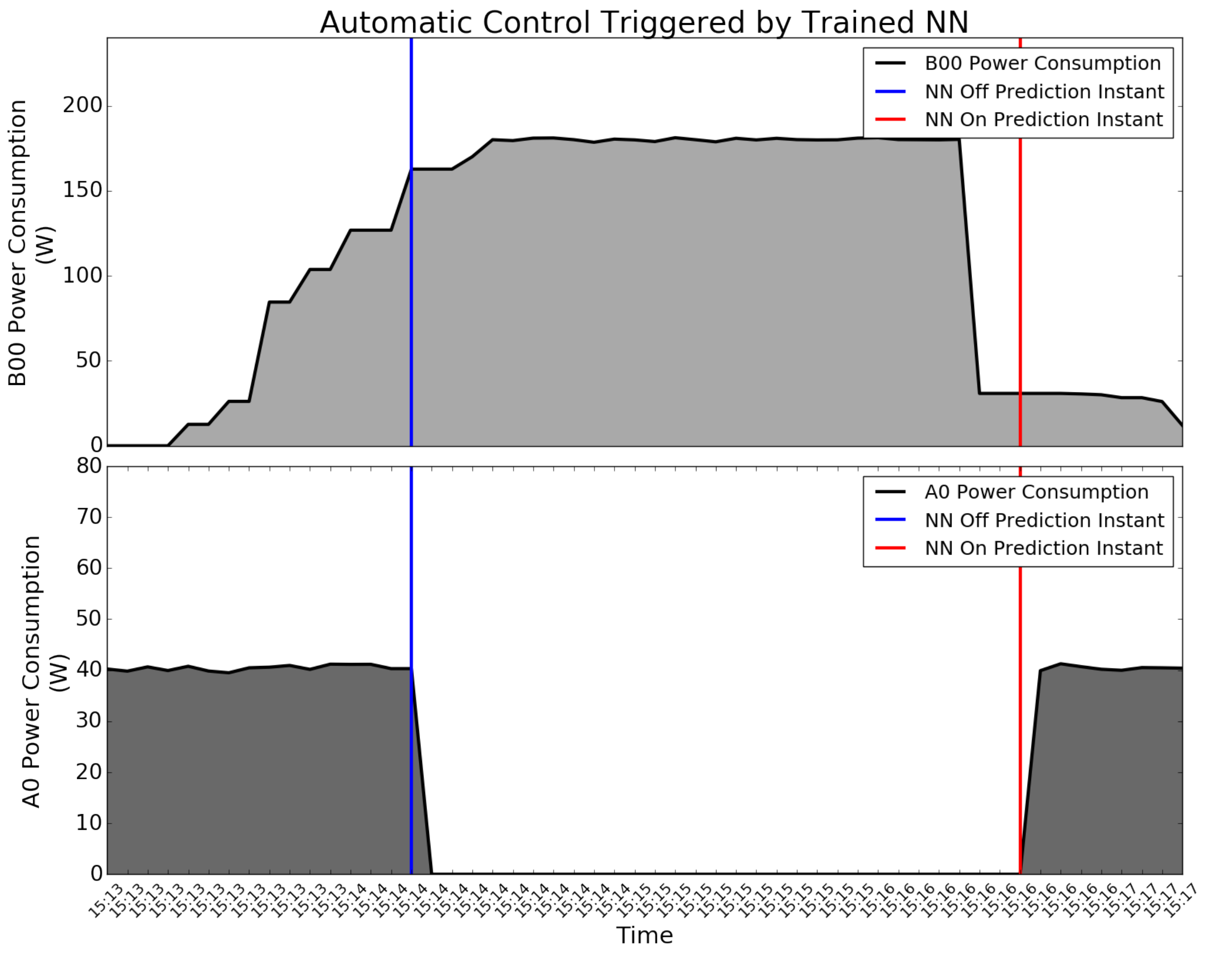


Figure 50 – Instants in execution where the changes in context of power consumption imply in the ANN predicting to turn light Off and On.

As predictions are only queried when the user is connected to the network, it was expected that Hour and external luminosity variables would have no impact on the predictions from the ANN. Further investigations could be done to check how the system would perform making predictions constantly, even when no profiled user is present in the room.

6.4 Fuzzy Control Energy Footprint Optmizer

In Figures 51, 52 and 53 it is possible to see the predicted output for the light dimmer in a context where internal luminosity measure is at about 25% and external luminosity is at about 45%. This exact scenario, from when the figures were made, represents a sunny day with the light inside the room turned off. The calculation resulted in the dimmer value to which the light should be adjusted when turned on.

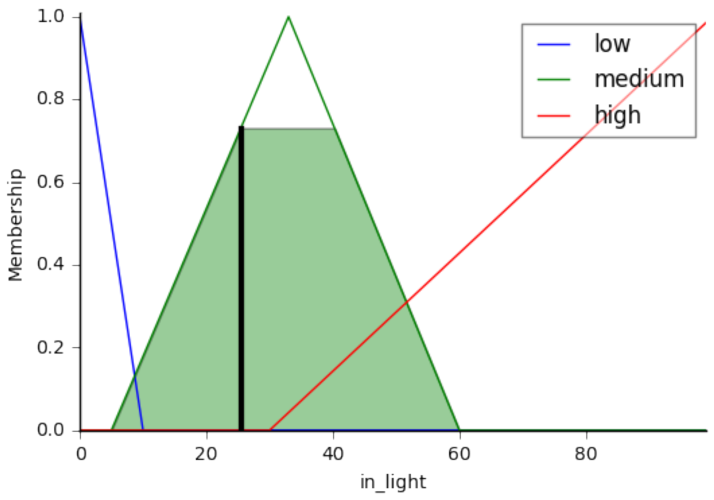


Figure 51 – Fuzzy results for internal luminosity of 25.45%.

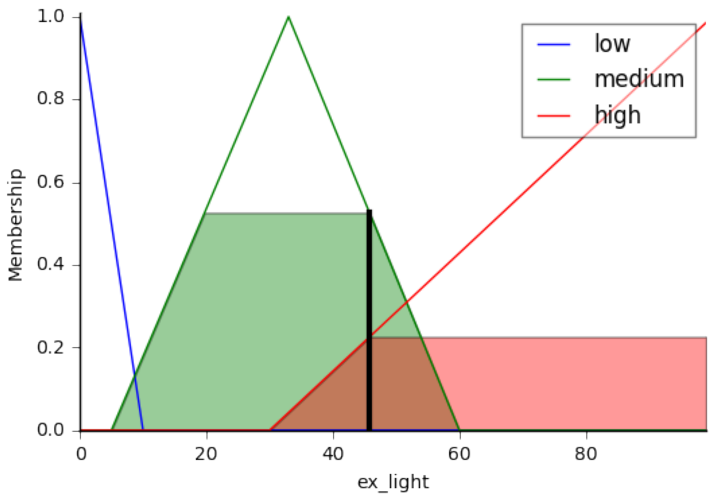


Figure 52 – Fuzzy results for external luminosity of 45.8%.

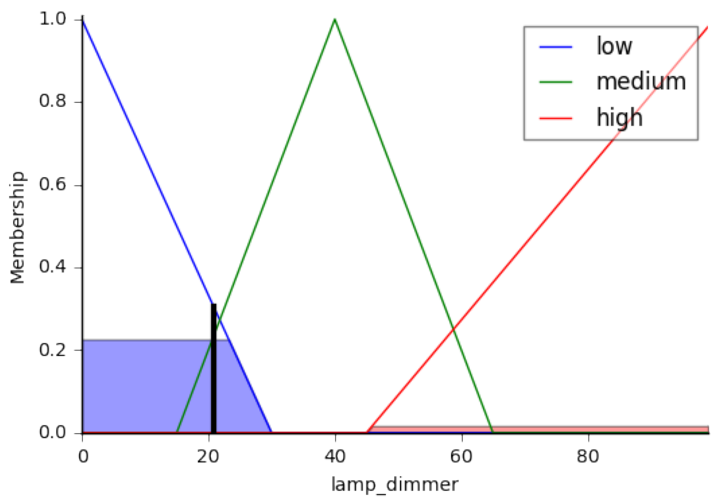


Figure 53 – Lamp dimerization output calculated by the fuzzy controller.

The fuzzy controller based on external and internal luminosity has proven to be an excellent approach to reducing the energy consumption footprint of the light. When in power saver mode, the decision engine makes the computations of the fuzzy controller and applies it to the result of the ANN. Figure 54 shows a snapshot of the light A0's energy consumption time series where the power saving mode is turned on (indicated by the red vertical line). This particular adjustment had almost no impact on illumination comfort and provided a reduction in the energy consumption of approximately 35%.

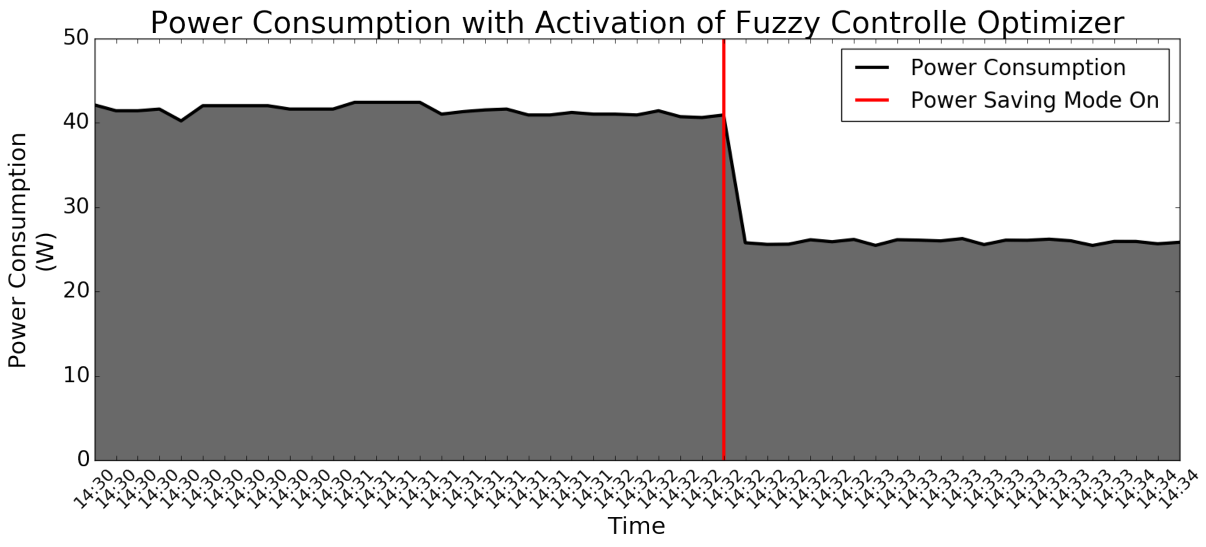


Figure 54 – Device Control Prediction after the Inclusion of B00 Power Consumption in Relevant Features.

Even though in this project only a fuzzy controller was implemented to be used as an optimizer, the system design is capable of implementing a series of techniques as optimizers.

System Component	Requirement	Attended
Intelligent IoT Gateway	User detection	✓
	Bridge different communication protocols	✓
	Parse and filter sensor data	✓
	Communicate with the internet	✓
	Periodically logging of environment data	✓
	Query context-aware algorithm for environment control	✓
	QoS ensurance	✓
Hybrid application	Cross-platform availability	✓
	Remote device control capabilities	✓
	Real-time data visualization	✓
	Data history visualization	✓
Context-aware decision engine	Automatic identification of relevant information	✓
	Inclusion of prior knowledge	✓
	Continuous user preference learning	✓
	User-specific preference prediction	✓
	Heuristic-based energy footprint optimization	✓
System integration	Successful communication between disparate modules	✓
	Resuming operation after shutting down	✓
	Maintain functioning without significant overhead	✓
	Ensure non-existence of dead-locks	✓

Table 7 – System attended requirements

This Section along with the description of the system development in Section 5 show that the requirements of the project were fully attended by the system proposed, according to each one's respective metrics, and implemented in this work. Table 7 lists each component's requirements and whether they were attended or not.

7 Conclusion and Future Work

In this work the implementation of an Adaptive Context-Aware Monitoring and Control system is described. The system comprises three main parts: *i)* Intelligent Gateway for the Internet of Things based on Intel Galileo Gen. 2 and EPOSMoteIII, responsible for providing intelligence to leaf nodes and bridging smart-devices (Modbus over IEEE 802.15.4) to the internet (TCP/IP over Ethernet); *ii)* Hybrid application that allows user remote control and monitoring of the room appliances and devices; *iii)* Adaptive Context-Aware decision engine that is able to select the best adjustments of the room according to user preferences.

The use of an intelligent embedded control IoT gateway can reduce the complexity and cost of end points, while providing the user with better services. In order for that to happen the gateway must be able to take on high-level management tasks such as sensor data filtering, enabling connectivity and user profiling though MAC address detection. The implementation of the gateway based on the Galileo Gen. 2 board was able to successfully deal with these issues. The user detection with the use of Nmap was slower than expected in some cases, but the use of pings along with the ARP tables of the system could circumvent this issue. Data parsing performance was sufficient and the error rate of the received data was very low, nevertheless, redundancy of measurements was implemented to minimize the chances of not having new data to send to the server.

An application for remote control and monitoring this sort of smart spaces is very useful. They provide the user with basic capacity for configuring the devices as he wishes, as well as providing feedback for the decisions. The developed app was able to successfully control and monitor the room, both from the smartphone and website. The use of a framework for developing hybrid applications allowed the implementation of a portable app with cross-platform availability. The choice of communication technologies for the exchange of messages with the server also has proven to be right.

Contextual-data analysis is very important and can provide the system with a vast range of new applications or functionalities. Between these functionalities are predictive analysis and automatic environment device control. Deep Neural Networks can learn to interpret complex real-world sensor data, and are among the most effective learning methods currently known for that purpose. ANNs may encounter multiple drawbacks when dealing with lack of data (cold-start) or highly-dimensional data. These drawbacks, however, can be avoided. The composition of data mining and machine learning techniques, such as the use of an extra-trees classifier to select relevant features of data that should be used by the ANNs, proposed by this project was very successful to reduce the data dimensionality without loss of relevant data for the training of the ANNs. The incorporation of prior knowledge by the ANN avoided the cold-start issue and improved the success of the relevant feature selection. The use of feedback concepts from reinforcement learning provided the context-aware engine with a continuous learning capability, which means that new user behaviors will be learned by the system.

Contextual environment data demonstrated to be very useful for the optimization of energy consumption of devices. Users are usually worried about the energy consumption of their appliances, but very seldom, worried enough to keep adjusting the settings of the appliances many times during the day. Fuzzy logic and Fuzzy control were able to provide a power-consumption reduction while maintaining an acceptable level of user comfort based in simple power-saving heuristics.

Future works are going to focus of the inclusion of new devices and new sources of information such as user's activity, location and condition, and new controlable devices for further validation of the developed context-aware algorithm, as well as evaluate and test yet other machine learning and data mining techniques for the automatic control of smart environments.

A demonstration video of the system working in LISHA's smart room is available at: <https://www.youtube.com/watch?v=G7WMr-fjB-4>

Bibliography

1 Cisco. *The Internet of Things - How the Next Evolution of the Internet Is Changing Everything*. "http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf". Accessed: October 2016. Cited on page 21.

2 Business Insider. *IoT Growth*. "http://www.businessinsider.com/how-the-internet-of-things-market-will-grow-2014-10". Accessed: July 2016. Cited on page 21.

3 FORBES. *A Simple Explanation Of 'The Internet Of Things'*. "http://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand". Accessed: December 2016. Cited on page 23.

4 COOK, D.; DAS, S. *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. [S.l.]: Wiley-Interscience, 2004. ISBN 0471544485. Cited on page 23.

5 POSLAD, S. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. 1st. ed. [S.l.]: Wiley Publishing, 2009. ISBN 0470035609, 9780470035603. Cited on page 23.

6 SCHILIT, B.; ADAMS, N.; WANT, R. Context-Aware Computing Applications. *1994 First Workshop on Mobile Computing Systems and Applications*, p. 85–90, 1994. Cited on page 24.

7 PALACE, B. *Data Mining*. 1996. "http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologie". Accessed: December 2016. Cited on page 24.

8 BREIMAN, L. Random forests. *Machine Learning*, v. 45, n. 1, p. 5–32, 2001. ISSN 1573-0565. Cited on page 25.

9 GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. *Machine Learning*, v. 63, n. 1, p. 3–42, 2006. ISSN 1573-0565. Cited on page 25.

10 MITCHELL, T. M. *Machine Learning*. [S.l.]: McGraw-Hill, 1997. 417–433 p. ISSN 87567016. ISBN 0070428077. Cited on page 25.

11 MISHRA, B.; DEHURI, S.; KIM, E.; WANG, G. *Techniques and Environments for Big Data Analysis*. [S.l.]: Springer International Publishing, 2016. 63-64 p. ISSN 9783319275208. ISBN 3319275208. Cited on page 26.

12 PEDRYCZ, W. *Fuzzy Control and Fuzzy Systems (2Nd, Extended Ed.)*. Taunton, UK, UK: Research Studies Press Ltd., 1993. ISBN 0-86380-131-5. Cited on page 26.

13 Software/Hardware Integration Lab. *EPOSMoteIII*. "http://epos.lisha.ufsc.br/EPOSMote+III". Accessed: October 2016. Cited on page 27.

14 Software/Hardware Integration Lab. *Application-Driven Embedded System Design*. "http://lisha.ufsc.br/ADESD". Accessed: October 2016. Cited on page 29.

15 Flanagan, D. *JavaScript: The Definitive Guide, 6th Edition*. [S.l.]: O'Reilly Media, 2011. 63-64 p. ISSN 9783319275208. ISBN 978-0596805524. Cited on page 31.

16 DUCKETT, J. *JavaScript and JQuery: Interactive Front-End Web Development*. 1st. ed. [S.l.]: Wiley Publishing, 2014. ISBN 1118531647, 9781118531648. Cited on page 31.

17 RAZZAQUE, M. A. et al. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, v. 3, n. 1, p. 70–95, Feb 2016. ISSN 2327-4662. Cited 2 times on page(s) 35 and 39.

18 LEE, S. H.; CHUNG, T. C. System architecture for context-aware home application. *Proceedings - Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, p. 149–153, 2004. Cited 2 times on page(s) 35 and 36.

19 DEY, A. K. Understanding and Using Context. *Personal and Ubiquitous Computing 5 (1)*, p. 4–7, 2001. ISSN 1617-4909. Cited on page 36.

20 MOON, J. W.; LEE, J. H.; KIM, S. Evaluation of artificial neural network-based temperature control for optimum operation of building envelopes. *Energies*, v. 7, n. 11, p. 7245–7265, 2014. ISSN 19961073. Cited on page 37.

21 ZHENG, H.; WANG, H.; BLACK, N. Human Activity Detection in Smart Home Environment with Self- Adaptive Neural Networks. *Networking, Sensing and Control*, p. 1505–1510, 2008. Cited on page 37.

22 WANGLEI; SHAO, P. Intelligent control in smart home based on adaptive neuro fuzzy inference system. In: *Chinese Automation Congress (CAC)*. [S.l.]: IEEE, 2015. p. 1154–1158. ISBN 9781467371896. Cited on page 37.

23 ZHANG, L.; LEUNG, H.; CHAN, K. Information fusion based smart home control system and its application. *Journal of IEEE Transactions on Consumer Electronics*, v. 54, n. 3, p. 1157–1165, 2008. ISSN 0098-3063. Cited on page 38.

24 YE, J. et al. The research of an adaptive smart home system. *ICCSE 2012 - Proceedings of 2012 7th International Conference on Computer Science and Education*, n. Iccse, p. 882–887, 2012. Cited on page 38.

25 Software/Hardware Integration Lab. *Smart Building*. "http://smartbuilding.lisha.ufsc.br/tiki-index.php". Accessed: October 2016. Cited on page 46.

26 MIIKKULAINEN, R. *Topology of a Neural Network*. [S.l.]: Springer US, 2010. 988-989 p. ISBN 978-0-387-30768-8. Cited on page 87.