CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

BEng Thesis

Creation of a pilot for the FlexOffer concept

Joss Santos

CISTER-TR-161205

2016/11/15

Creation of a pilot for the FlexOffer concept

Joss Santos

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: 1120527@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

The revolution in the energy market and the end user need to control the things he has around him lead to the creation of a new concept: flexoffers. The flexoffer is built around scheduling energy usage with the prices of the energy market. The flexoffer is comprised of the pattern of usage and a window of when the pattern can begin. Those parameters are exposed to the energy markets using an Aggregator, entity responsible for gathering all the flexoffers in a region. The market will reply to the flexoffer proposition with the best time for the consumption of that energy, which in this case is when the prices are the lowest.

This project revolves around the implementation of a pilot capable of applying that concept to a real life situation. Thus FlexHousing was created. It allows for the control of the energy usage of the appliances in a home or building. Such is possible by the integration of external technologies. Virtual Power Solutions, a national energy solutions company developed a product called Cloogy. The Cloogy is a gateway installed in one 19s house and is capable of communicating with smart device provided by VPS. In this case, smart plugs were deployed. These smart plugs are placed between the appliance and the electrical outlet. From there the plug is able to control the flow of electricity towards the appliance. Sensors are also installed in these plugs, capable of collecting data regarding the energy usage amongst other metrics. In order to operate the plugs, VPS also provided us with an external API, capable of receiving request and forwarding those request back to the plugs.

FlexHousing is comprised of 2 modules: the FlexHousing middleware and the FlexHousing FrontEnd. The middleware acts as a hub for every operation in the FlexHousing environment. It bridges the various system in play, the flexoffer services and the VPS services. FlexHousing allows for the creation of rooms and the attachment of devices equipped with VPS technology. The user is then able to perform a variety of operations, such as, applying flexoffers, remotely controlling the appliances and collect data from the sensors in the VPS devices. For the interaction with the user, FlexHousing FrontEnd was developed. It provides a user-friendly graphical interface for the user to setup and control FlexHousing middleware. The interface revolves around the rooms and the devices. The user is able to check the flexoffers he has applied and the schedules the flexoffers services sent back. If the user wants, he can deploy the VPS smart plugs to non flexoffer complaint devices, such as lamps, television and computer. Using the FrontEnd, he is also capable of using them.

PESTI- TECHNICAL REPORT

CREATION OF A PILOT FOR THE FLEXOFFER CONCEPT

Student: Joss Santos - 1120527

Organization: CISTER research unit

Project Managers: Luis Lino Ferreira & Michele Albano

Project Supervisor: Constantino Martins

CISTER - Research Center in
Real-Time & Embedded Computing Systems

isep Instituto Superior de Engenharia do Porto

INESCTEC

Creation of a pilot for the flexoffer concept

LEI-DEI

2015/2016

1120527 – Joss Dos Santos

Degree in Informatics Engineering

October 2016

ISEP Supervisors- Constantino Martins

External Supervisor- Michele Albano & Luis Lino Ferreira

“Strength and growth come only through effort and struggle”

Napoleon Hill

I want to thank all the key participants involved in the creation of projects. Professor Michele Albano and Professor Luis Ferreira were an unimaginable help, always present for any kind of discussion. Bruno Silva was a valuable colleague, helping me throughout this internship. I also want to thank Paulo and Renato for having dealt with me all of these months.

I want to thank CISTER, for making this experience possible, as well as the DEI department for giving me the tools and education that I need in order to pursue my dreams.

To all of my family that had to put up with all the stress and complications that were met along this project, I say sorry for being such a hassle but also thank for accepting me as such.

My friends couldn't be left out of this, they helped me built the road I took to achieve my dreams. They are too many to enumerate them all, but I'm sure they know who they are.

A special thanks to Jordann and Ana. Without your support, I wouldn't have been able to get where I am.

Joss Santos

The revolution in the energy market and the end user need to control the things he has around him lead to the creation of a new concept: flexoffers. The flexoffer is built around scheduling energy usage with the prices of the energy market. The flexoffer is comprised of the pattern of usage and a window of when the pattern can begin. Those parameters are exposed to the energy markets using an Aggregator, entity responsible for gathering all the flexoffers in a region. The market will reply to the flexoffer proposition with the best time for the consumption of that energy, which in this case is when the prices are the lowest.

This project revolves around the implementation of a pilot capable of applying that concept to a real life situation. Thus FlexHousing was created. It allows for the control of the energy usage of the appliances in a home or building. Such is possible by the integration of external technologies. Virtual Power Solutions, a national energy solutions company developed a product called Cloogy. The Cloogy is a gateway installed in one's house and is capable of communicating with smart device provided by VPS. In this case, smart plugs were deployed. These smart plugs are placed between the appliance and the electrical outlet. From there the plug is able to control the flow of electricity towards the appliance. Sensors are also installed in these plugs, capable of collecting data regarding the energy usage amongst other metrics. In order to operate the plugs, VPS also provided us with an external API, capable of receiving request and forwarding those request back to the plugs.

FlexHousing is comprised of 2 modules: the FlexHousing middleware and the FlexHousing FrontEnd. The middleware acts as a hub for every operation in the FlexHousing environment. It bridges the various system in play, the flexoffer services and the VPS services. FlexHousing allows for the creation of rooms and the attachment of devices equipped with VPS technology. The user is then able to perform a variety of operations, such as, applying flexoffers, remotely controlling the appliances and collect data from the sensors in the VPS devices. For the interaction with the user, FlexHousing FrontEnd was developed. It provides a user-friendly graphical interface for the user to setup and control FlexHousing middleware. The interface revolves around the rooms and the devices. The user is able to check the flexoffers he has applied and the schedules the flexoffers services sent back. If the user wants, he can deploy the VPS smart plugs to non flexoffer complaint devices, such as lamps, television and computer. Using the FrontEnd, he is also capable of using them.

Keywords (Theme): Internet of Things, Flexoffer, Energy automation.

Keywords (Technologies): Java, Derby Apache, REST, C#, Service-oriented Architecture.

A revolução dos mercados energéticos e a necessidade do consumidor final em controlar as coisas que estão a sua volta levou à criação de um novo conceito: a flexoffer. A flexoffer é composta pelo padrão de consumo energético e da janela temporal em que esse consumo pode começar. Esses parâmetros são relacionados com os mercados energéticos usando o Aggregator, entidade responsável por reunir todas as flexoffers numa região. O mercado irá indicar a melhor hora em que a flexoffer pode começar, em que a melhor hora é quando os preços estão os mais baixos.

O projeto baseia-se na implementação de um piloto capaz de aplicar o conceito da flexoffer a uma situação real. Para tal foi criado o FlexHousing. Permite o controlo do consumo energético dos aparelhos dentro de uma casa ou edifício. Tal é apenas possível através a utilização de tecnologias externas. VPS, Virtual Power Solutions, uma empresa nacional de soluções energéticas desenvolveu um produto chamado Cloogy. O Cloogy é um gateway instalado no edifício do utilizador e é capaz de comunicar com os aparelhos elétricos desenvolvidos pela VPS. Neste caso, esses aparelhos são fichas inteligentes. As fichas são colocadas entre o aparelho a controlar e a ficha de eletricidade. As fichas possibilitam o controlo de fluxo de eletricidade que vai para o aparelho. As fichas também têm sensores, capazes de recolher informação sobre consumo elétrico do aparelho, entre outras métricas. A VPS também forneceu a utilização de uma API externa, capaz de receber pedidos e de os reenviar para a fichas.

FlexHousing é composto por 2 módulos: o FlexHousing Middleware e o FlexHousing FrontEnd. O middleware age como centro de todas as operações no ambiente FlexHousing. Consegue ligar os diferentes sistemas, os serviços flexoffer e os serviços VPS. O ambiente FlexHousing permite a criação de salas e o registo de aparelhos equipados com a tecnologia VPS. O utilizador pode então proceder à aplicação de flexoffers, controlo remoto dos aparelhos e a recolha de dados a partir dos sensores nas fichas da VPS. Em relação à interação com o utilizador, o FlexHousing FrontEnd foi desenvolvido. Fornece uma interface gráfica com grande usabilidade, para que o utilizador possa configurar e controlar o middleware. A interface trabalha sobretudo com as salas e com os aparelhos. O utilizador tem então a possibilidade de verificar as flexoffers que aplicou e os horários que daí originaram. Se pretender, o utilizador pode instalar as fichas em aparelhos que não compatíveis com flexoffers, tais como candeeiros, televisões ou computadores. Usando o FrontEnd o utilizador pode esses aparelhos controlar remotamente.

Notation	Meaning
CISTER	Research Centre in Real-Time &Embedded Computing Systems
DEI	Departamento de Informática
FH	FlexHousing
FHFE	FlexHousing FrontEnd
FHMW	FlexHousing Middleware
FO	Flexoffer
IoT	Internet of Things
ISEP	Instituto Superior de Engenharia do Porto
LEI	Licenciatura em Engenharia Informática
REST	Representational State Transfer
RESTful	Characteristic of a device/system that conforms the constrains of REST.
RUP	Rational Unified Process
SOA	Service Oriented Architecture
SoS	System of Systems
UML	Unified Modeling Language
VPS	Virtual Power Solutions

INDEX

FIGURE INDEX

TABLE INDEX

1. INTRODUCTION

This section will introduce the reader regarding the framework of this project, its goal and proponent.

1.1 FRAMEWORK

CISTER Research Centre has one great work and academic environment: it allows for challenging state of the art problems, dealing with international partners and means to expand yourself as an academic student. If the work is appreciated by any international partner, it the first step for an internship abroad, where you may experience new challenges and new situations, not available if you were to stay in your country. Furthermore, the Internet of Things research and application area is about to expand [1]: by being in such an early approach, one is able to be ahead of the curve and be part of the creation of new paradigm, similar to the web based and mobile services behaved in the last decade.

Arrowhead is project focused on “reinventing” efficiency and flexibility by using collaborative automation. By using networks of embedded devices, it is enabling interoperability and integration of nearly any device. Arrowhead’s goal is to implement and evaluate pilots for different application fields (Smart building, industrial production, energy production); Enable integration with older/legacy systems; Provide a framework for the said purposes.

The Arrowhead Framework builds and transforms the local cloud to meet automation system requirements regarding: Real time properties; Security and safety; Engineering of automation functionalities. A local cloud is defined a collection of devices or IoT in a relative proximity. The local cloud always provides a number of basic core services enabling fundamental Service Oriented Architectures(SOA) properties like service registration, service discovery, authentication and authorization plus orchestration of system of systems (see Figure 1).

The ServiceRegistry System -Allows a service that is producing to expose itself to the cloud and allows consumer services to discover producer services they wish to consume from.

The Authorization System-Is responsible to control which service can consume a certain producer.

The Orchestration System-Allows coordination (orchestration) of which producer will a certain consumer be able to employ/use.

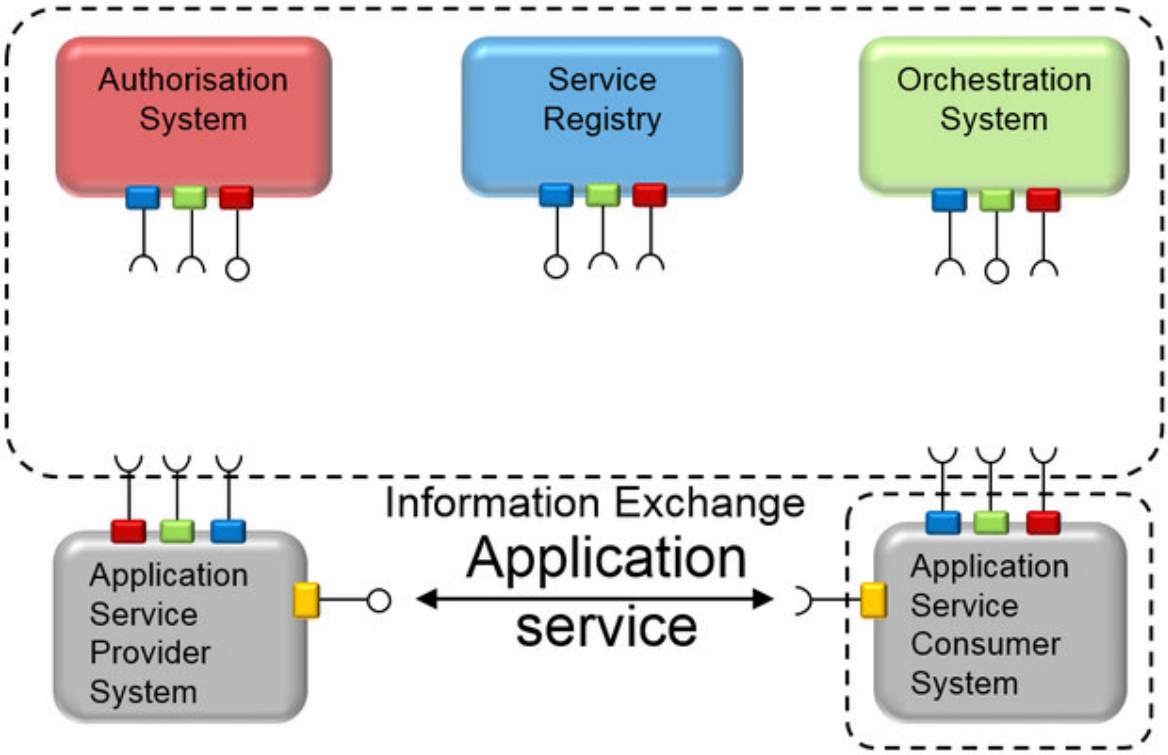

Figure 1- Arrowhead Framework System of System [2]

A device is defined "Arrowhead-compliant" when with its features respects 3 items:

- design according to Arrowhead Framework Templates- It must be described using a System of system template document.
- Adapt legacy system or implement new system according to Arrowhead Framework principles/patterns
- Perform interoperability tests- A compliance test tool is available to test the interoperability.

1.2 PROJECT GOAL

This project's goal is to twofold:

- Validate the flexoffer concept in a new environment
- Validate the application of the Arrowhead Framework with non-Arrowhead compliant devices.

This proof of concept revolves around creating a pilot for a home system that can use Flexoffers to maximize the energetic efficiency of home appliances using a hub for the communication with the different devices like the in-house terminals (electrical sockets, light bulbs) and an external Aggregator. Flexoffers (FO) are a new concept which aims to balance energy demand and response by synchronizing consumption with production. The FO concept, which works by scheduling when energy is consumed, has the advantage of reducing the energy costs of industrial or domestic installations. In order for the FO to be relevant for the Energetic Market (EM) it has to be combined or aggregated with other FO's on the same energetic grid. The Aggregator receives the FO from all the house-holds and combine them and send it as a unique FO. When the EM reply arrives, his responsibility is to redistribute the energy between the house-holds it supervises. For this project, the external technologies that are in use are smart plugs provided by a Portuguese energy solution company. The employment of these plugs allows us to enforce the schedule sent back from the AG.

1.3 ORGANIZATION OVERVIEW AND PRESENTATION

CISTER is one of the leading international research centres in real-time and embedded computing systems. In both the 2003 and 2007 evaluation processes, the Unit was granted the classification of 'Excellent' (the highest possible mark at that time) [3].

CISTER has been focusing its activity in the analysis, design and implementation of Real-Time and Embedded Computing Systems. Particularly, CISTER has provided advances in architectures for distributed embedded real-time systems, real-time wireless sensor networks, cyber-physical systems, middlewares for embedded systems and on the usage of multicore processors on real-time systems. CISTER is consistently involved in several collaborative European, national RTD projects and initiatives. Relevant to this project, the unit was a core partner in the ArtistDesign and CONET (Cooperating Objects) European NoE, EMMON (EMbeddedMONitoring) projects, lead the International SENODs project (together with Portugal Telecom and the Carnegie Mellon University), on energy-efficient data centres and has recently finalized the ENCOURAGE (Embedded iNtelligent COntrols for bUildings with Renewable generation and storage) project. Currently CISTER is involved in several European projects, particularly relevant to this project is the participation, in

multiple workpackages of the Arrowhead project, the participation in DEWI, CISTER is also participating in the MANTIS project, which is just about to start.

1.4 CONTRIBUTIONS

This project represents the only solution that integrates the sole implementation of the Arrowhead concept of flexoffers with technology to manage appliances, given in this case by a national company. This platform, in its current stage is, easily deployable, only needing a device to run the jar and internet connection, as long as the user has an installation of the external technologies (A Cloggy gateway and compliant devices such as plugs or sensors) approached further is this paper. FlexHousing, marketing/code name given to this application, empowers the user with the full control of his devices. This platform allows monitoring, flexoffer application, actuation on-the-go and even energy control/planning. The user has the information to understand and know what is going on in his facility, this being either his house or industrial site. The existence of the web interface also makes this platform the only one for an energy manager, in regards of the flexoffer. The existing ones only target the user and from an energy point of view, not in energy or device manager one. There already implementations that manage flexoffers but FlexHousing is the first to manage energy and flexoffer compliant devices.

1.5 REPORT STRUCTURE

This report is divided in four main chapters, Context, Work Environment, Technical Description, Testing and Conclusions.

Chapter **Erro! A origem da referência não foi encontrada.**, Context, starts by describing the problems in question. It gives an insight on the various areas that had to be studied in order to create a valid solution. It also gives a description of the solution that was developed.

Chapter 3, Working Environment, depicts the how the work was planned and what were the technologies that were used to create the end product.

Chapter 4, Technical Description, how the name implies, describes the implementation of the solution from a technical point of view. The 1st sections explains how the description is documented. In this case, it's made by using the Arrowhead project documentation.

Chapter **Erro! A origem da referência não foi encontrada.**, Tests Description, focus on software testing, in which we have the description of each test..

Chapter 6, Conclusion, describes the conclusions regarding all aspects of the project, both technical and management. It approaches the limitations, future work, goals achieved and gives a final overview of the internship and project

2. CONTEXT

This section will allow the reader to understand the origin and theoretical stand-point of the project, how solution will be tackled and its design.

2.1 PRESENTED PROBLEM

Flexoffer is the idea that has been developed within the EU FP7 project MIRABEL [2]. It permits exposing demand and supply loads with associated flexibilities in time and quantity for energy commerce, load levelling, and different use-cases. Flexoffers are generic entities, and may accommodate numerous varieties of consumers (e.g., electrical vehicles, heat, pumps, home equipment, industry) and producers (discharging electrical vehicles, solar panels). it's presently undergoing an EU standardization process through CEN/CENELEC CWA. In its simplest form, a flexoffer specifies a quantity of energy, a duration, an earliest begin time, a latest finish time, and a price, e.g., "I want ten KWh over two hours between one AM and five AM, for a value of 0.35 DKK/kWh". An additional complete example is shown in Fig. 1 [4].

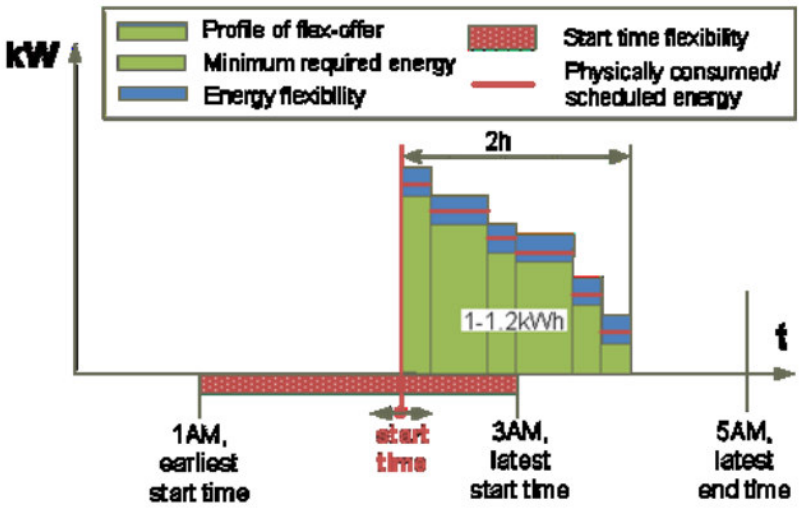

Figure 2 – Typical Flexoffer example [4]

Flexoffer are built by supplying information about the earliest start time, latest star time and the time slice with their respective high and low load of energy. This information is needed to better meet the energetic needs: during a time-slot, the energy usage may vary, thus, we may find a better solution if we work between values. Flexoffers can be applied to low producing/consuming entity such has households but can also be applied to larger ones such as factories. Flex-offers will be aggregated and disaggregated irrespectively to a kind of consumption or production they represent. Each collective and non-aggregated flex-offers may be mixed and dealt uniformly. A flex-offer may be seen as a sort of "option" that a user wants put out on an energy market. The flex-offer may be rejected, for instance if the price isn't right. If the choice is accepted, the flex-offer is

given a schedule, e.g., the flex-offer is scheduled at two AM, and the consumer system is notified. On the best case, the schedule is applied as specified. However, one of perks of using flex offer may only be put in play once things don't go as expected, for example as a result of an abrupt drop in wind energy. In this case, the flex-offer will be rescheduled, shifted to three AM, once the wind has come back. [4]

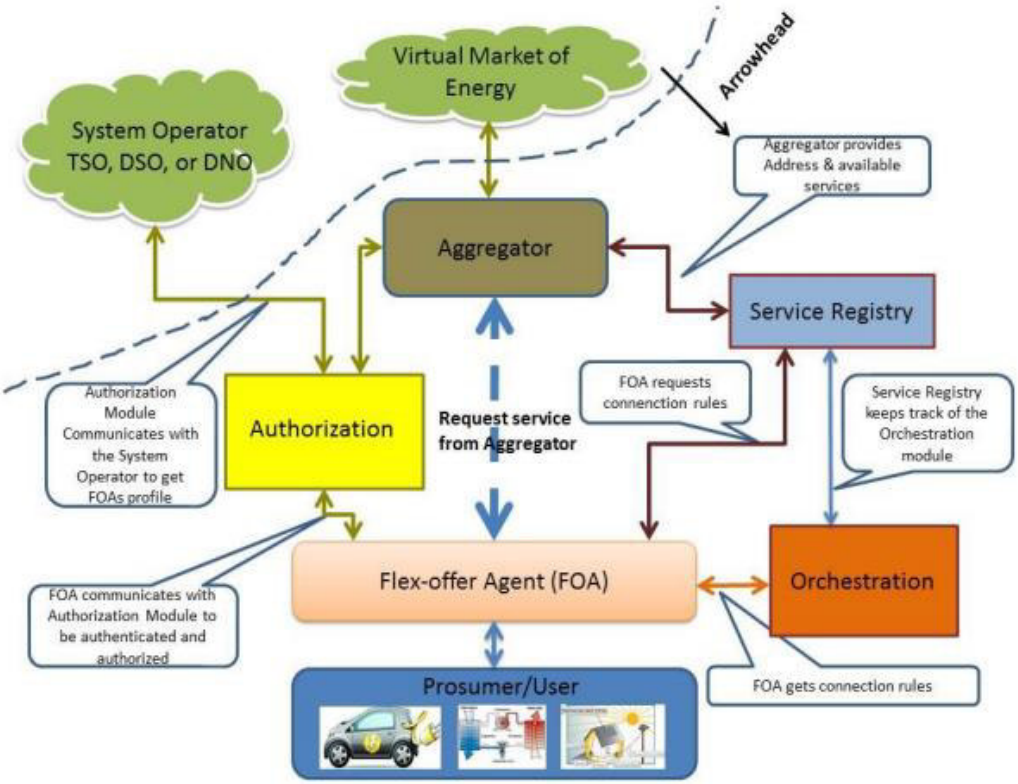

Figure 3- High level design for the Virtual Market of energy Flex-offer Agents [4]

The planned architecture is structured upon 5 modules, where 3 belong to the core arrowhead framework – Service registry, Authorization, and Orchestration, and the other 2 modules exchange business logic information – the aggregator and the Flex-offer Agent (FOA). [4] Its design already provides functionalities for obtaining info regarding the creation of flex-offers, the power consumption profile of specific devices, and control the execution of a scheduled flex-offer. Flex-offer Agents are very versatile, its design permits for its total implementation to be running on one device or distributed through many devices. As an example, the company providing the flex-offer service may offer the user a specific hardware device, solely for the FOA. The Aggregators receives flex-offers from FOAs, combines them with flex-offers from other sources, into larger macro flex-offers and then puts them to the Virtual Market of Energy. Note that Flexoffers need a specific dimension before being able to be bided on at the Energy Market Afterwards, the aggregator receives a response from the Virtual market of Energy, disaggregates the response and sends the FOA its consumption schedule. Many types of Aggregators may exist,

and a few Aggregators may be more specific for the management of electrical motors whereas others may be more adequate for the management of heating systems. In addition, selecting the most adequate aggregator additionally depends on the geographic region. To get the information of a specific aggregator, a FOA uses the Service registry module, to register itself, and the Orchestration module to find the aggregator that matches the needs. Has stated before, both modules can be reach using the arrowhead system. The Service Discovery service has already 2 implementations, one uses DNS Service Discovery (DNSSD) [5] and another uses Berkeley internet Name Domain (BIND) [6] [7]. The security issues are dealt by the Authorization module. This module is by different entity, most often the electrical company, with whom the prosumer is supposed an agreement to profit from flex-offers. By using a combination of Public Key Infrastructure (PKI) and X.509 certificates over REpresentational State Transfer (REST) protocol the Authorization module was built [8]. All communications may be encrypted by using Extensible Messaging and Presence Protocol (XMPP) [9] over Transport Layer Security (TLS) [10]. After that, the FOA and therefore the aggregator should first obtain its security certificates through the Authorization module. Only after, are the modules able to connect. The FOA obtains the address of the Orchestration module using the Service registry module that stores info concerning all of the accessible service producers, their location and characteristics. As to being discovered, an arrowhead compliant module will begin by communicating with the Service registry module and then register itself by providing info concerning its address and its offered services. The Orchestration module contains info regarding connection rules and system configuration, so permitting the FOA to get the address of the most adequate aggregator for the devices controlled by it. The aggregator or a system administrator ought to antecedently have configured the connection rules for the aggregator considering its geographical location and therefore the kind of devices that it's enabled to receive flex-offers from. This feature allows FOAs to connect to Aggregators that implement specialized algorithms capable of optimizing the results for specific styles of devices. The Orchestration module has been implemented using REST. The aggregator and the FOA communicate using XMPP, and exploits the present arrowhead to establish connections by using their services and protocols. The main advantage of XMPP is that it supports the Publish/Subscribe communication paradigm, that provides an asynchronous and extremely scalable many to-many communication model. In addition, XMPP is additionally in the process of being standardized.

2.2 BUISNESS AREA AND TECHNICAL CONTEXT

The internet was created to help research by connecting various universities [11]. It started as an exchange of packets with a speed of 56 kb/s and as evolved into the World Wide Web we know today. Today's society uses the internet for various purposes such has net browsing, using Mozilla Firefox or Google Chrome, communication with emails or Instant Messaging and to share various media, videos, photos or music. More and more devices are now able to connect themselves onto the internet: Cell phones, Cars, Appliances. If any object has some integrated electronics capable of running some basic software, it can be connected too. This allows it to send or and receive data and act depending on it. An electrical socket can be turned off after a server sent it a shutdown signal or an oven can be turned on after the user used an App on his cell phone

to send a message to do so. The number of these “Things” that are currently connected and active is increasing [12]. The next step is to diminish the interactions between man and things and have exclusive thing to thing communication. Such evolution will be accompanied by various solutions in terms of devices, technologies and performance. New technologies have been invested in, in order to have a better performance while old ones have been refurbished. IoT can have big impact on our society and on our economy. The transport industry can be heavily impacted too: having an accurate reading of passengers can have an impact on which route of a sub way grid reinforce, therefore reducing the waiting of the travelers and attaining a better response time. There also place for IoT in medical care: A little microchip implanted in a patient can track its movements around the hospital and have an on-the-go reading of the medication he has already been prescribed. The solutions are limited by our imagination and by our current technology.

IoT has various methods to apply itself to the real world: If you must act as a response to a certain event then you can use a IoT solution to make it happen. Some solutions are already in place around us: newborn babies in most modern hospitals have a small tracker either under their skin or in a snug-fitted bracelet. This allows to track their locations and act with the utmost quickness, either to avoid a serious condition or to avoid theft. Sensors and devices are all around us.

2.2.1 ENERGY SOLUTIONS

Most energy companies bill you for the power you rent: even you don’t consume has much as you are given, you will still be billed. By having a sensor directly connected to main power line going inside your house, which is normally the cable that goes into the meter, you can have an accurate reading off how much you are actually consuming. The electrical grid is intelligent, hence its new name: smart grid (see figure 4).

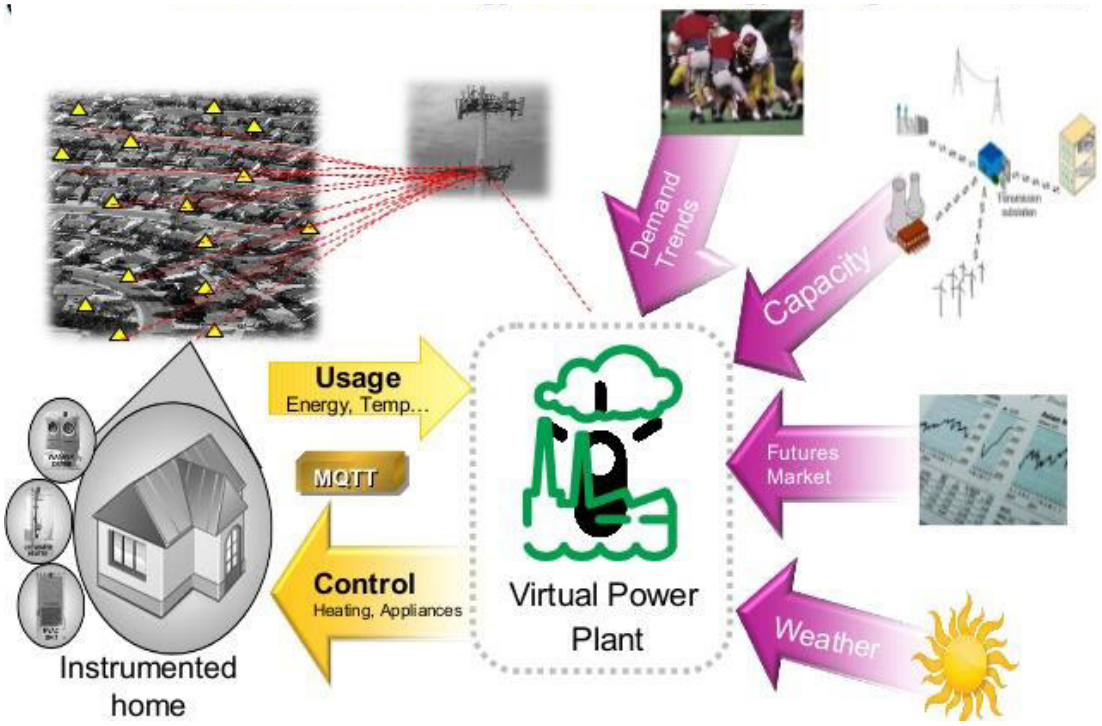

Figure 4- Real Time Energy Management in a smart grid [13]

There are some solutions for outside public lighting employing smart grids [14]: Having sensors on street lamps that will receive that on the luminosity so that the intensity of the light that the lamp is giving can be related to current natural lighting. You could also relate the luminosity emanating from the street lamp with the time and day of the week: On week days, people are less active during the earlier hours of the day in contrast with the night life during the weekends: More people equals more need for lighting. These kinds of problem lead to a new paradigm: the old hierarch unidirectional grid isn't built correctly: the smart grid needs more connectivity, bi directionality and is more complex. The end users are now able to be relevant to the grid in consumption and also production. These users are known as prosumers. With the number of prosumers increasing every year, the last mile of the grid needs to be structured as it will behave differently from before this surge [15]. A common trait of the entities or end points in the last mile of the smart grid is the usage of sensors and actuators that will manage the energy and handle the schedules. The ones currently available in the market don't have the computing power to handle complex or heavy protocols: the sensors normally act as relays, sending their data to a gateway installed inside the building. The gateway is connected to internet, bridging the interaction between the user, the sensors/actuators and the grid [16].

2.2.2 SMART HOUSING / SMART BUILDING

Having sensors installed on appliances can enable us to receive information about them and make them do actions in consequence. When you leave your work place, you can, with your cellphone, make you AC turn on so that when you arrive home, the room temperature is as you like. Studies have revealed that each appliance has a unique energetic signature. A freezer has a big spike in consumption has it starts it's cooling cycle, then going to a more moderate one finally shutting off completely when the desired temperature is reached. Controlling which appliances are consuming the most energy can allow the user to react to any kind of unnecessary spending and actually do some savings. It can have a big impact on security: Closed Circuit TV (CCTV) cameras can be connected directly to the internet and allow instant viewing of the exterior or interior of your house. Door control over the internet can enable you to open the doors even if you're not home. Monitor the movement inside your house be connecting the sensors to a database. This redundancy can be used in case of a break in. This can also save lives: if the routine of a family is to wake up around 7 in the morning to start their day, if no movement is picked up in the living room or kitchen at around 8, it can make the phone ring or even send a text or email to someone to check if that situation is normal. As long has it has some kind of connectivity any device can be "smart" (see figure 5)

There are already products using IoT for smart housing:

- Neurio – By installing a Wi-Fi enable device with the power/electrical panel, through the use of sensors, Neurio can monitor the usage of each device, identifying them by their energetic signature [17];

- Canary – A single hub contains sensor for air quality, sound, motion, temperature and motion. By using machine learning it can establish the regular behaviour of your house hold. Any data capture by those sensors outside of the standard behaviour triggers the emission of an email or an alert on their mobile app [18];

- Ivee Sleek- An alarm looking device, enabling you to talk with the house. It connects to various other smart devices, allowing you to speak commands to them. Also able to report the weather and other news through the usage of internet [19];

Figure 5- IoT house-hold fields of application [20]

2.2.3 ENERGY MARKETS

The energy markets consist of producers, transmitters, distributors and consumers. The producers create the energy from power plants. They can be from either fossil or green origins. The transmitters are responsible for the supply of energy, taking it from its origin to the demand points (eg: from the dam to the city) and the distributors for the local distribution within the city (the overhead cables in the streets). Society is, nowadays, becoming a low consumption economy, driven to use more competitive prices and greener energy. A European agreements of commitments known as “20/20/20” has set goals, regarding energy for 2020: A minimum of 20% reduction in GHG emissions, 20% of energy production coming from renewable resources and a 20% reduction in energy usage, by upping energy efficiency. In order to stimulate international development of renewable energies, Renewable Energy Certificates System were implemented, relating the power being sold and its impact in the environment. By 2020, under EU legislation, 80% of consumers will have smart meters, allowing a greater efficiency, more competitive prices and a better demand-response management. The nationwide is responsible, fed by power plants,

transport the energy through the country, normally with 400 and 100kV. Then we have regional grids, relaying the national grid to the local one, with energy, typically, within 130 and 5kV. Small energy producers (large wind farms or decentralized power plants) and large consumers (large industrial complexes) can be connected to the regional grid. The local grid provides the consumers with 400V energy. Domestic wind and sun power plants are connected to the local grid. Upon obtaining the obligatory licenses and agreements, generation and consequent supply of electricity, plus the administration of electricity retail markers are fully open to competition. Still, public concessions are needed for the assignment of the transmission and distribution of components of the electricity industry.

Generation of electricity can be divided into the ordinary regime and special regime.

The ordinary regime refers to the majority of electricity production. In Portugal, the rights to manage the 26 large hydropower plants are retained by EDP (Energias de Portugal) and the transmissions are managed by the National transmission grid, currently credited to REN Rede Eléctrica.

The electricity market is opened since 2006, with all consumers in the mainland being able to choose their electricity supplier, without additional charges for switching companies. On the other side, suppliers can openly buy and sell electricity, and can access the national transmission and distribution networks upon payment of the required fees, set by the Regulatory Authority. In 2001, the Portuguese Government presented several energy efficiency measure, with the aim to “support the progressive implementation of telemetering for electricity, water and gas as a strategy for distribution network and quality service improvement”. Later that year, together with the Spanish Government, approved the Electricity Iberian Liberalized Market (MIBEL), stating that citizens from the Iberian Peninsula could purchase electricity freely from any agent in the market. Compulsory installation of digital meter for new facilities and the replacement of all traditional meters by smart meters are some of the agreements stipulated for MIBEL. In 2011, Portugal started a wide implementation of Smart Electricity Grid (Smart Grid), with the final goal to cover six million of all customers until 2017.

The special regime is associated with the production of electricity, via renewable sources. Generation of electricity under the special regime is subject to different legal requirements, but on the other hand benefits from special pricelists. Portugal is one of the countries in the European Union (EU) most dependent of fuel importation (77% in 2010). Hydropower in the most significant renewable energy source, with the total share of renewable energy consumed of 22,9% in 2010 [21].

2.4 SOLUTION OVERVIEW

By using external technologies, in this case provided by VPS [22], a Portuguese energy solutions company, we are able to apply the Flex-Offer(FO) concept to a real-life situation. By using smart-plugs as our actuators and sensors we can control the energy usage of appliances in order to make them work, respecting the Flex-Offer created on them. FlexHousing is a middle-ware application that communicates with the flexoffer cloud through its components, such as the `FlexOfferInterface` and `FlexOfferAgent`, and with VPS API, that provides the necessary services to operate with the smart-plugs. The application runs in a device, inside the building, and provides services through an API so that the WEB-Service, we also implemented, allows the user to configure FlexHousing to his needs. The application also allows the user to check the energy consumption of any plugs he has installed, regardless if a Flex-Offer has been applied or not. This also helps the user get to know his appliances better and make informed decisions about whether or not to apply a certain Flex-Offer to a device. Once a FO has been applied to a device/plug, the application maintains the energy usage of the said device linked to that FO until the user chooses otherwise, by either establishing a new FO or just remove the device from the FO interface of the application. The WEB-Service server is running externally and provides an easy-to-use graphical-interface for the user to input all the information, regarding the house, FO's and devices.

From the user's point-of-view, FlexHousing will allow the user to:

- Manage his devices- The user is able to register new ones, edit the configuration of old ones (either name or location). We are currently only using smart plugs but the application will allow to use any device provided by VPS (Water temperature sensors, gas sensors, etc.) [23]. If the user wants he can also turn off and on any device. Every plug has the capability to cut the flow of electricity (The execution of the command is what allows us to apply the FO concept);
- Check the Consumption – Because we are using ISA technology and devices, we provide the user the possibility of checking the current consumption of any appliance/plug you have installed. You have the possibility of monitoring a device for a given period of time, resulting in a graph of energy consumption throughout that period;
- View and manage his Energetic Profile- The aggregation of every FO the user has in his building will give us his Energetic Profile. With this option, we allow the user to create the FO for his appliances. The user inputs the information needed for the FO creation such as start time, end time, latest start or finish and more importantly the energy for each time-slice. If available, the consumption graph of the device he is trying to setup will display in order to help the user build a correct FO. Nonetheless, the user is able to setup any kind of energy configuration he wants;
- Manage the house – Create new rooms in order to have devices be placed there. This also allow the user to have an overview of the house and check the relationship between the rooms and devices;

3. WORK ENVIRONMENT

The work developed for the internship differs from a traditional software development: new products are services are being created in order to create a proof of concept demonstrator. This work will be a staple for the Arrowhead European Project.

3.1 WORK METHODS

For the accomplishment of the initial proposed project objectives, an iterative work method was adopted, using the Rational Unified Process technique.

According to the 4 phases of RUP [24], the project was distributed in the following way:

1. Inception – Research and code analysis. It was studied the best technologies to put in practice for the project.

2. Elaboration – Code design, such as functional and non-functional requirements were documented.

3. Construction – All the use cases were implemented.

4. Transition – Some prototypes were made and posted on practice, resulting on code tests and improvements.

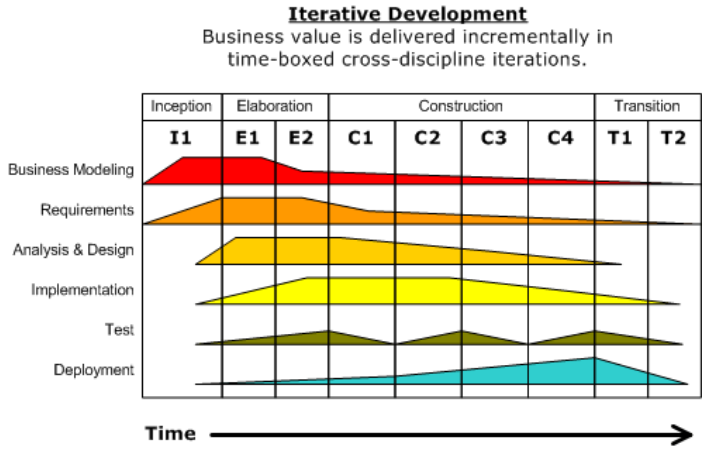

Figure 6- Typical RUP Chart [25]

Because this project was being worked on in a research centre, it has various partners with different responsibilities. This work is to be seen as a demonstrator of concepts and solutions, in this case developed beforehand. The demonstrator has the responsibility of showing the reliability of the Flexoffer for the real world. That was the only requirement for this project therefore any sub adjacent analysis and design were directed to features and use-cases created by myself. The analysis & design step took longer because of the high load of theoretical concepts and by the fact that most of the code directly linked to this project was created by partners in Aalborg, Denmark.

During the project, various milestones were implemented (see Table 3).

3.2 PROJECT PLANNING

A considerable amount of new technologies was introduced therefore a large amount of time was allocated for their adaptation and introduction. Several small tasks were necessary before actually tackling the project, such as installing the AH server, testing the technologies developed, testing the devices received from the partners. Table 1 represents the Gantt diagram with the deadlines for each task approached in this project.

Table 1 - Gantt diagram for the internship

Task	Duration	Begin	End
Kickoff	1d	25/02/16	25/02/16
Studying Framework Arrowhead	232d	25/02/16	14/10/16
REST e XMPP in Java	7d	25/02/16	04/03/16
Dummy application using grizzly	3d	25/02/16	29/02/16
Installation and maintenance of Openfire Sever	4d	29/02/16	03/03/16
Testing of Dummy application	5d	29/02/16	04/03/16
Local installation of Arrowhead Core Services	6d	04/03/16	11/03/16
Studying of documentation	4d	04/03/16	09/03/16
Installation on physical machine	3d	07/03/16	09/03/16
Service configuration	3d	09/03/16	11/03/16
Dummy application using local Arrowhead Deployment	6d	11/03/16	18/03/16
Design and implementation	4d	11/03/16	16/03/16
Testing	3d	16/03/16	18/03/16
Pilot for flexoffer concept	100d	21/03/16	05/08/16
Design	90d	21/03/16	22/07/16

Investigation of previous Arrowhead and flexoffer applications. Try-outs for various architectures using the smart plugs. Each new design was introduced using UML then they were submitted to the supervisors for acceptance. All the Use-Case were also accepted by the project supervisors.			
Implementation	80d	11/04/16	29/07/16
The implementation had 2 phases: the middleware, hosting the services and the business logic, and the web-server, serving as a front end “renderer” for the user to manage his devices. The front end was only started after the middleware was in an advanced stage of development.			
Testing	65d	09/05/16	05/08/16
Unit, integration, deployment and performance tests			
Final Report	100d	21/03/16	05/08/16
Structure and Content	85d	21/03/16	15/07/16
Result Gathering	50d	09/05/16	15/07/16
Result analysis	6d	18/07/16	25/07/16
Revision and Proofreading	10d	25/07/16	05/08/16
End of internship Tasks	51d	05/08/16	14/10/16
Writing of an article about FlexHousing for INForum 2016. Preparing a presentation and video for the presentation at INForum 2016.			

Nearly every milestone and deadlines were met, even though some new tasks were introduced that might have delayed the project. Quick reaction and effective planning allowed us not to be delayed. At the end of the project, due to some delay and due to time constraints the implementation of the FrontEnd had some setbacks and some the deadlines in the original planning were crossed. One of the features wasn’t implemented.

3.3 MEETINGS & MILESTONES

For each major decision or event, we had a meeting informal meetings, either to decide to employ a solution or approach or to check if a decision was correct. The meetings took place either in the supervisor’s offices or at the work station. Milestones were establishing to control the workflow and to have palpable results, in a gradual and incremental way.

Table 2 - Meetings during internship

Date	Presences	Description
25/02/16	Joss Santos, Luis Ferreira, Michele Albano, Paulo Barbosa, Renato Ayres e Pedro Moura	Discussion about each project and projected results
10/03/16	Joss Santos, Luis Ferreira, Michele Albano, Constantino Martins	Meeting between ISEP and external supervisors
11/03/16	Joss Santos, Luis Ferreira, Michele Albano	Discussion for code design
06/04/16	Joss Santos, Luis Ferreira, Michele Albano	Project status
02/05/16	Joss Santos, Luis Ferreira, Michele Albano	Project status and demonstration of prototype
09/06/16	Joss Santos, Luis Ferreira	Demonstration of features
19/07/16	Joss Santos, Michele Albano	Discussion for possible publishing of paper
31/08/16	Joss Santos, Luis Ferreira, Michele Albano	Demonstration of FrontEnd prototype
15/09/16	Joss Santos, Luis Ferreira, Michele Albano	Final Demonstration

The millstones were decided by the supervisors, except when it was required for an event such as an event at CISTER or an external one. Such is figured in Table 3.

Date	Description of milestone	Day it was finished
01/05/16	Prototype of Middleware capable of connecting to all the services	01/05/16
06/06/16	Final implementation of Middleware	07/06/16
31/08/16	Prototype of FrontEnd capable of communicating with middleware	29/08/16
15/09/16	FlexHousing as a complete system	15/09/16

Table 3 - Milestones for the project

3.4 USED TECHNOLOGIES

This section will briefly approach every technology used, at some time, along the project. Beneath, you will find Table 4 containing information about the technology and regarding where it was used.

Table 4 - List of technologies and libraries used and where

Name	Where it was Used
JAVA	All the code used in the middleware was written using JAVA. The code received from the Arrowhead platform and from the partners was also written in JAVA.
Jersey & Grizzly	These JAVA libraries enable the usage of services hosted by an API. These are used in the middleware in order to consume service hosted by API but also to host them.
Apache Derby	The data stored in the middleware. The persistence is guaranteed by an Apache Derby database.
XMPP	XMPP is used for communication between some of the modules in FlexHousing. It allows for secure message exchange.
Linux	A local server of the Arrowhead core services was installed at CISTER. The ISO installed used a CentOS distribution.
C#	The web client code is written in C#. The FrontEnd system is going to run on a ISS server.
MVC	The web client will mostly do CRUD operations. MVC allows an easy interaction between app and user.
.NET	This framework provides several components, essential for web clients.

Each technology usage was heavily discussed. Several approaches with other technologies were possible but these were the ones that were either chosen or continued (The WP5 modules were written in JAVA, therefore a JAVA written Middleware was the obvious choice)

Java

Java is an object-oriented programming language first released by Sun Microsystems in 1995. It is fast, secure, and reliable. From laptops to datacentres, game consoles to scientific supercomputers, cell phones to the Internet, Java is widely used. Java language is very based on C and C++ languages, many of Java´s defining characteristics come from this two predecessors, which are refinements and responses to the predecessor's limitations. [26]

What really defines Java is its portability, because to make C and C++ work in different CPUs it is needed a compiler for each type of CPU, and compilers are expensive and time-consuming to create. Therefore, back then Java founders decided to work on a portable, platform independent language that could be run every type of CPUs, leading to the creation of Java.

Currently the latest Java version is Java SE 8 and represents another very significant upgrade with the introduction of lambda expression. The purpose of lambda expressions is to simplify and reduce the amount of source code needed to create any functions. [27]

Apache

Apache Derby is simple but powerful: easily deployable, as it is seen as Java class library, but from a relational database point of view it has various features like crash recovery, rollback on transactions, query injection, views and even constraints for the primary and foreign keys.

Installing it is as simple as copying a jar file to your project. It can be embedded to a project or used as a stand-alone package. Because of this you can use the same package for various platforms and operating systems simply by copying a few files.

You can employ Apache derby in 2 ways: when its embed its part of the application. It's part of the Java Virtual Machine, launched and stopped at the same time as the project. The jar contains both the database engine and JDBC driver; In the network mode it can service application from different VM, or in other words, from different application in different coding languages such as php, python or even C. All you have to do is use the specific communication modules and you're good to go.

XMPP

Extensible messaging and Presence Protocol (XMPP) is an open XML technology for real-time communication, that powers a large range of applications as well as instant electronic communication, presence and collaboration. XMPP is a protocol; a collection of standards that enables systems to speak to each other. XMPP is employed wide across the net, however is commonly unadvertised. The presence indicator tells the servers what is your current availability. In technical terms, presence determines the state of an XMPP entity; in common man terms, whether or not you're there and prepared to receive messages or not. The 'messaging' a part of XMPP is the 'piece' you see; the instant message sent to the users. XMPP has been designed to send all messages in real-time employing a very efficient push mechanism. whereas existing net primarily based mechanisms usually build many unneeded requests introducing network load, and are consequently not real-time. Defined in an open standard and exploiting an open systems

approach of development and application, XMPP is meant to be extensible. In different words, it's been designed to grow and accommodate changes. [9]

BIND

BIND is open source package that implements the domain name System (DNS) protocols for the web. it's a reference implementation of these protocols, however it's also production-grade software, appropriate to be used in high-volume and high-reliability applications. The name BIND stands for “Berkeley internet Name Domain”, as a result of the code originated within the early Eighties at the University of California at Berkeley. [6]

BIND is out and away the most widely used DNS software on the web, providing a sturdy and stable platform on top of which organizations will build distributed computing systems with the knowledge that those systems are absolutely compliant with published DNS standards.

C-Sharp

C# syntax is very expressive, nonetheless it's also straightforward and simple to learn. The curly-brace syntax of C# is instantly recognizable to anyone acquainted with C, C++ or Java. Developers who understand any of those languages are usually ready to begin to work profitably in C# inside a really short time. C# syntax simplifies several of the complexities of C++ and provides powerful options like nullable value types, enumerations, delegates, lambda expressions and direct memory access, that don't seem to be found in Java. C# supports generic methods and types, which give enhanced type safety and performance, and iterators, that alter implementers of collection classes to define custom iteration behaviours that are straightforward to use by client code. Language-Integrated query (LINQ) expressions build the strongly-typed query a first-class language construct.

As an object-oriented language, C# supports the ideas of encapsulation, inheritance, and polymorphism. All variables and methods, together with the main method, the application's entry point, are encapsulated among class definitions. A class could inherit directly from one parent class; however, it may implement any variety of interfaces. methods that override virtual methods in a parent class need the override keyword as a way to avoid accidental definition. In C#, a struct is sort of a light-weight class; it's a stack-allocated type which will implement interfaces however doesn't support inheritance.

In addition to those basic object-oriented principles, C# makes it simple to develop software elements through many innovative language constructs [28].

APS.NET MVC

APS.NET MVC is a framework for building internet applications employing a MVC (Model view Controller) design. The MVC model defines internet applications with three logic layers:

- The business layer (Model logic)
- The display layer (View logic)
- The logic layer (Controller logic)

The Model is the part of the application that handles the logic for the application information. Often model objects retrieve information (and store data) from a database. The view is the elements of the application that handles the display of the information. Most often the views are created from the model information. Interaction is handled by the Controller. The Controller will receive the input from the view and act accordingly. Typically, controllers read data from a view, manage user input and send input data to the model. The MVC separation helps you manage advanced applications, as a result of you'll focus on one side a time. as an example, you'll target the view without depending on the business logic. It additionally makes it easier to test an application. The MVC separation additionally simplifies cluster development. different developers will work on the view, the controller logic, and also the business logic in parallel [29].

4. TECHNICAL DESCRIPTION

As partner of Arrowhead Framework, the developer is obliged to write the technical documentation following the Arrowhead templates and guidelines. Since most of this documentation approaches both project analysis and implementation, it was decided to put in this chapter all written documents

4.1 ARROWHEAD DOCUMENT METHODOLOGY

Every Arrowhead partner must document and describe its developed solutions according to the Arrowhead compliant documentation. This has the purpose of accomplishing a common understanding of every Arrowhead partner. The Arrowhead compliant methodology includes design patterns, documentation templates and guidelines that aim at helping systems to conform to Arrowhead framework specifications.

The Arrowhead compliant documents consist in three levels: System-of-Systems, System and Service levels Figure 7 depicts.

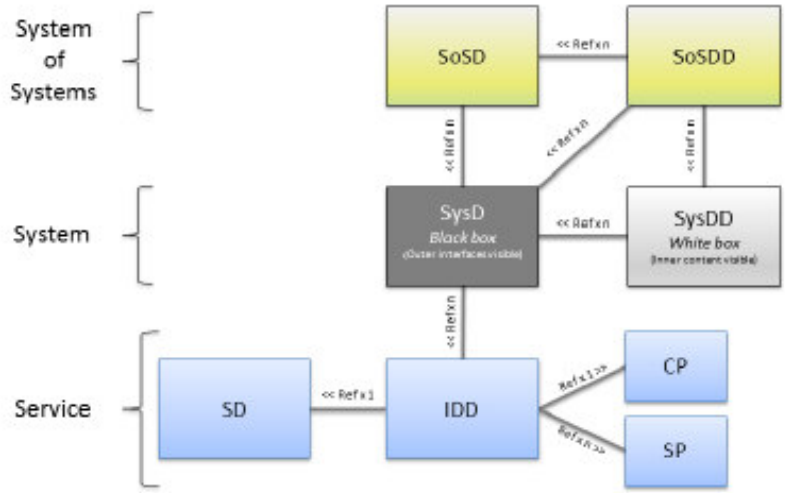

Figure 7 -The Arrowhead documentation relationships [30].

The approach is to apply terms “black-box” and “white-box” only in the System Level, in sense of writing an abstract high-level description of a system approaching only its behaviour, and on the other case in the sense of writing with detail the implementation done.

4.1.1 SYSTEM-OF-SYSTEMS LEVEL

At the System-Of-Systems (SoS) there are two types of documentation, System-of-Systems Description (SoSD) document and System-of-System Design Description (SoSDD).

SYSTEM-OF-SYSTEMS DESCRIPTION (SOSD) TEMPLATE

This document should contain an abstract high level view, describing the main functionalities and generic architecture, without referring any specific technology. Such document must include use-cases to help understanding the expected behaviour. Bases on these use-cases, the document should include behaviour diagrams. It is also recommended the support of UML diagrams, mainly component and activity diagrams [30].

In this document, it is also important to include information about non-functional requirements, which the security must be treated separately. This includes the definition of security principles that SoS needs to follow on a non-technical generic level, the security objectives and the assets which need to be protected.

SYSTEM-OF-SYSTEMS DESIGN DESCRIPTION (SOSDD) TEMPLATE

This document describes how a “System-of-System Design Description” has been implemented on a specific scenario, describing the technologies used and its setup. The document starts with an abstract high-level view of the SoS realization, describing how its main functionalities can be logically implemented. Specific use-cases are described next, supported by structure and behaviour diagram.

The non-functional requirements implemented by this realization must be listed along with its security features. To support the validation of the security attributes of this SoS realization, it is also necessary to include information identifying the data flows in the system as well as its threads and vulnerabilities [30].

4.1.2 SYSTEM LEVEL

At the system level there are two different representations, the “SysD Template” consists in a “black-box” design, while the “SysDD Template” consists in a “white-box” design.

SYSTEM DESCRIPTION (SYSD) TEMPLATE

This document provides the main template for the System Description of Arrowhead compliant systems. As a “black-box”, there should be a description of the main services and interfaces of a system without describing its internal implementation where all the system produced/consumed services are listed. It is recommended the use of component diagrams to represent the interoperability of different systems. This structural view can be complemented with a high-level behavioral view such as sequence diagrams [30].

SYSTEM DESIGN DESCRIPTION (SYSDD) TEMPLATE

This document provides the main template for the description of Arrowhead Systems, technological implementations, describing in detail the proposed solution. Here it is encouraged the usage of formal or semi-formal models in order to enable the automation generation of code from the specifications as much as possible. When automation is not possible, the document should be precise enough to guide developers towards an implementation that matches these specifications [30].

4.1.3 SERVICE LEVEL

The service level consists of four documents: the SD Template, the IDD Template, the CP Template and the SP Template.

SERVICE DESCRIPTION (SD) TEMPLATE

A service description document provides an abstract description of what is needed for systems to provide and/or consume a specific service. SD´s for Application Service are created (specified) by the developers of any Arrowhead compliant system and by the developers of the Core Arrowhead Framework services. The SD shall make it possible for an engineer to achieve an Arrowhead compliant realization of a provider and/or consumer of description of how the service is implemented by using the Communication Profile and the chosen technologies [30].

The document starts by describing the main objectives and functionalities of the service and follows on defining the Abstract Interfaces and an Abstract Information Model. On Abstract Interfaces section all interfaces should be detailed using a UML sequence diagram. The Abstract Information Model section must provide a high level description of the information model with types, attributes and relationships, bases on UML Class diagram. Finally, non-functional requirements must be described for each service.

INTERFACE DESIGN DESCRIPTION (IDD) TEMPLATE

An IDD provides a detailed description of how a service is implemented by using a specific Communication Profile and specific technologies. This document describes each of the interfaces in a separate sub-section, and the functions included in each interface. To support the descriptions, it is recommended the use of UML sequence, class and components diagrams. There must be an Information Model section present in the document, containing detailed information about the data formats used by the interface along with metadata information. [30]

COMMUNICATION PROFILE (CP) TEMPLATE

The CP document describes the types of message exchange patterns, defining in detail how the CP handles security issues, regarding authentication and encryption based on the protocol specifications. For instance, in the use of CoAP, DTLS is enabled. This document can be identified by three characteristics: transfer protocol (e.g. CoAP); security mechanism (e.g. DTLS); data format (e.g. XML).

SEMANTIC PROFILE (SP) TEMPLATE

The SP describes the data format by pointing out what its type (e.g. JSON; XML) and how specific a piece of data is encoded.

4.2 SYSTEM DESCRIPTION AND DESIGN

This section approaches each document written for the project. It presents the System of System Design and Design Description of the Pilot, followed by the Design and Design Description for each system implemented. Finally, it describes the services of the middleware using a mix of SD and IDD documents

4.2.1 SYSTEM-OF-SYSTEMS DESIGN

This document provides a high level overview of the FlexHousing system. It will approach the use case for the system as well as the system that are in play.

1. SYSTEM DESCRIPTION OVERVIEW

The pilot named FlexHousing aims to apply flex-offer concepts to a real life situation from the domotics area.
In particular, the FlexHousing System of Systems (FHSoS) was created to demonstrate the real life management of appliances based on flexoffers. FHSoS acts as a proof of concept, allows a user to create flexoffers upon devices, and lets the power usage of the device be dictated by the aforementioned flexoffer. In order to be made more user friendly, FHSoS provides graphical frontends and allows for the abstraction of devices: the user can organize his devices into rooms (Kitchen, living, room, garage…) that pertain to buildings/houses. The devices are equipped with different sensors to collect data from the environment and the appliances, and actuators to manage energy consumption of the appliances.
This pilot considers that most devices are smart plugs equipped with both sensors and actuators. The actuator of the smart plug allows to remotely switch on and off the appliance it is installed onto. Moreover, through its sensors, the smart plug can collect data regarding energy consumption, and the user can then create flexoffers based on past consumption. FHSoS is articulated into the FlexHousing middleware (FHMW), responsible for the flexoffers and devices management, and the FlexHousing fronted (FHFE), which is a graphical interface for the setup of user configurations and all-around managing of the building/house. The FlexHousing frontend is hosted by a web server and interacts with the FlexHousing middleware through the services it provides.
A context diagram of the systems is depicted in Figure 8.

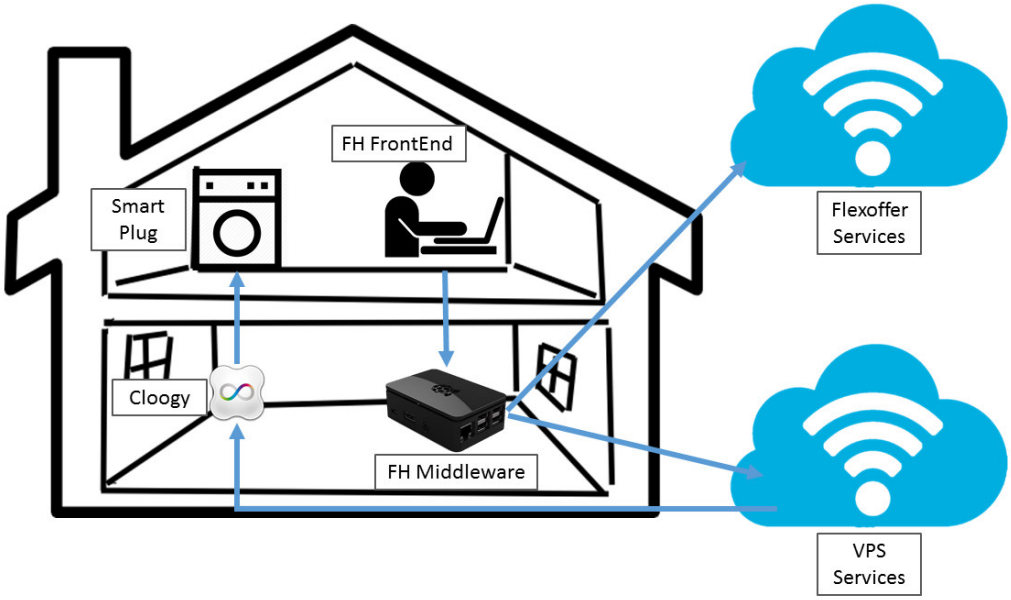

Figure 8 – Overview of the all the systems

The FlexHousing SoS communicates with the VPS API, which is exposed by an external service provider and communicates with the Cloogy gateway (the device with the infinity symbol in Figure 1) to interact with the appliances equipped with the smart plugs. Table 1 reports a summary of the concepts described in this section.

Table 5 Concepts related to the flexoffer pilot

Name	Description	Contains
Building/house	A location pertaining to a user	Rooms, Cloogy
Room	A physical place in a building/house	Devices
Device	A smart plug, usually connected to an appliance	Sensors and Actuators
Sensor	Collect data from environment, and from the appliance it is connected to	
Actuator	It provides remote control of the energy feed to the appliance it is connected to	
Appliance	An electrical machine that uses electricity. E.g.: HVAC systems, lights, washing machines	
FHSoS	The System of Systems supporting the FlexHousing pilot	FHMW, FHFE
FHME	The FlexHousing Middleware, which is controlled by the FHFE, and interacts with the flexoffer system and the VPS API	
FHFE	The graphical FlexHousing FrontEnd	

Flexoffer system	A set of Arrowhead services to interact with energy market to sell energy flexibility and buy energy	
VPS API	A service to drive the operations of the Actuators present in the Devices	
Cloogy	A gateway that connects the house to the internet, and thus to the Flexoffer system and the VPS API	

1.1 INTERACTION BETWEEN SYSTEMS

As already presented in Section 1, FlexHousing SoS is composed by 2 systems: the FlexHousing middleware (FHMW) and the FlexHousing front end (FHFE). The FHMW interacts with 2 other services, the Flexoffer service and the VPS service.
For the interaction with the Flexoffer service, the FHMW instantiates a session of communication using the Arrowhead Framework, which remains valid for future interactions with the Flexoffer service. With regards to the VPS service, the FHMW has to authorize itself against the VPS service, and future service fruition will be authorized by caching the result of the authorization step.

Both FHMW system and FHFE system have 2 states: logged and non-logged, and they ideally transit together between the two states.
When the user starts the interaction with the front end, he is immediately prompted to do the login. The user inputs his credentials and the FHFE requests the FHMW for authorization. When the FHMW transits to the logged state, as described in the next paragraph, it will also make the FHFE switch to the logged state. When it is in the logged state, the user is able to interact with the whole system, and all the features are now executable.
Regarding the FHMW, when it is in the non-logged state, it cannot perform any operation that involves devices of external services: actuation on devices, application of flexoffer or the retrieval of measurements. Anyway, the FHMW is still allowed for the management of the house (operations on the Rooms). When the FHMW executes the login operation against the VPS service, it is then able to do any FlexHousing operation, and it stores the authorization results received from the VPS service to authorize automatically all other operations it wants to perform. Table 6 depicts the location of the description of the systems. Figure 9 depicts the components that were developed for this pilot.

Table 6 - Pointers to the description of each system

System name	Path
Middleware	…\SysDD FlexHousing-MiddleWare.docx
FrontEnd	…\SysDD FlexHousing-FrontEnd.docx
Flexoffer Services	…\Arrowhead\Meetings\Multi WP Workshops \2013-11-05\ Porto\Documenting\Examples \SysDD\Arrowhead SysDD Aggregator_v0.1.docx
VPS Services	[Legacy System]

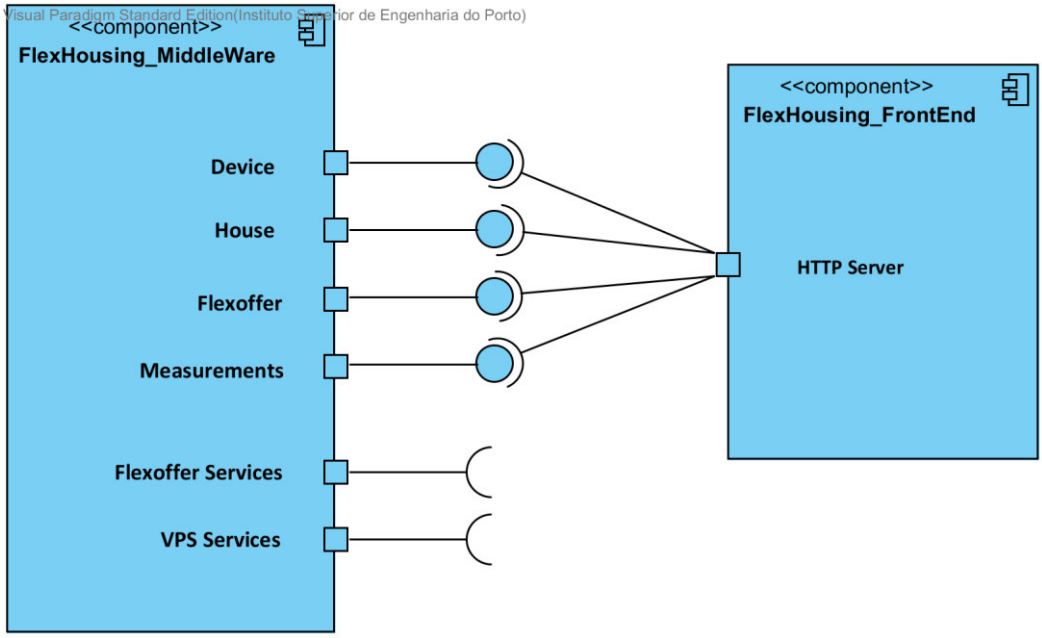

Figure 9- Component diagram for the systems

As figure 2 describes, the middleware provides 4 interface that will be consumed by the FrontEnd. In the other hand, the middleware consumes for 2 interfaces, one for the Flexoffer Services and another for the VPS ones.

2. USE-CASES

FlexHousing SoS offers the possibility to manage (create, categorize, etc) flexoffers, devices and the house. It also allows for operations on those entities (application of flexoffers on devices, etc).

2.1 FUNCTIONAL REQUIREMENTS

FlexHousing SoS has 8 major functional requirements:

- Send Flexoffer: the user selects a device to apply a flexoffer on. He will input the earliest and latest time the energy consumption profile can begin. Then he will form the pattern of consumption by indicating the minimum energy and the flexibility window. The flexoffer is created and it is ready to be sent to the Flex-offer services. The execution flow of this use case is presented in Table 3, and its message flow chart is represented in Figure 4.

- Receive Schedule: This feature consists of 3 parts: the emission of the flexoffer, the reception of the schedule and the creation of the time triggered emissions of actuation to enforce the received schedule. From then on the user is able to check the schedule that is active on any given device. The execution flow of this use case is presented in Table 4, and its message flow chart is represented in Figure 5.

- Login: The user has to login to be able to manage and interact with his devices. The login/password combination is received by FHMW and sent to the VPS services. From there on, the FHMW has the authorization to interact with the devices attached the VPS account the user used. When the user wants to interact, FlexHousing will execute the needed operations. The execution flow of this use case is presented in Table 5, and its message flow chart is represented in Figure 6.

- Manage House: This allows to do CRUD (Create, Read, Update and Delete) operations on the house, more specifically to the rooms. The execution flow of this use case is presented in Table 6, and its message flow chart is represented in Figure 7.

- Manage Devices: The managing of the device allows for CRUD operations on device but also on the sensors attached to said devices. When a device is registered, it has to be both at FHMW and at VPS. Although, devices may be already registered at VPS are passible to be registered in the FHMW, just by supplying the physical ID of the device, normally found on the device itself. The execution flow of this use case is presented in Table 7, and its message flow chart is represented in Figure 8.

- Check Measurements: Because the devices are equipped with power and tension sensors, the user can check the values for those readings. Those readings are stored at VPS servers and FHMW is able to collect them. The execution flow of this use case is presented in Table 8, and its message flow chart is represented in Figure 9.

- Actuate on Device: To apply flexoffers on to devices, they need the Actuator sensor to present on them but that doesn't mean you can't have non flexoffer compliant appliances (Lamps and TV for example) with those plugs. The user is able to control their power flow: he can turn them on and off remotely The execution flow of this use case is presented in Table 9, and its message flow chart is represented in Figure 10.

- Verify if Flexoffer was respected: After a schedule of a flexoffer was executed, the user can check if the energy values of the flexoffer were respected. That is, if the measurements in power of a device for a specific period of time corresponds to the power level established. The execution flow of this use case is presented in Table 10, and its message flow chart is represented in Figure 11.
-

The previous list of features can be found depicted in the Use Case diagram in figure 3.

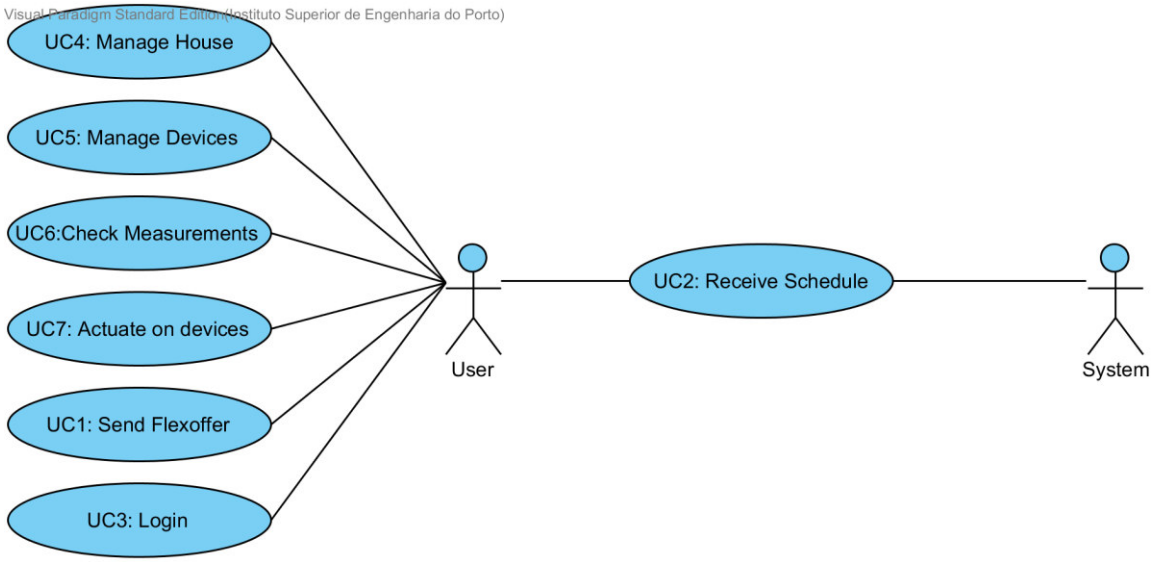

Figure 10- Use Case diagram for FlexHousing

2.2 NON-FUNCTIONAL REQUIREMENTS

Regarding the non-functional requirements, there are five that must be respected:

- Availability: The system must be online and accessible as long as possible, 24 hours per day and 365 days per year.

- Integrity: Dealing with sensible industrial requests the system must always report any execution error. Some errors may have deep impact if a flexoffer malfunctioned when it was attached to an important machine.

- Interoperability: The systems must be able to communicate independently of their implementation: they just need to use the technologies used by their corresponding services.

- Performance: The system and its requests must have the shortest execution time therefore an advanced hardware, fast internet, and good programming code should be adopted.

- Scalability: The System must support new features. Those features must be added seamlessly and must not hinder the previous ones. For a bigger management of energy, the system must be able to scale in hardware and, in repercussion, scale in performance.

2.3 USE-CASES EXECUTION FLOW

This section will approach each Use-case describing them in their execution workflow. The following section will contain sequence diagrams for each of the UC described.

Table 7 - Use Case 1 execution flow.

UC 1 - Send Flexoffer
ID: 1
Brief description: The user selects a device in order to apply a flexoffer on.
Primary actors: -FlexHousing Middleware, FlexHousing Front End
Secondary actors: -User
Preconditions: - None
Main flow: 1- The user selects the option 2- FlexHousing Middleware returns the list with the devices that are actuatable. 3- The user selects the device and inputs the information about the flexoffers 4- FlexHousing Middleware receives the information, stores it in the database and attaches the flexoffer to the device.
Post conditions: - The flexoffer was stored in the database and is now eligible to be converted to a schedule.
Alternative flows: 2*- The system may not have actuatable devices registered, returning an error stating so.

Table 8 - Use Case 2 execution flow.

UC 2 - Receive Schedule
ID: 2
Brief description: The system will execute the Flexoffer emissions towards the Aggregator. FlexHousing will then convert that Flexoffer schedule into a time triggered Actuation schedule
Primary actors: -FlexHousing Middleware, FlexHousing Front End
Secondary actors: -User -FlexofferAgent
Preconditions: - The database has flexoffers that will be applied the next day
Main flow: 1- FlexHousing Middleware starts the Flexoffer Emission process. 2- FlexHousing Middleware gathers all the Flexoffer and contacts the FlexofferAgent 3- The FlexofferAgent sends each Flexoffer to the FOServices. 4- The FOServices will attach a Flexoffer schedule to the Flexoffer. 5- The Schedule is inserted in the database. 6- An Actuation Thread is created using the information for the Schedule. 7- The User can check the Schedule attached to a certain Flexoffer
Post conditions: - The schedules are stored in the database - Actuation thread are created to control the device, based on the Flexoffer
Alternative flows: 4*- Flexoffers aren't valid. The FOServices sends back an error

Table 9 - Use Case 3 execution flow.

UC 3 - Login
ID: 3
Brief description: The user logins into FlexHousing which consequently logins in the VPS services
Primary actors: -FlexHousing Middleware, FlexHousing Front End
Secondary actors: -User -VPS Services
Preconditions: - The user must have an account registered at the VPS Services with the same credentials
Main flow: 1- User inputs the credentials 2- FlexHousing Middleware receives them and sends them to the VPS Services 3- The VPS Services send back the authorization. 4- FlexHousing Middleware stores the authorization 5- FlexHousing Front End confirms the login to the user
Post conditions: -The authorization is now stored in the system. -Any operation that involves the VPS Services will have the authorization attached to them.
Alternative flows: 3*- If the credentials aren't correct for the VPS Services, an error will be sent back instead of the authorization.

Table 10 - Use Case 4 execution flow.

UC 4 - Manage House
ID: 4
Brief description: Allows the user to do CRUD operations on the House, more specifically on the rooms.
Primary actors: FlexHousing Middleware, FlexHousing Front End
Secondary actors: User
Preconditions: - None
Main flow: 1- The user inputs the name for a new Room 2- FlexHousing Middleware receives the name, creates a new Room and stores it in the database 3- The User selects a Room 4- FlexHousing Front End shows the details of the Room, including name, House and the Devices attached to that Room 5- The User selects another Room and inputs the new information to change. 6- FlexHousing Middleware receives the information, retrieves the Room from the database, changes the details and stores it back. 7- The User selects a Room to be deleted 8- FlexHousing Middleware deletes that Room from the database.
Post conditions: From this Use Case a room can be created, deleted, updated and read.
Alternative flows: -None

Table 11 - Use Case 5 execution flow.

UC 5 - Manage Devices
ID: 5
Brief description: This allows the user to do CRUD operations on the Devices.
Primary actors: -FlexHousing Middleware, FlexHousing Front End
Secondary actors: - User -VPS Services
Preconditions: - User is logged in at VPS
Main flow: 1- The user inputs the details for a new Device 2- FlexHousing Middleware receives the date, creates a new Device, stores it in the database and relays the information to the VPS Services 3- The User selects a Device 4- FlexHousing Front End shows the details of the Device, including name and the Flexoffers and Schedules attached to that Device 5- The User selects another Device and inputs the new information to change. 6- FlexHousing receives the information, retrieves the Devices from the database, changes the details and stores it back. 7- The User selects a device to be deleted 8- FlexHousing deletes that device from the database.
Post conditions: -The Device can be created, registered, updated, read and deleted
Alternative flows: 2*- If the device is already registered at VPS Services then the User only needs to input the physical ID of the Device.

Table 12 - Use Case 6 execution flow.

UC 6 - Check Measurements
ID: 6
Brief description: The user obtains the data collected by the Sensors attached to a Device.
Primary actors: -FlexHousing Middleware, FlexHousing Front End
Secondary actors: -User -VPS Services
Preconditions: - Login done at VPS Services - Device registered in both systems
Main flow: 1- User selects a Device 2- User selects the Sensor to collect data from, as well as the period of time of the collection 3- FlexHousing Middleware sends a request to the VPS Services 4- VPS Services sends back the data 5- FlexHousing Middleware stores the data 6- FlexHousing Front End presents the data to the User
Post conditions: -The measurements are stored in the local database.
Alternative flows:

Table 13 - Use Case 7 execution flow.

UC 7 - Actuate on Device
ID: 7
Brief description: The user can turn on and off a specific device remotely.
Primary actors: -FlexHousing Middleware, FlexHousing Front End
Secondary actors: -User -VPS Services
Preconditions: - The device has to be registered. - The device has to have an Actuator sensor - The Login at the VPS has to be valid
Main flow: 1- The User selects the Device. 2- The User selects the Actuation option. 3- FlexHousing Middleware receives the request and send the request to the VPS Services. 4- VPS Services receive the request and act accordingly to the command issued.
Post conditions: - The state of the Device was changed.
Alternative flows: 4*- The command issued may not change the state of the device if it is changed to state it was already at.

Table 14 - Use Case 8 execution flow.

UC 8 - Verify if Flexoffer was respected
ID: 8
Brief description: This allows to check if the flexoffer was respected: see if the consumption of the device corresponds to energy allocated by the Flexoffer
Primary actors: -FlexHousing Middleware, FlexHousing Front End
Secondary actors: -User -Flexoffer Services -VPS Services
Preconditions: - The Schedule that is being verified needs to have been executed.
Main flow: 1- The User selects the Device 2- The User selects the Flexoffer 3- The information is sent by FlexHousing Front End to FlexHousing Middleware 4- FlexHousing Middleware selects the corresponding schedule 5- FlexHousing Middleware request the measurements for the device for the same period of time as the schedule 6- FlexHousing Front End presents the data from both origins to the user, flagging any records that violated the flexoffer.
Post conditions: -
Alternative flows:

3. DIAGRAMS

This sections contains the diagrams sequence for the Use Case presented in section 2.

3.1 SEND FLEXOFFER

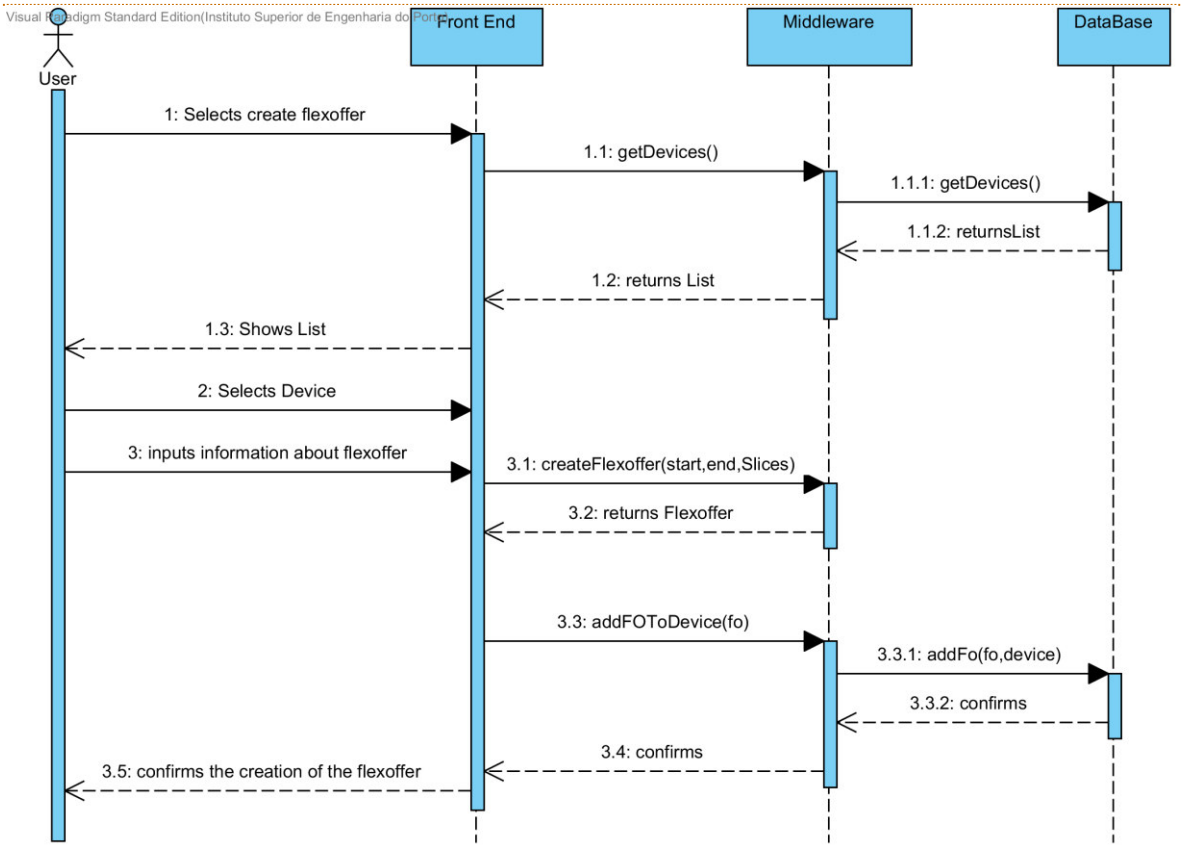

Figure 11 - Sequence Diagram of UC1.

The diagram in Figure 11, represents the workflow execution for UC 1 in which the user wants to apply a flexoffer to a specific device. The user initiates the process and the FrontEnd retrieves the device list from the Middleware. The user then picks the device he wants to apply the flexoffer on. From there, the information of the flexoffer is introduced and is sent back to the middleware. There, it is stored in the database.

3.2 RECEIVE SCHEDULE

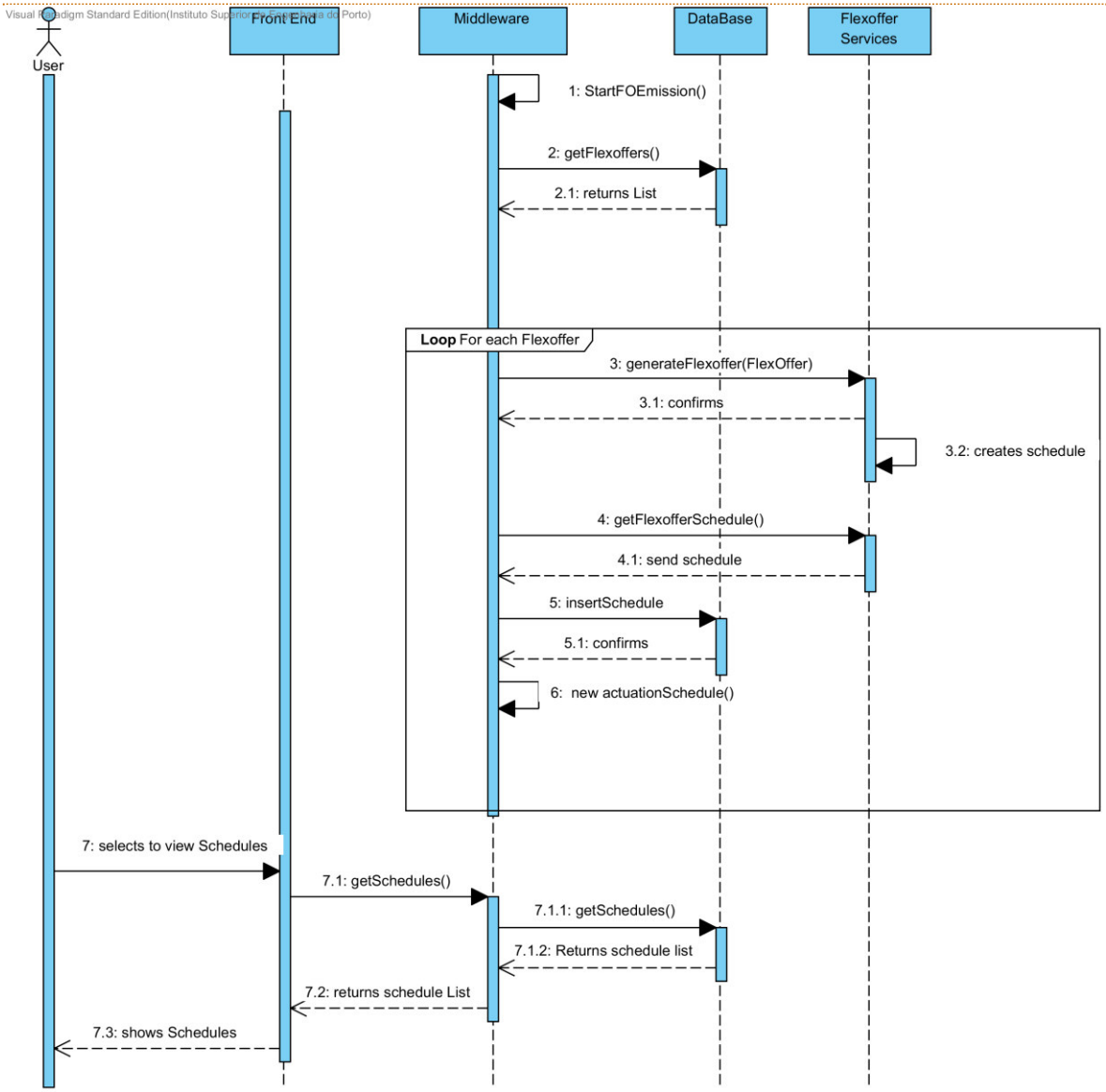

Figure 12- Sequence Diagram of UC2.

Figure 12 depicts the workflow for UC2. The diagram also depicts the execution of the pre conditions, which is the emission of the flexoffer and retrieval of its schedule. In order to retrieve the schedule, the flexoffer has to be sent. That is accomplished by contacting the flexoffer agent with the information about the flexoffer. The agent will communicate with the Flexoffers Services and proceed to the emission. The schedule is then sent back by the services and stored in the database. An actuation schedule is also created in order to automate the energy usage of the device.

3.3 LOGIN

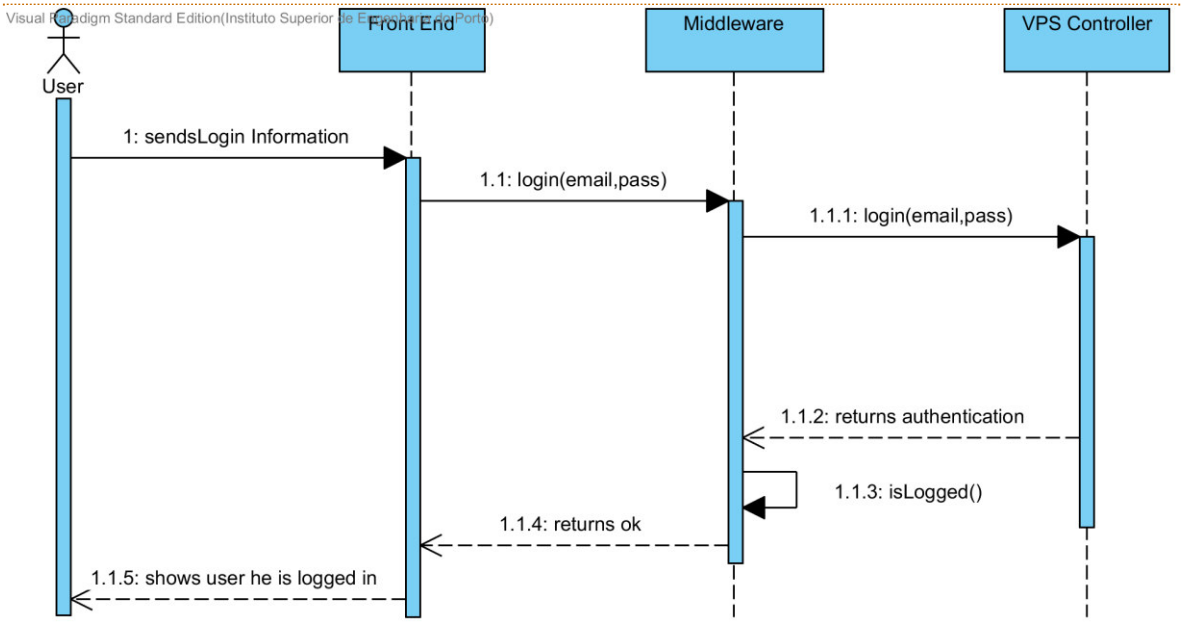

Figure 13 - Sequence Diagram of UC3.

Figure 13 evidences the steps required for the user to login into the system. The login is handled by the middleware and then is sent to the VPS services. When the response is retrieved, the user is officially logged in.

3.4 MANAGE HOUSE

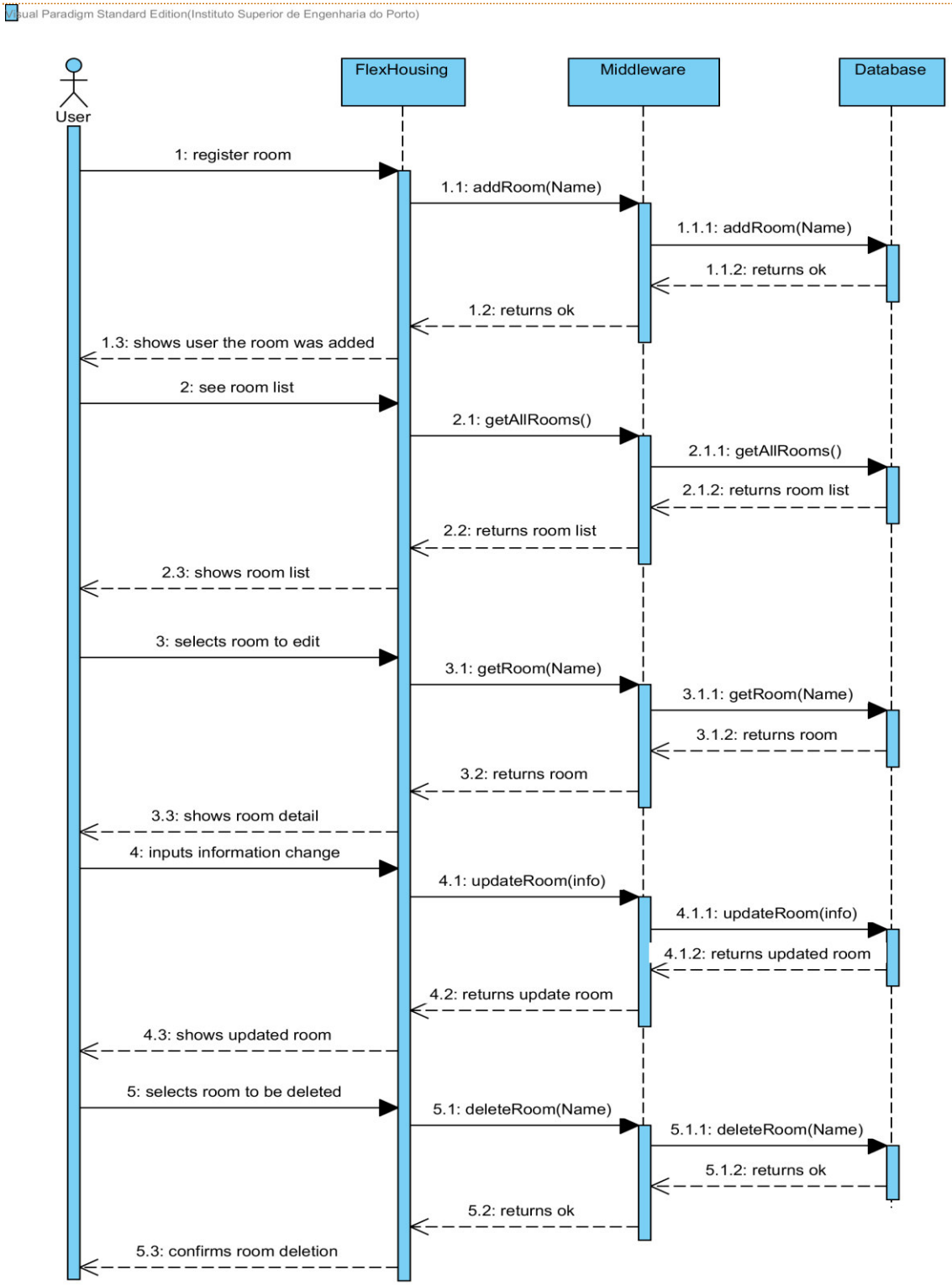

Figure 14 - Sequence Diagram of UC4.

Figure 14 describes all the possible operations the user is able to do related to his house/building. Each operation is handled by the middleware has repercussions on the database.

3.5 MANAGE DEVICES

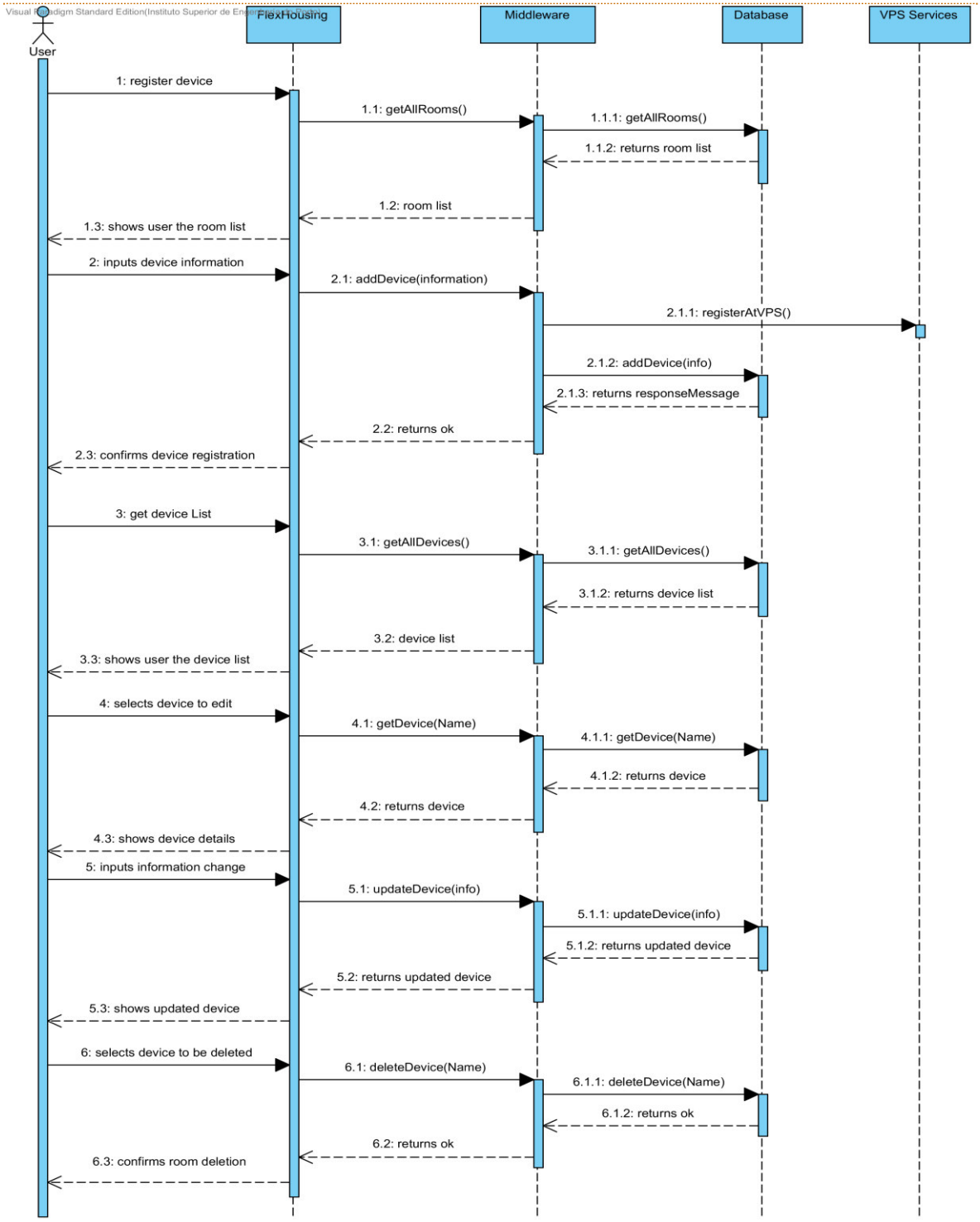

Figure 15 - Sequence Diagram of UC5

Similar to Figure 15, the diagram in Figure 8 describes the possible operations but in this case towards the devices. Again, the requests are handled by the middleware and the operation are persisted in the database

3.6 CHECK MEASUREMENTS

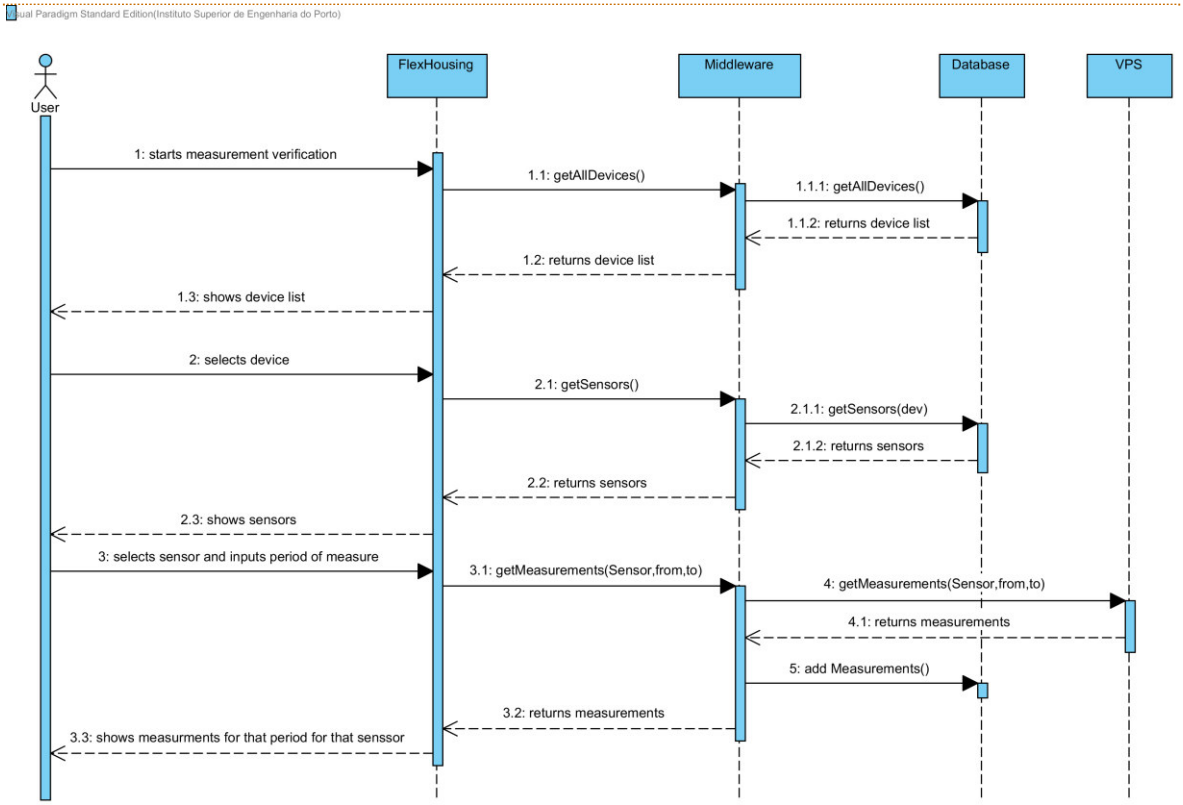

Figure 16 - Sequence Diagram of UC6.

Figure 16 depicts the steps the system goes through for the retrieval of measurements. Again the systems retrieves the device list to allow the user to choose the device to query. The User inputs the parameters for the search. The request is received by the middleware which proceeds to query the VPS services for the info the user wants to obtain. When the response is received, the measurements are presented to the user.

3.7 ACTUATE ON DEVICES

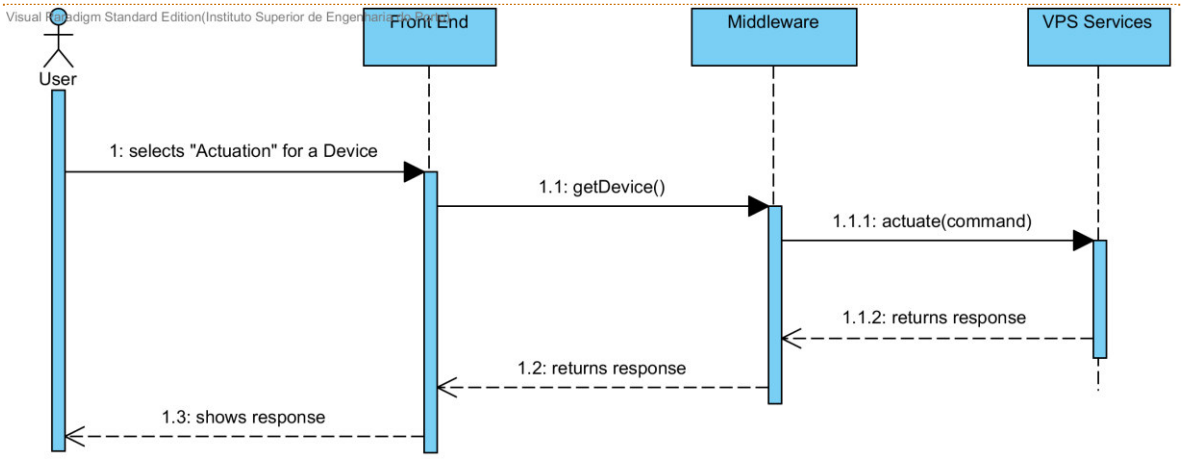

Figure 17 - Sequence Diagram of UC7.

In Figure 17 is depicted the sequence diagram for the actuation on a device. The middleware, once again, handles the request. The middleware will then execute a request to actuate on the device the user selected towards the VPS services. When the request is processed by them, the actuation will occur.

3.8 VERIFY IF FLEXOFFER WAS RESPECTED

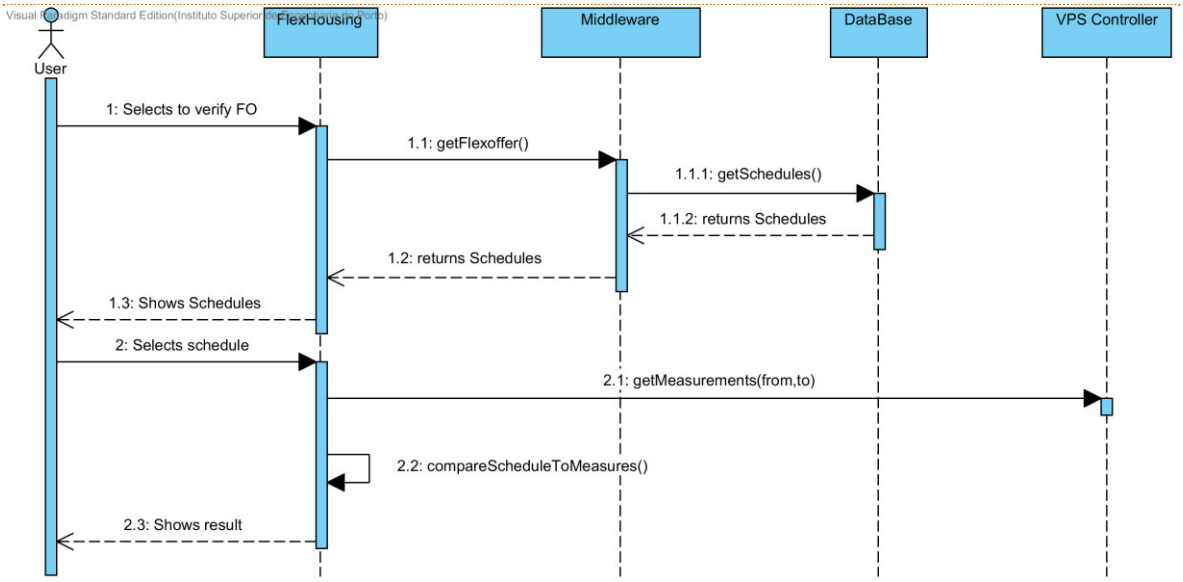

Figure 18 - Sequence Diagram of UC8.

Figure 18 shows the steps for the verification to whether a flexoffer was respected. The schedule is cross checked with the measurements that were gathered in that same period. With that information, this system is able to detect any violation of the pattern the flexoffer had.

4. SECURITY

This section defines high-level security principles the system needs to follow on a non-technical, generic level.

4.1 SECURITY OBJECTIVES

FlexHousing will have two states on both its systems: logged and non-logged. The transition is done when the login is done at the VPS Services.

For the communication between FlexHousing Middleware and the Flexoffer services, the Middleware has the credentials in order to create a session for the exchanges of messages. When the link is established, every request is exchange in that session.

Any operation or exchange with the VPS services will fail if they aren't accompanied with a valid token, the same for the authentication between the Middleware and FrontEnd.

4.2.2 SYSTEM-OF-SYSTEMS DESIGN DESCRIPTION

This document defines the System-of-Systems Design Description of the FlexHousing Pilot. It essentially complements the System-of-Systems Description with implementation details

1. OVERVIEW

The implementation of the FlexHousing pilot is built around 3 main components, as shown in Figure 1:

- A local environment with both FlexHousing systems (FlexHousing Front end FHFE and FlexHousing Middleware FHMW), the gateway("Cloogy") and a smart plug
- The Flexoffer services running on a server in Denmark
- The VPS services

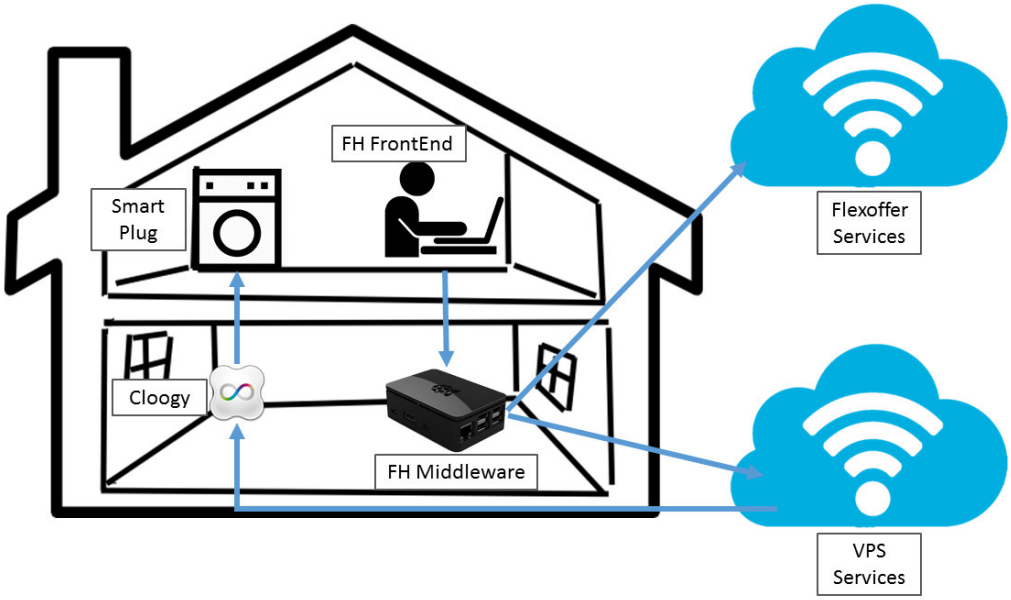

Figure 19 - FlexHousing System overview

The Middleware FHME is being executed on a Windows environment as well as the front end FHFE. In the same local installation, there is the Cloogy (the gateway from VPS) and a VPS smart plug. The FHFE receives the inputs from the user, which translate in to the emission of request towards the FHMW. From there on, it can lead to 3 execution paths:

- The FHMW communicates with VPS services in order to interact with devices
- The FHMW connects with Flexoffer Services for operations related with flexoffers
- The FHMW receives a request that is related to the data structures local to the FHMW, such as the rooms

2. SYSTEMS

Table 15 - Pointers for the systems in play

System name	Path
Middleware	SysDD FlexHousing-Middleware
FrontEnd	SysDD FlexHousing-FrontEnd
Flexoffer Services	Arrowhead\Meetings\Multi WP Workshops \2013-11-05\ Porto\Documenting\Examples \SysDD\Arrowhead SysDD Aggregator v0.1.docx
VPS Services	[Legacy System]

2.1 MIDDLEWARE

The FlexHousing Middleware (FHMW) is responsible for the major connectivity of the pilot: it sends the flexoffers to the Flexoffer services, sends the requests for the operations on the VPS devices, exposes its services for the FlexHousing FrontEnd (FHFE) and manages the Database. The FHMW also does its own implementation of a DER object (see [1]) for the emission of flexoffer and reception of schedules.

2.2 FRONTEND

It acts as a layer between the user and the middleware FHMW. It allows for all the operations the middleware is capable of. It also provides user friendly graphs for the representation of more complex or abstract objects such as the flexoffers, schedules or collections of measurements.

2.3 FLEXOFFER SERVICES

The FHMW interacts with an instance of the Flexoffer system, through the Flexoffer services it exposes. The instance of this Arrowhead system is currently the one deployed in Alborg, Denmark. The system was developed by other partners of the Arrowhead project. It is responsible for the reception of flexoffers, compilation of schedules and their respective delivery to the Flexoffer Agents. The communication with the Flexoffer system is done over a XMPP server hosted by the system.

2.4 VPS SERVICES

The details of these services are protected by a Non-Disclosure Agreement. These services bridge the FHMW and the devices. The FHMW sends requests to the VPS Services, which exposes an API. From there on, the server hosting the VPS services communicates with the Cloogy gateway through the internet. The Cloogy gateway is located in the house/building of the user, and it sends requests to the devices to drive their operations.

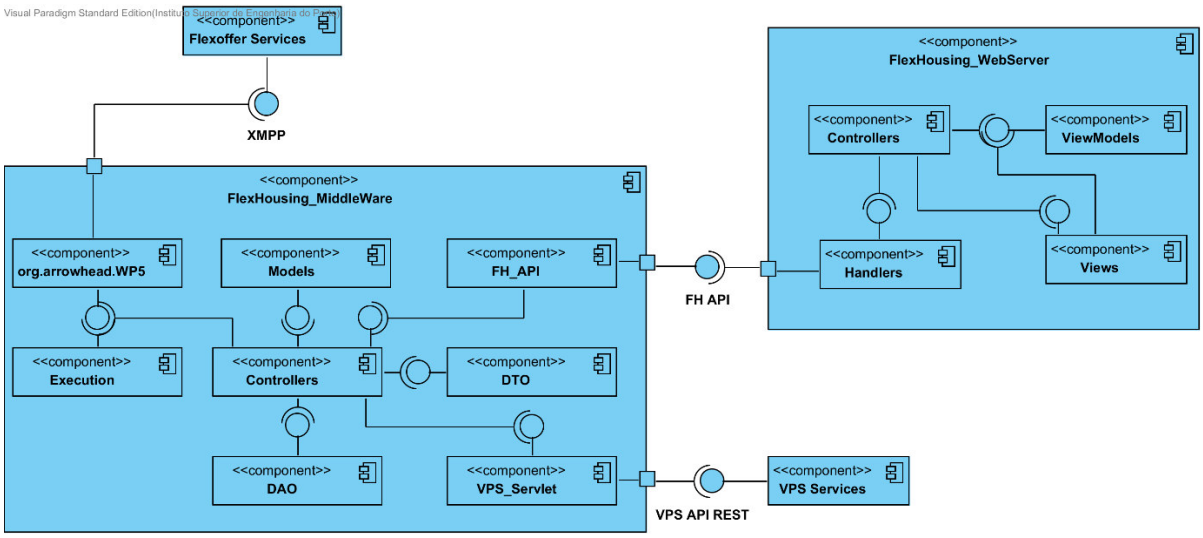

Figure 20- Components Diagram for the FlexHousing Pilot

In Figure 20 is depicted the components diagram for FlexHousing. The Middleware will be hosting most of the services and logic of the system. It will be connected to the VPS and Flexoffer services. Finally, the FrontEnd system will consume the services provided by the FH_API interface provided by the Middleware

3. NON-FUNCTIONAL REQUIREMENTS REALIZATION

-Availability: The system is available 24/7. The FHMW feeds the flexoffers for the following day at 23h00 each day, to receive the corresponding schedules. Thus, flexoffers received between 23h00 and 24h00 of each day cannot be applied for the energy schedule of the following day.
-Integrity: The system handles all information the same way; Any action or change on any object or instance is automatically persisted in the database owned by the FHMW.
-Interoperability: All systems communicate using protocols that are implementable on virtually any platform or coding language. Any FlexHousing system can be replaced with a different implementation and still be able to communicate with the other system as long as it respects the protocols in use.
-The Flexoffer services can handle over 100 000 flexoffers per day. It was not possible to perform any other stress test, since the available hardware was limited to just one Cloogy gateway and one VPS smart plug.

4. SECURITY OBJECTIVES

FlexHousing has two states on both its systems: logged and non-logged. The transition is done when the login is done at the VPS Services. Most operations for the FHMW aren't feasible if the login isn't done previously. Through that operation, an authorization token is generated. The FHMW holds on to the token and attaches it to any operations involving the VPS services, to authorize them.

Regarding the communication between FlexHousing and the Flexoffer services, the FHMW is configured with Arrowhead credentials in order to create a session using XMPP. In the XMPP terminology, the FHMW enters a room that is created by the Flexoffer services, and every exchange is done inside the room, creating a secure environment.

5. WORKFLOW OF USAGE

The user logs in, using the credentials related to the VPS services, to connect to the devices, located in the user house/building and already registered in the VPS services. Then the user is presented with the list of devices he can use. From there on, he can collect the measurements from the "Active Power" sensor that all devices have, which will be referred later as "the energetic profile of the devices". With that information, the user can have an idea of what the regular consumption of each devices is. Then the user can apply a flexoffer to that specific device in two ways:

- The measurements taken earlier can be employed in order to create a flexoffer whose pattern of energy usage corresponds to the energetic profile of that device.
- the user can create the flexoffer from scratch:

Then the user inputs remaining information for the setup of the flexoffer (name, start time, end time). From there on, the FHMW will receive and store the information of the flexoffer. At a specific time (Currently, at 11pm), the middleware initiates the flexoffer emission process: it gathers all the flexoffers that are active for all the devices, starts a XMPP connection with the Flexoffer Services and sends them. Each time a schedule is received, it is put in the database. Before midnight, the FHMW retrieves all the schedules for each device and executes a threadpool of actuation schedules, which will drive the execution of the energy schedules. Every 15m, the actuation schedule verifies if an appliance should consume energy, and it will switch it on/off accordingly. Such is accomplished by executing a request at the VPS services. Figure 21 depicts the steps described previously.

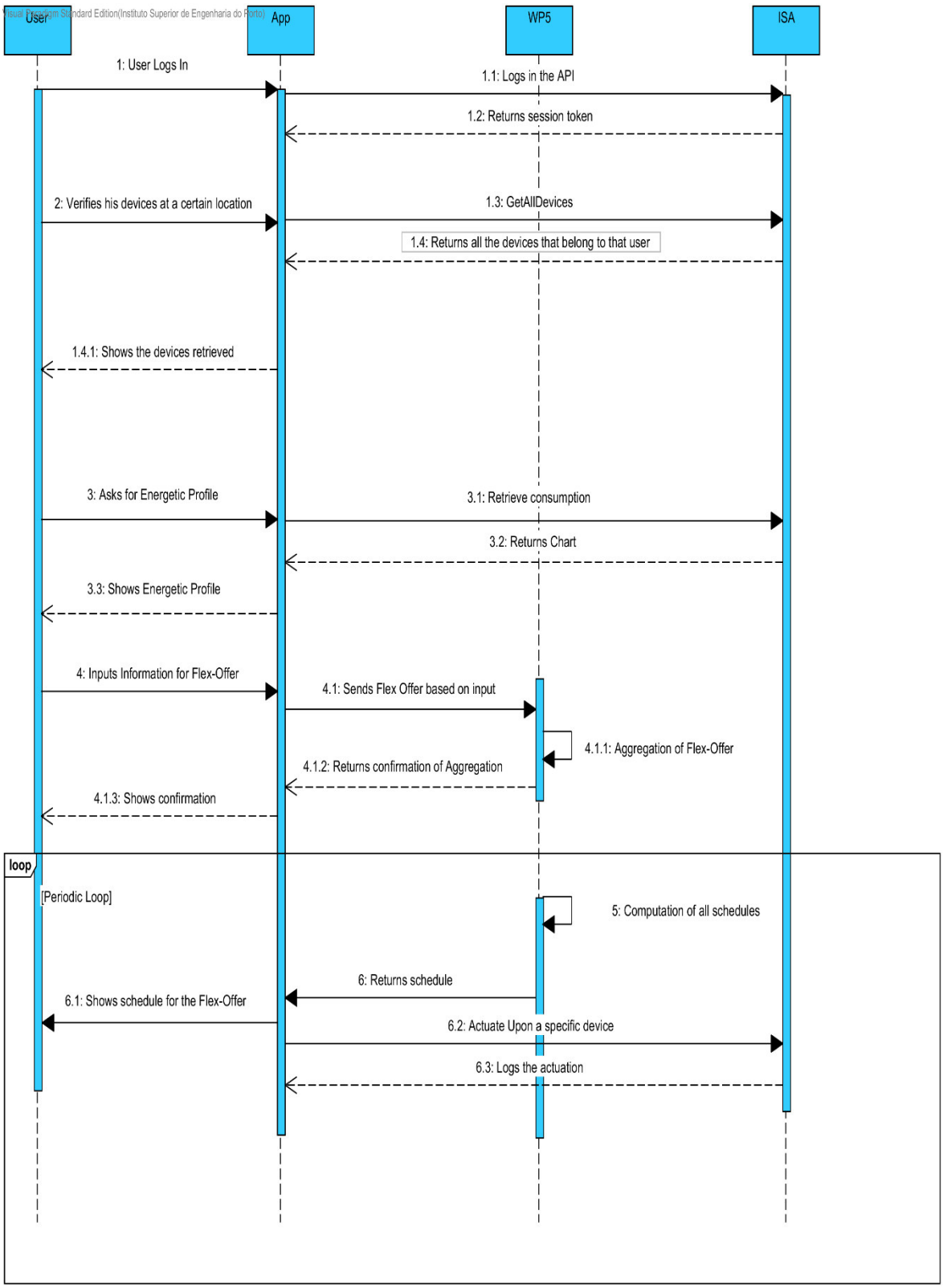

Figure 21- Sequence diagram for the typical usage of the system

6. USE CASE IMPLEMENTATION

The following figures depicts the implementation of some of the UC described in the SoSD.
This particular page (Figure 22) allows for the creation of the flexoffer. The user inputs the name, start and end time, and the pattern of the consumption. The user also as the possibility of exporting a pattern from the measurements of energy.

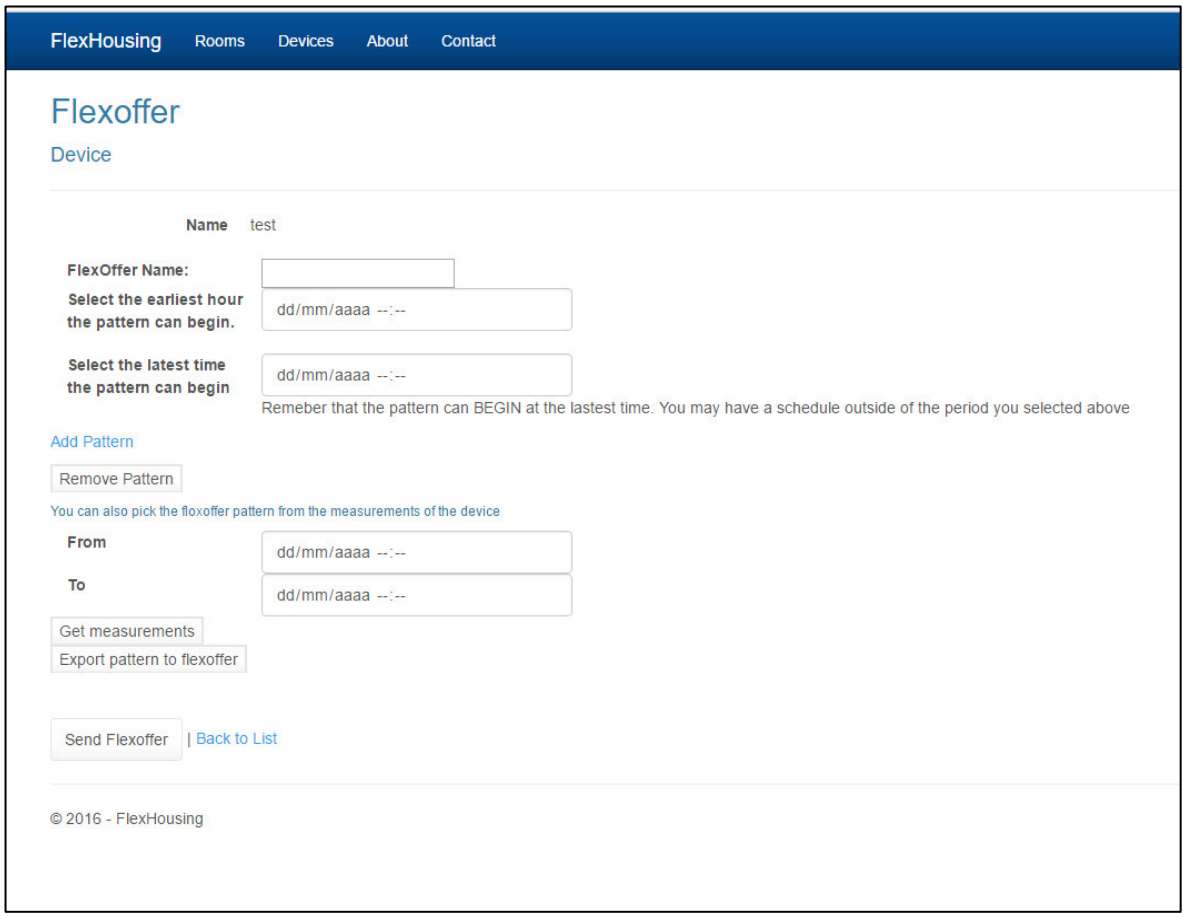

Figure 22- Creation of flexoffer screenshot

The following figure (figure23) regards the index view for the Room. It allows for the creation of new rooms and the inspection of the details.

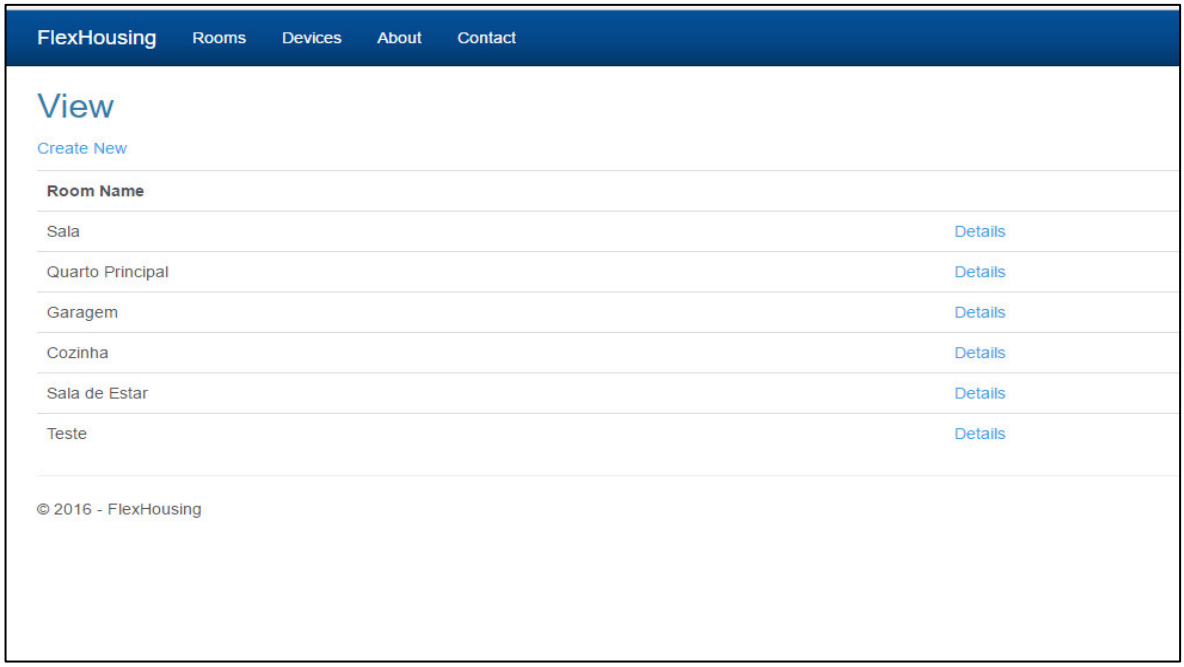

Figure 23- Managing the rooms screenshot

The following screen shot (Figure24) depicts the list of devices with the various options for each.

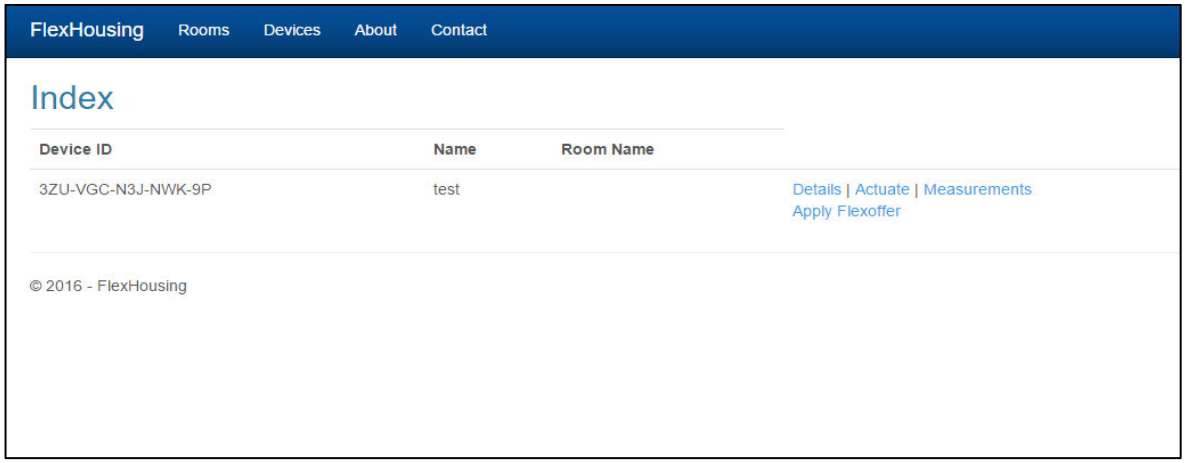

Figure 24- Device list screenshot

This figure shows the options of receiving measurements for a certain device

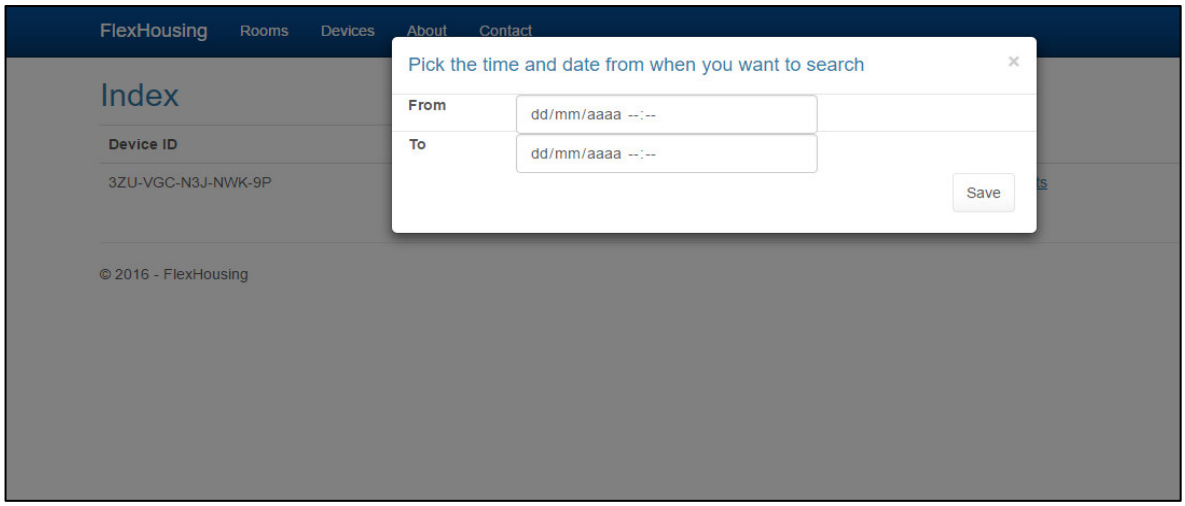

Figure 25- Screenshot of the 1st step to retrieve measurements

4.2.3 SYSTEM DESIGN – MIDDLEWARE

This document provides the System Description for the for the FlexHousing Middleware (FHMW) system. It provides a high level overview of the FHMW module

1. SYSTEM OVERVIEW

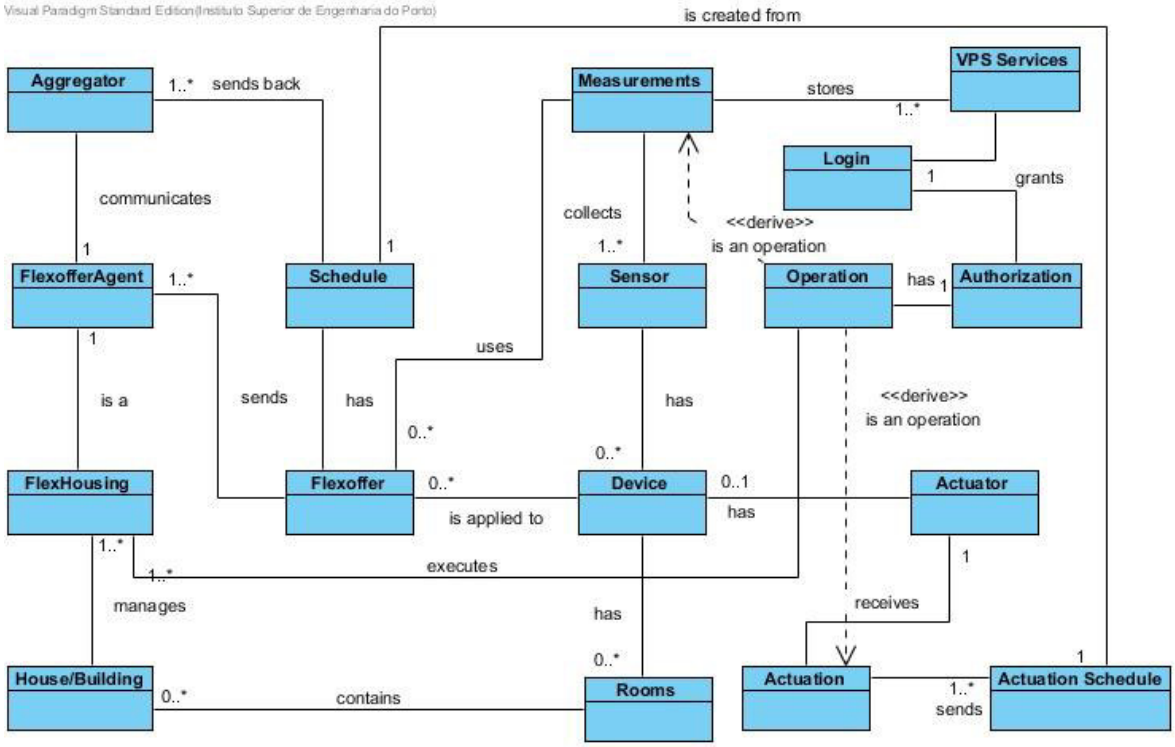

Figure 26 - Domain Model of the FlexHousing system.

In figure 26 is the domain model for `FlexHousing` system. The core of the system revolves around the `Device`, which is an abstraction for the smart plugs that can be deployed in a house to control and measure the energy consumption of appliances. In order to allow for sorting and traceability, the devices are encapsulated in `Rooms`. `FlexHousing` middleware is responsible to bridge the house, rooms and devices, with the `FlexofferAgent`, who is responsible to generate and send the `Flexoffers` to the `Aggregator`. In response, the `Aggregator` will send back `Schedules`. Each device has `Sensors` to collect `Measurements`, which get stored in the `VPS Services`, and `Actuators`, which can switch on/off the appliances connected to the `Device` and are controlled through the `VPS Services`. `FlexHousing` middleware executes `Operations` to either gather the `Measurements` or to feed `Actuations` to the `VPS Services`. `Actuations` will trigger the `Actuator` of the `Device` into turning off and on the appliance connected to the `Device`. But to do the said `Operations`, a `Login` is required at the `VPS Services` which will grants an `Authorization` that is attached to the `Operations` in order for them to be executed. All of this allows for the creation of `Actuation Schedules` from the flexoffer `Schedules`. From there on, the system will execute `Actuations` for the respect of the times and energy values. In order to make the

`Flexoffer` close to reality as possible, the `Flexoffer` may use `Measurements` to correctly establish its energy values for each period of time.

2.USE CASES

The FHMW system is part of the FHSoS (see SoSD document) and it aims to satisfy the requests it receives from the FHFE. As such, the Use cases of the FHMW correspond to the ones described regarding the System of Systems.

2.1 FUNCTIONAL REQUIREMENTS

This system provides 4 interfaces: each manages or deals with a major type of objects or instances: Flexoffer, Device, House and Measurements. Every service offered by FHMW is Arrowhead compliant.

2.2 NON-FUNCTIONAL REQUIREMENTS

Regarding the non-functional requirements, the following are needed:

- Availability: The system must be online and accessible as long as possible, 24 hours per day and 365 days per year.

- Integrity: This system is dealing with sensitive information that may in jeopardize the user's home/building

- Interoperability: New systems have to be able to communicate with the middleware. This means the technology it uses has to be widespread and adaptable.

- Performance: The system and its requests must have the shortest execution time therefore an advanced hardware, fast internet, and good programming code should be adopted.

- Scalability: The System must support new features. Those features must be added seamlessly and must not hinder the previous ones. For a bigger management of energy, the system must be able to scale in hardware and, in repercussion, scale in performance.

2.3 USE-CASE EXECUTION FLOW

The middleware is driven by means of requests. Each request leads to a response. If the request aims to retrieve data, then it will be met with the communication of the data requested. On the other hand, if the request is to register or to alter an item, if no exception was raised, the request will be met with a code 200 response. Most operations will require that the user is logged onto the system. Every change of state, object or stored data is automatically persisted in the database. Figure 27 depicts the workflow described.

Table 16 - Workflow execution for the handling of requests

UC 1 - Respond to request
ID: 1
Brief description: The system will respond accordingly to the request received
Primary actors: -FlexHousing Middleware
Secondary actors: -Requesting system
Preconditions: - It most case, requires the system to be in a logged state.
Main flow: 5- The Middleware receives a request 6- The request is handle by the appropriate controller 7- The controller executes the logic for the operation 8- The controller retrieves the data from database 9- A DTO is built to send back the information 10- The service replies with the requested information
Post conditions: -
Alternative flows: 4*- The system inserts the sent data into database. 5*- A response with a code is built and sent, accordingly to the result of the operation

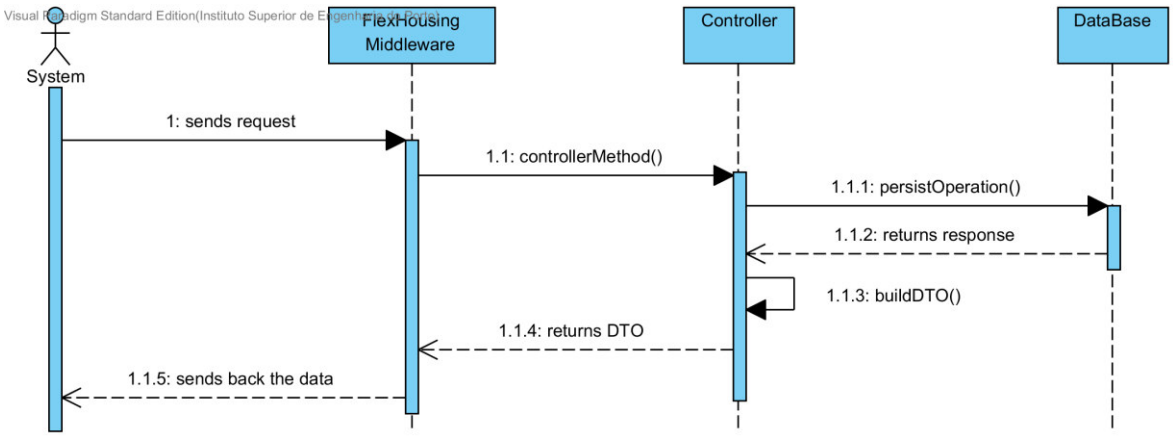

Figure 27- Normal execution of a request response

3. APLICATION SERVICES

Being a distributed system and also being a middleware, FHMW exists between other systems: it consumes services offered by other system and offers services of its own. In this case, as Figure 28 depicts, FHMW offers 4 interfaces while consuming from 2.

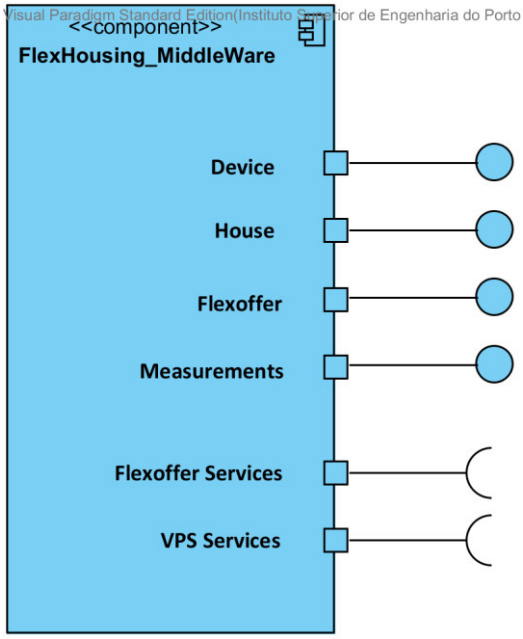

Figure 28 – Component diagram for the FHMW interfaces

2.1 PRODUCED SERVICES

The middleware provides 4 interfaces, each with its own functions. The Semantic profile and Interface Design Description are both included in the IDD for the middleware. These interfaces are used when an external entity needs to interact with FHMW. More details are provided in section 4.2.3.

FlexHousing Interface	SP-IDD FlexHousing Middleware

2.2 CONSUMED SERVICES

As the name suggests, the FlexHousing middleware bridges 2 major systems: Flexoffer services and VPS services.

Flexoffer Services	…\Arrowhead\Meetings\Multi WP Workshops \2013-11-05\ Porto\Documenting\Examples \SysDD\Arrowhead SysDD Aggregator_v0.1.docx
VPS Services	[Legacy System]

4.2.4 SYSTEM DESIGN – FRONTEND

This document provides a high level overview of the FlexHousing FrontEnd system. It will approach the domain model for the FHFE as well as the usage of services by it employed.

1. SYSTEM OVERVIEW

The FlexHousing FrontEnd (FHFE) system allows the user to configure the settings and manage his house using services hosted on the middleware FHMW. FHFE offers the option of organizing his appliances and devices into buildings/houses and rooms. Using the FrontEnd, the user can manage his house, checks on the various flexoffers and corresponding schedules he has on his devices, and analyze the data collected from the sensors located on his devices.

The goal of this system is to offer a user friendly graphical interface that connects the user to its devices and offers a simple way to exploit the flexoffer capabilities. Through the usage of charts, the flexoffer are depicted and information is conveyed to the user.

On the first approach, the user might get overwhelmed by the customization options, but after a short learning curve, the system will feel rather easy and intuitive.

1.1 DOMAIN MODEL

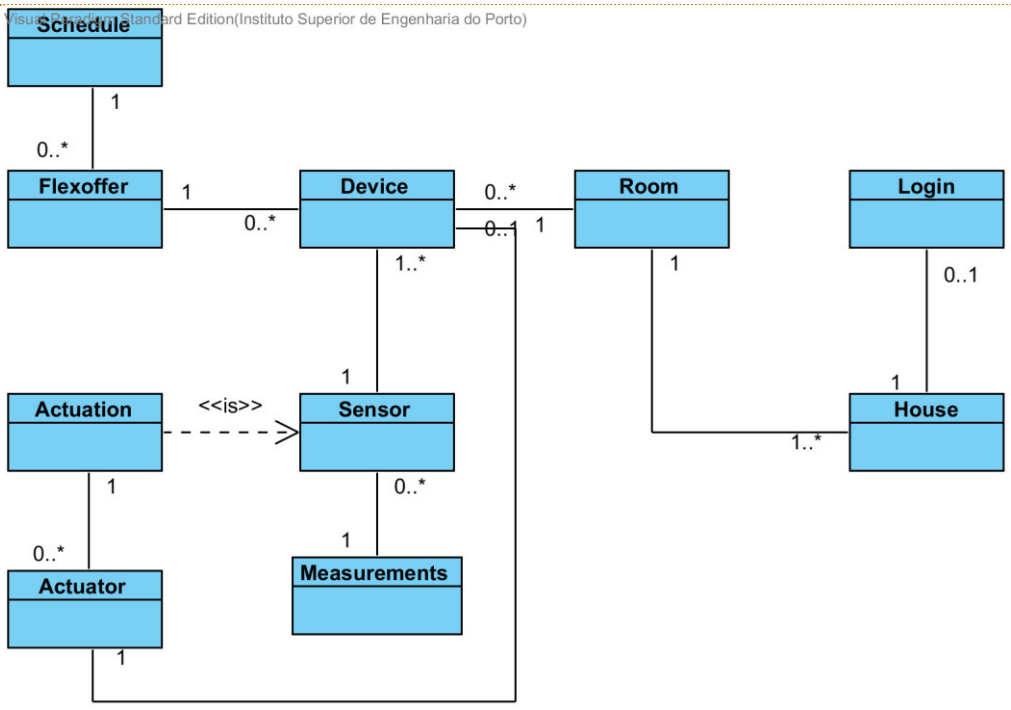

Figure 29 - Domain Model of FlexHousing FrontEnd system.

The main building block of the system is the `Device`. The device is connected to an energy-consuming appliance, and the device belongs in a `Room` and that room to a `House`. Each device has `Sensors`, which are the generic sensor/actuator elements. In the model, the `Sensor` is further refined (sub classed) into an `Actuator` when it has the capability to actuate. Each sensor is able to collect `Measurements`. The actuator sensor will produce `Actuations` who are

responsible for toggling the energy state of the device. A `Flexoffer` is applied to a device. Many flexoffers can be applied to a device and every single flexoffer will lead to the creation of a `Schedule`. Figure 29 depicts the domain model for the FHFE.

2. USE-CASES

This FHFE Systems shares the same UC as the System of System it is inserted into. For further description, please refer to the SoS Description and SoS Design Description documents.
Similar to the FHMW, the FHFE has 2 states: Logged and Non-Logged.
The FHFE starts as Non-Logged, and it has very limited available features. All the FHFE features can be used when the system is in the Logged state. The transition between states occurs when the user logs in within the FHFE.

3. APPLICATION SERVICES

FHFE doesn't provide any services but, in order to interact with the FlexHousing Middleware, consumes the services it provides. Figure 30 the previous description.

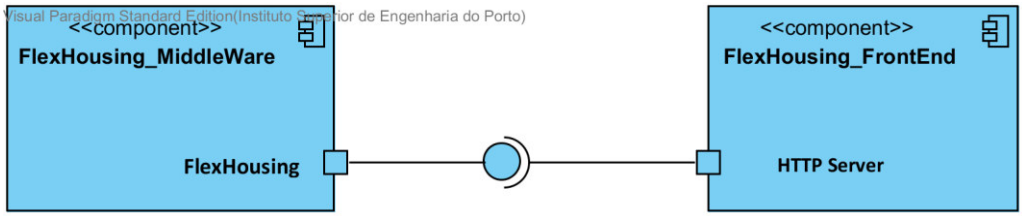

Figure 30 - Component diagram for the FHFE interface

3.1. PRODUCED SERVICES

FHFE doesn't provide any services in the Arrowhead sense, due to the fact that it is a front end used only in conjunction with an instance of the FHMW.

3.2. CONSUMED SERVICES

The FHFE only consumes services provided by the FHMW.

System name	Path
Middleware	`SysDD FlexHousing-Middleware`

4. SECURITY OBJECTIVES

The communication between FHFE and FHMW is not encrypted, but most operations will fail if the FHFE doesn't provide valid login information.

4.2.5 SYSTEM DESIGN DESCRIPTION – MIDDLEWARE

This document defines the System Design Description of the FlexHousing System. FlexHousing Middleware, as the name describes, is a middleware connecting the Flexoffers Services and the VPS Services while still providing an interface for user interaction. The focus of this document is to describe the implementation of the system architecture, the used components, and the use-cases supported by UML diagram.

1. SYSTEM DESIGN DESCRIPTION OVERVIEW

Name	FlexHousing Middleware
Owner	ISEP

The FlexHousing Middleware system has been developed by CISTER/ISEP. The system is part of the Flexoffer Pilot implement as part of the Work Package 5 of the Arrowhead Project. The middleware is the central part of pilot, bridging the Flexoffer Services and the Virtual Power Solutions Services. Virtual Power Solutions, or VPS, provide devices smart plugs and smart meters, allowing for remote access through the emission of request at their external server.
The middleware consumes services provides by the VPS Services and communicates with the Flexoffer Services using dedicated XMPP rooms. This system also provides an API for the configuration and usage of the services hosted.

System name	**Path**
FlexHousing Middleware	`Arrowhead/SysD FlexHousing Middleware`

This system is implemented using Java, in order to stay similar to the implementation of the Flexoffer Services and to minimize communication conflicts.

This documents lacks the Use Case section due to the fact that most of the services this system provides aren't Use Cases themselves, but are actually an essential part of the Use Case of the SoS (System of Systems) Pilot. The description of each services can found in the SD/IDD document

System name	**Path**
FlexHousing Middleware SD-IDD	`Arrowhead/SD-IDD FlexHousing Middleware`

This document (found later in this chapter) provides the sequence diagram and description of each service as well as an example for each request.

2. DECOMPOSITION OF THE SYSTEM

The system is comprised of 3 communication modules:

- The VPS Services: Request sent towards the VPS Servers need to be authenticated by using a header containing a token, obtained by login in with valid credentials. Table 17 depicts an example of response to a successful login

Table 17 - Example for a valid request for a token

```
<Session
xmlns="http://schemas.datacontract.org/2004/07/VPS.iEnergy.Entities.Sessions"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<RefreshToken>6V5xjGuoESL06jK0CuI6d/xbN25DRS4NIMrOTA1BMLbFI7sZnCyQPVJ+B0
7/5Ju25qgNGJDu4casNkGpqqwDsgaRQ+FxYPXAqtGuh0Relsej2O/ekc1erzcq</RefreshTo
ken>
  <Timeout>7200</Timeout>

<Token>wR1tD9G7QexviwwKZEqeADY+0H1astb5nESqx2NuXgz064HUebPzuzmtrquPDx
Ja1o0NESHhzFqr5VQCH9gnxzRkwnkoUkGYc2UMfmDQZ2G3CiywGa2QVHQO</Token>
</Session>
```

- The Flexoffers services: The exchange of messages is done inside a XMPP room which is hosted by the services. The Middleware has the correct credentials for the joining of the room. Table 18 depicts part of the configuration of the middleware in regard to the FO services

Table 18 - Code sample for the XMPP credentials

```
/**
   * The id of this flexoffer manager. Used both for the XMPP network and the
   * uniqueness of flex offers
   */
     private String id = "isepagg";


  /** The id of the aggregator found by using Arrowhead service discovery **/
  private String aggId = "aggregator-test-tool";
```

- The FlexHousing API: The communication is not encrypted but the middleware system has 2 states: Logged and Non-Logged. The Non-Logged state will make most operations fail. Once The middleware executes the VPS login then it transits to the logged state and enables the full usage of the services.

-

3. SOLUTION DESCRIPTION

The purpose of this section is to describe the implementation of the solution. Along this section, an overview of the system architecture is described with the support of a component diagram, in which each component is explained along with a class diagram of the core classes. One of the core functions of the middleware is the automatic emission of flexoffers and creation of actuation schedules. Finally, the database which the system works with, are explained with the support of a database model diagram.

3.1. COMPONENT DIAGRAM

Figure 31 depicts the organization of this system for a logical point of view. Also provides a representation of the communication between components. The Controller employ most of the other components and are only used by the services hosted by the API. The DAO controls all the operations for the data while the Execution has the logic for the automated mechanism. The or.arrowhead.wp5 and VPS_Servlet are responsible for the communication with the external services.

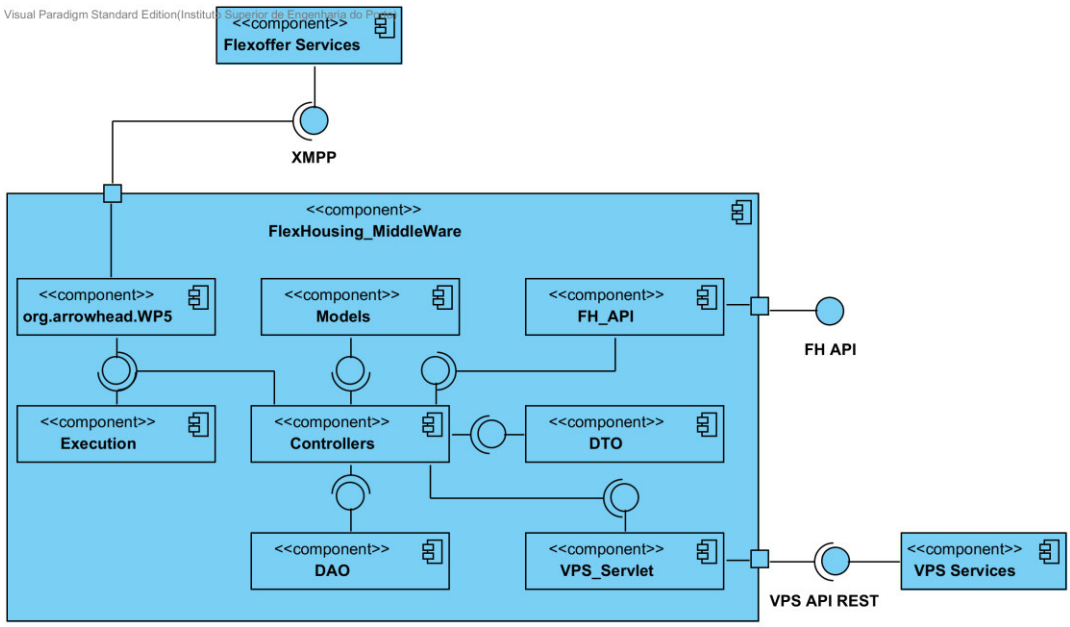

Figure 31 - Component Diagram of the FlexHousing Middleware system.

3.2 COMPONENT DESCRIPTION

This section describes each component, providing an enumeration of each Class belonging to it.

3.2.1 MODELS

The Models component contains the information about the domain objects. It supplies the controller with the entities used within the system. A description of each class can be found in table 19.

Table 19 - Class for the Models Component

Class Name	Description
Device	Represents the smart plugs. Always linked to a Room
Schedule	Abstract class for the actuation schedule
InfraDaySchedule	Implementation of the Schedule for same day flexoffers
NextDaySchedule	Implementation of the Schedule for periodic flexoffers
Room	Represents the rooms of the House
Sensor	Representation of the sensors attached to a Device
Measurements	Entity representing the data collected from the Sensors

3.2.2 ORG.ARROWHEAD.WP5

The WP5 package, short for org.arrowhead.wp5, has the implementation of DER for this system. WP5's responsibilities are the emission of flexoffers, retrieval of schedules and the connection to the Flexoffer Services. MyFlexibleResource is a singleton in order to make sure that the emission of the flexoffer originates for the same agent. A description of each class can be found in table 20.

Table 20 - Class for the Arrowhead Component

Class Name	Description
HouseDER	The House DER is an implementation of the Abstract DER. It contains implementation of the `generateFlexoffer()` functions, tailored to this system.
MyFlexibleResource	Entity responsible for the connection to the Flexoffer Services. Has methods for the XMPP connection and the Service Discovery for the Aggregator.

3.2.3 CONTROLLERS

The Controller component contains the definition of every controller of the system. The controllers are responsible for the usage of the other components of the system in order to implement the logic. The House Controller contains a cached version for the objects with the most usage. When an object suffers a change, both the cache and the database are altered. The HouseController is therefore a Singleton, to avoid multiple instances of the same objects. A description of each class can be found in table 21.

Table 21 - Class for the Controller Component

Class Name	Description
DeviceController	Responsible for every action on a Device object.
HouseController	Contains a list of every Room and Device. Allows for the managing of the house. Has the method to login at the VPS Services

3.2.4 EXECUTION

The Execution component contains all the implementation of every automatic process of system. Also responsible for the setup of every communication component (FH API, org.arrowhead.wp5 and VPS servlet). Both timer apply the Threadpool pattern. A description of each class can be found in table 22.

Table 22 - Class for the Execution Component

Class Name	Description
Main	Class executed when initiating the system. Starting point for every component
FlexofferTimer	Class responsible for keep track of the time of day for the emission of flexoffers.
ExecuteFOEmission	A runnable thread created for every flexoffer. Responsible for emitting the flexoffer, retrieve the schedules and their respective persistence.
ActuationTimer	Class responsible for keep track of the time of day for creation of the actuations schedules.
ExecuteActuations	A runnable thread for the creation of the actuation schedules. Retrieves the schedule for the flexoffers and monitors the energy usage programed for the device

3.2.5 DAO

The DAO component allows for a layer between the database and the system. Responsible for interacting with database, by executing queries. The DAO is a singleton in order insure concurrence and that there's only one connection to the data at any given moment. A description of each class can be found in table 23.

Table 23 - Class for the DAO Component

Class Name	Description
DAO	Has every operation related to database. Responsible for receiving objects and store them in the database. Also retrieves objects from database.

3.2.6 VPS SERVLET

This component is responsible for the communication between this system and the VPS Services. For the connection to be valid an authorization token is attached to every request. As such this implements the Singleton pattern, thus enforcing that there's only one valid connection to the VPS Services, at any given time. A description of each class can be found in table 24.

Table 24 - Class for the VPS_Servlet Component

Class Name	Description
VPSController	Contains the definition for every VPS related request. Builds http request aiming for the VPS API and also handles the responses.

3.2.7 DTO

This component acts a layer between the domain objects and the API. It creates representational objects, originating from the ones in the Models Component, but only containing relevant information for the operation it was requested for. A description of each class can be found in table 25.

Table 25 - Class for the DTO Component

Class Name	Description
ActuationFH	Used to demand the actuation on a given device
ActuationVPS	Object used by the VPS services for the actuations on devices
FlexofferDTO	Representation of the flexoffer only containing the fields configurable by the end user
LoginSession	Object used by the VPS Services in order to execute the login
MeasurementsDTO	Represents the relevant information gathered from the Measurements received after a request for such at the VPS Services
NewDeviceDTO	Object containing the physical ID of the VPS device
ScheduleDTO	Contains the fields of the schedules that are relevant to the end user
SensorDTO	Used when a request evolving the sensor is received. Contains the sensor name and ID
Statistics	Object containing the statistical information of the system.

3.2.8 FH API

The FH API component contains the definition of the services hosted by system. Used by the http server that FlexHousing Middleware creates. Helps creates the routing of the API. A description of each class can be found in table 26.

Table 26 - Class for the FH_API Component

Class Name	Description
DevicePath	Definition of the services attached to the Device interface
FlexofferPath	Definition of the services attached to the Flexoffer interface
HousePath	Definition of the services attached to the House interface
MeasurementsPath	Definition of the services attached to the Measurement interface

3.3 EMISSION OF FLEXOFFERS AND ACTUATION SCHEDULES CREATION

The whole system centers around enforcing the flexoffers applied to the devices. In order to simplify and automatize the process, 2 mechanisms were implemented:

- One responsible for the emission of flexoffers and reception of the corresponding schedule
- Another for creating a mechanism to retrieve the schedules from data base and enforce the devices to operate depending on them

As such we have 2 threads running, one for each mechanism. As soon as the system is stated 2 methods will load those threads into a Timer. Each Timer is configured to execute the thread when a certain time is met.

3.3.1 FLEXOFFER EMISSION

When the system is 11pm then the timer Task executes the FlexofferTimer(FT). Ft will gather all the devices that have active Flexoffers from the database. From there it will build a ExecutorService with a thread pool size equals to the number of devices. From there for each device, every active Flexoffer is retrieved form the database. For each flexoffer a ExecuteFOEmission thread is built and by the ExecutorService. This description can be found Figure 32.

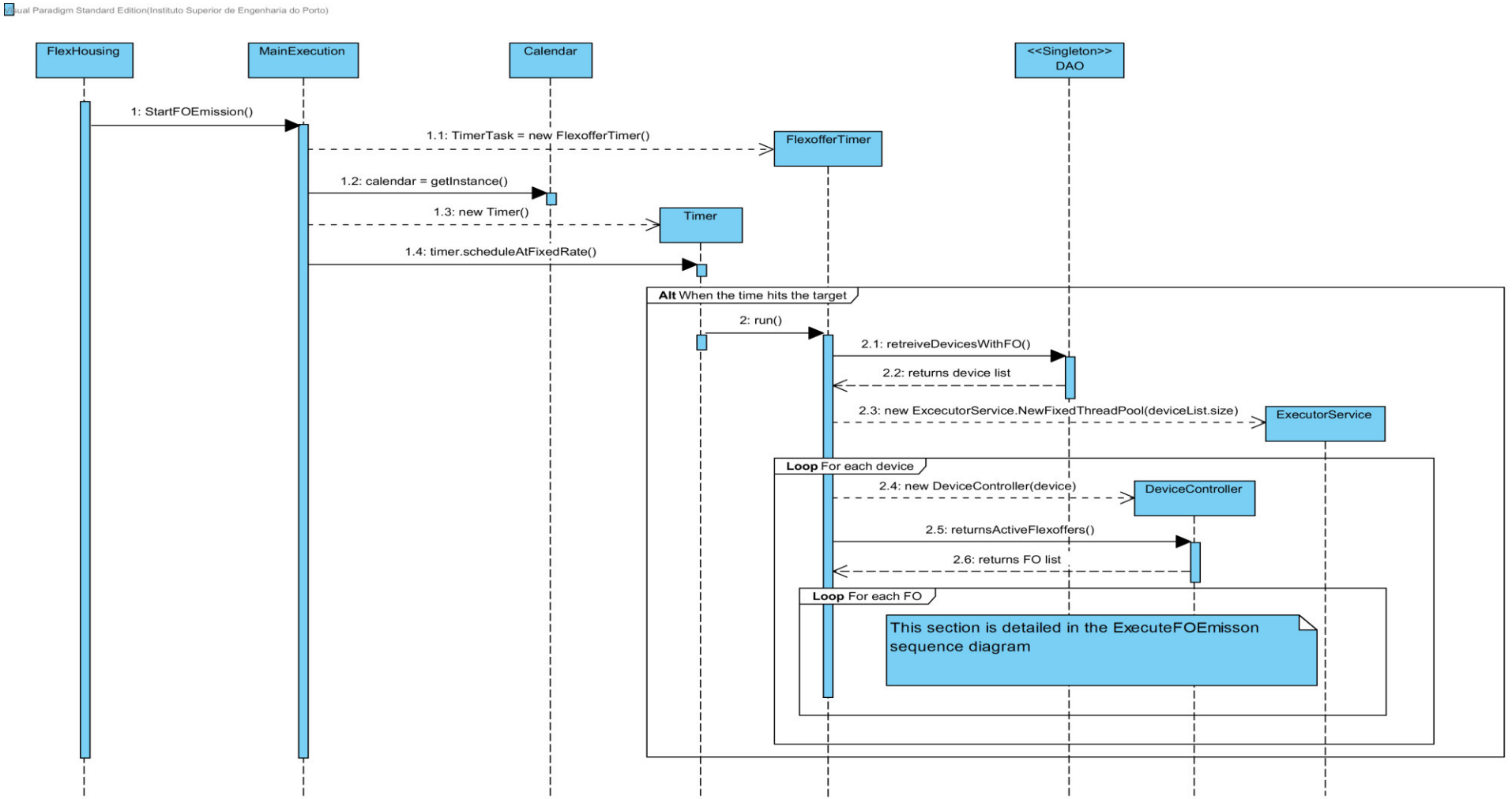

Figure 32 - Diagram sequence for the FlexofferTimer

The ExecuteFOEmission thread is built using a Flexoffer and a Device. The thread retrieves the instance of the FlexofferAgent and executes the generateFlexoffer method and the flexoffer is then sent to the Flexoffer Services. The thread then waits for the schedule to be attached to the flexoffer. When the schedule is delivered the WP5 ID of the Flexoffer is updated in the database. The thread also retrieves the schedule that was sent back and adds to the database. Figure 33 is an UML diagram depicting this description.

3.3.2 ACTUATION SCHEDULE

Similar to the emission of flexoffers, when the system is started a thread runs to monitor the time of the system. When the time hits the target then it executes the ActuationTimer thread. Again similar to the flexoffer emission the thread builds an ExecutorService in order to execute a pool of threads. The size of the poll is, again, again determined by the number of devices with schedules that was previously retrieved from the database. Then the schedules for each device is retrieved. ActuationTimer then builds a ExecuteActuation thread for each schedule. This description is also depicted in Figure 34.

When a ExecuteActuation is created, it receives, as parameters, a flexoffer schedule and a device. From there the schedule in converted into a NextDaySchedule. The NextDaySchedule contains an array of integers with 96 positions, one for every quarter hours in a day, called commutations. The first position corresponds to 00h00 while the 5^{th} one corresponds to 01h15 and so on. When created, the NextDaySchedule will fill all the slots of the commutation array with 0. Then will fill the array with 1's on the positions corresponding to the time that the flexoffer schedule was allocated consumption. For example, if a schedule has consumption moments at 00h15,00h30 and 01h00 it will to the 5 first positions of the array to look like this: [0, 1, 1,0, 1, …]. The ExecuteActuation thread retrieves the commutation array and executes a cycle for all the positions. For each iteration it will verify if the state changes from the previous state. If the states differ, then the thread will invoke the actuate method of the device in order to toggle the state. This description is depicted in figure 35.

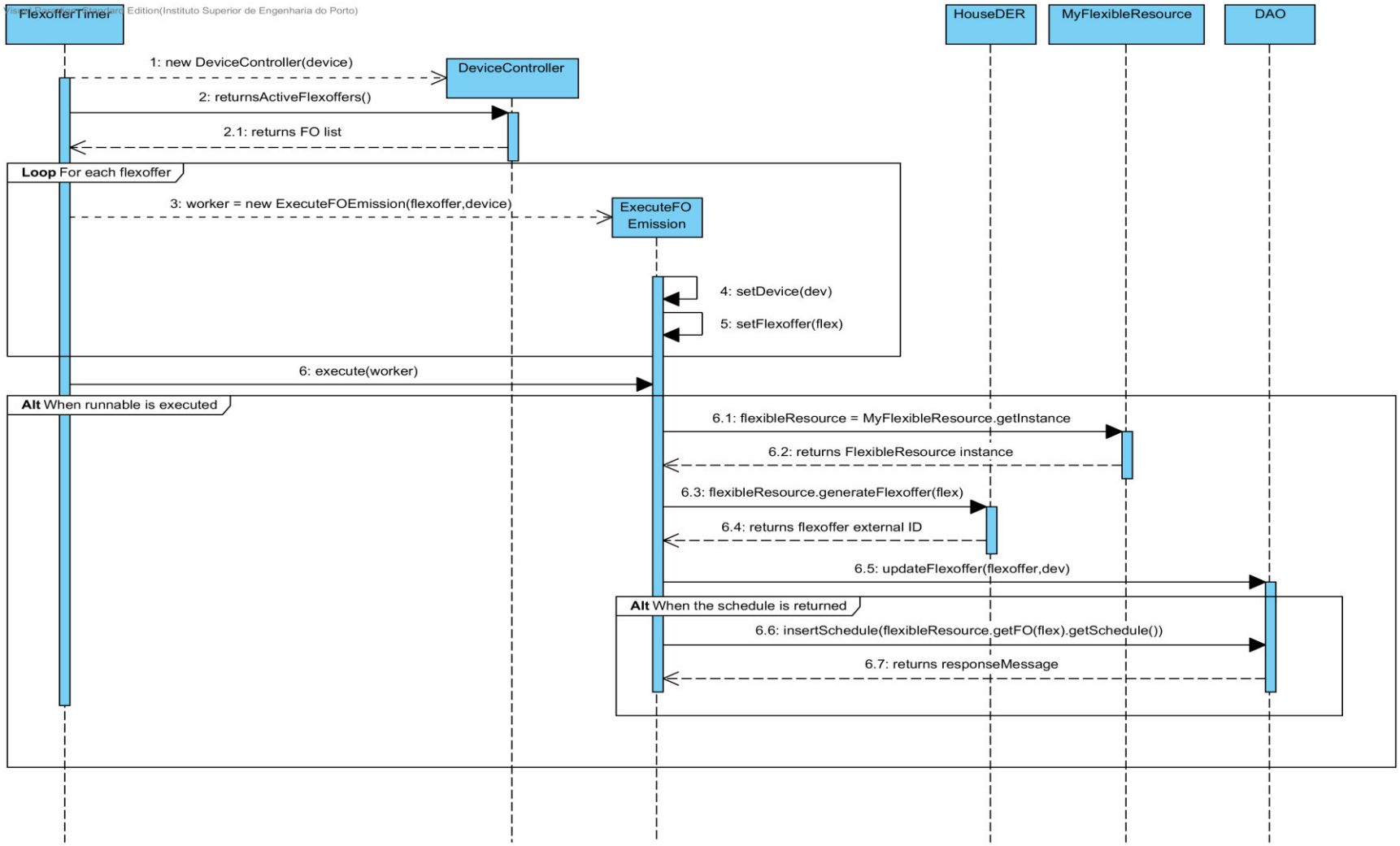

Figure 33 - Sequence diagram for the inner mechanism of the FO emission

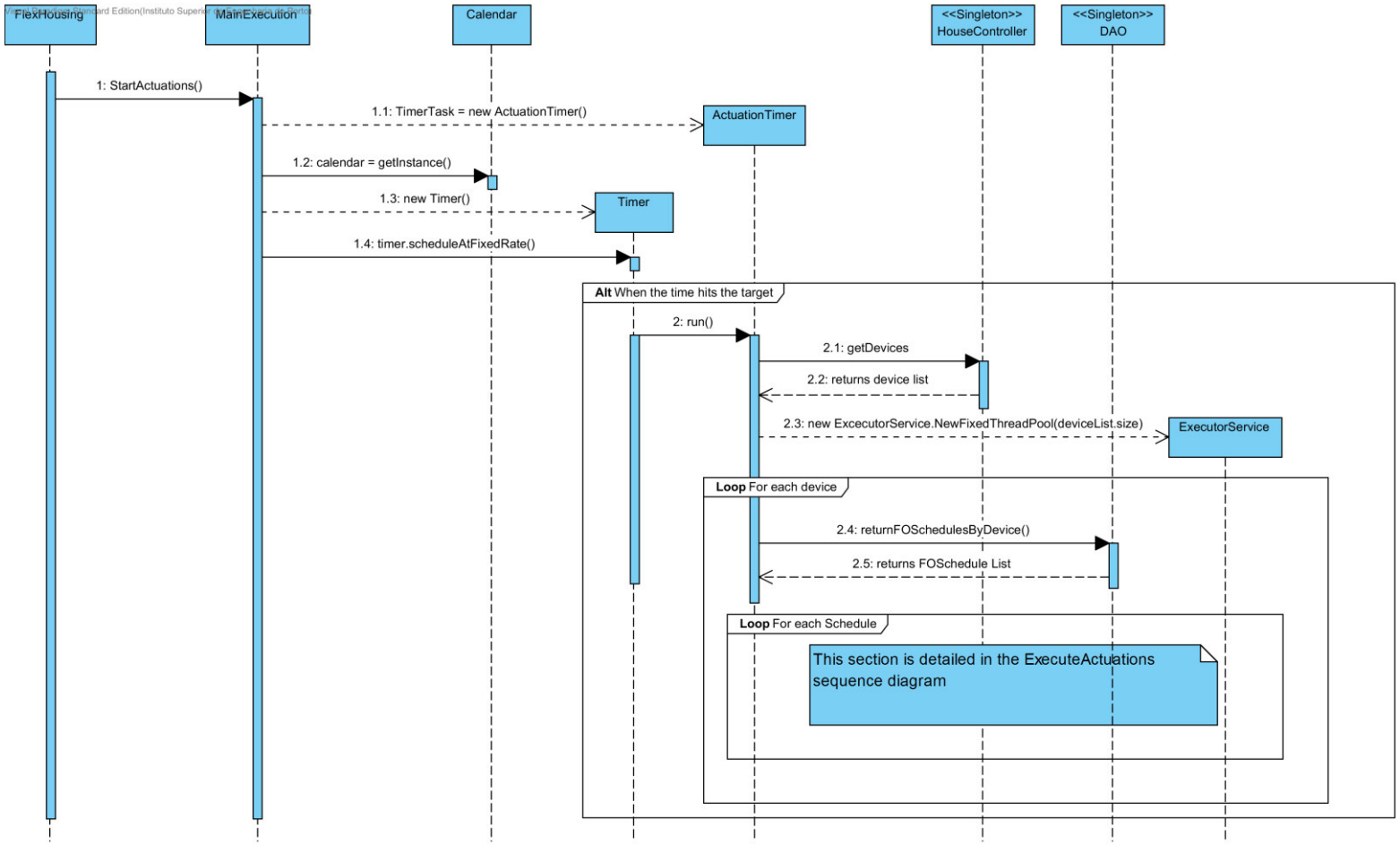

Figure 34 - Sequence diagram for the creation of ExecuteActuation threads

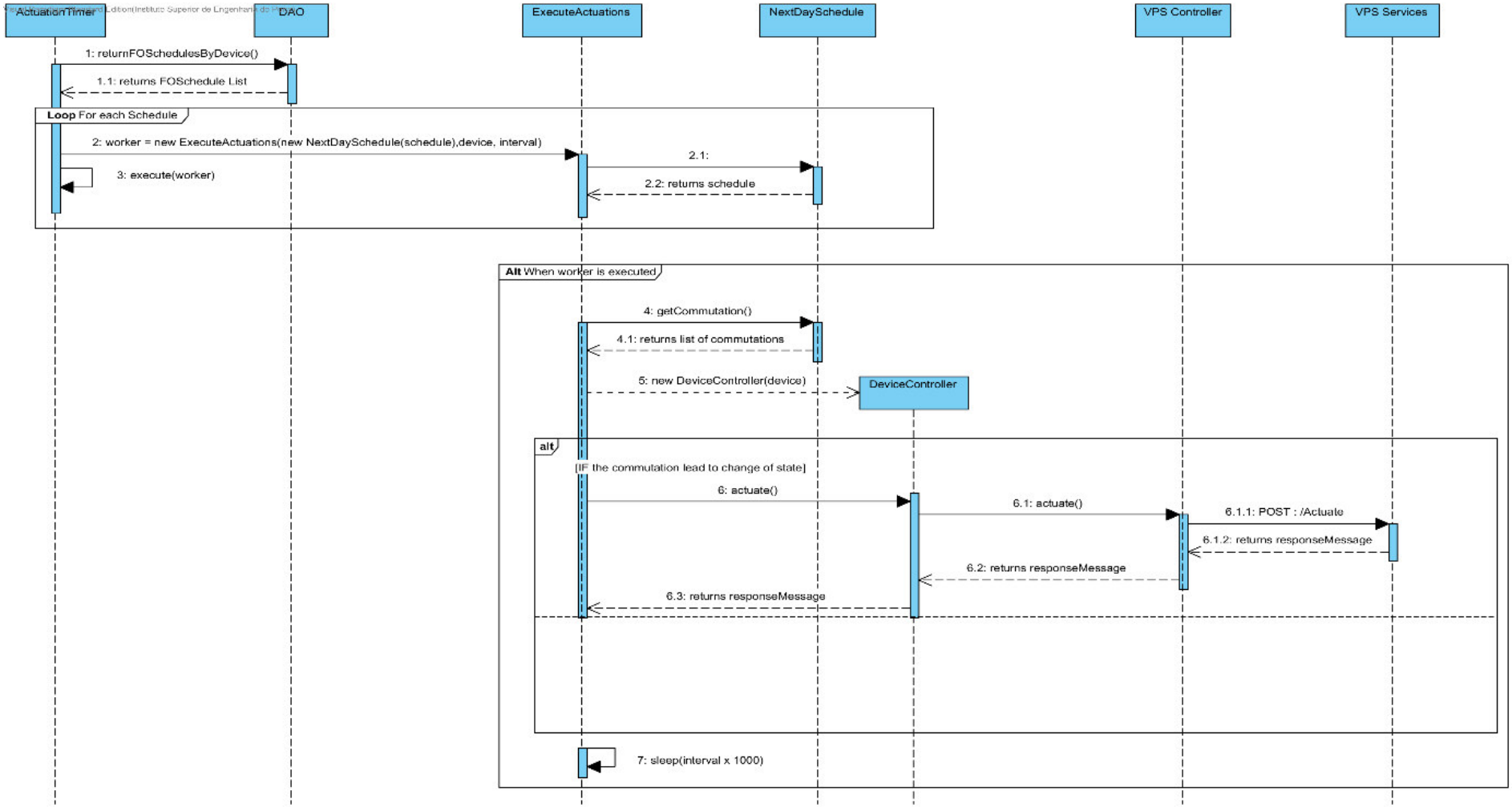

Figure 35 - Sequence diagram for the ActuationExecution thread

3.4 CLASSES AND PATTERNS

This section describes the core class in the FlexHousing Middleware system, as well as, enumerating the pattern in use. The Class diagram of the core class can be found in figure 36.
The classes are distributed into layers. The DAO class is in the Persistence layer and is only accessed by the controller which are in the Application layer, accompanied by the internal mechanism class. The models are in the in the Business layer. Every other class is in the External Services layer.
The controllers are only invoked by the business or external services. The DAO is only invoked by the Controllers. An instance of a Model class can only be created, updated, read or deleted by a controller. Each request sent to services hosted by the middleware is met the invocation of a controller method. Further description and evidence of this can be found in the SD/IDD description of the FlexHousing Middleware system.

System name	Path
FlexHousing Middleware SD-IDD	`Arrowhead/SD-IDD FlexHousing Middleware.docx`

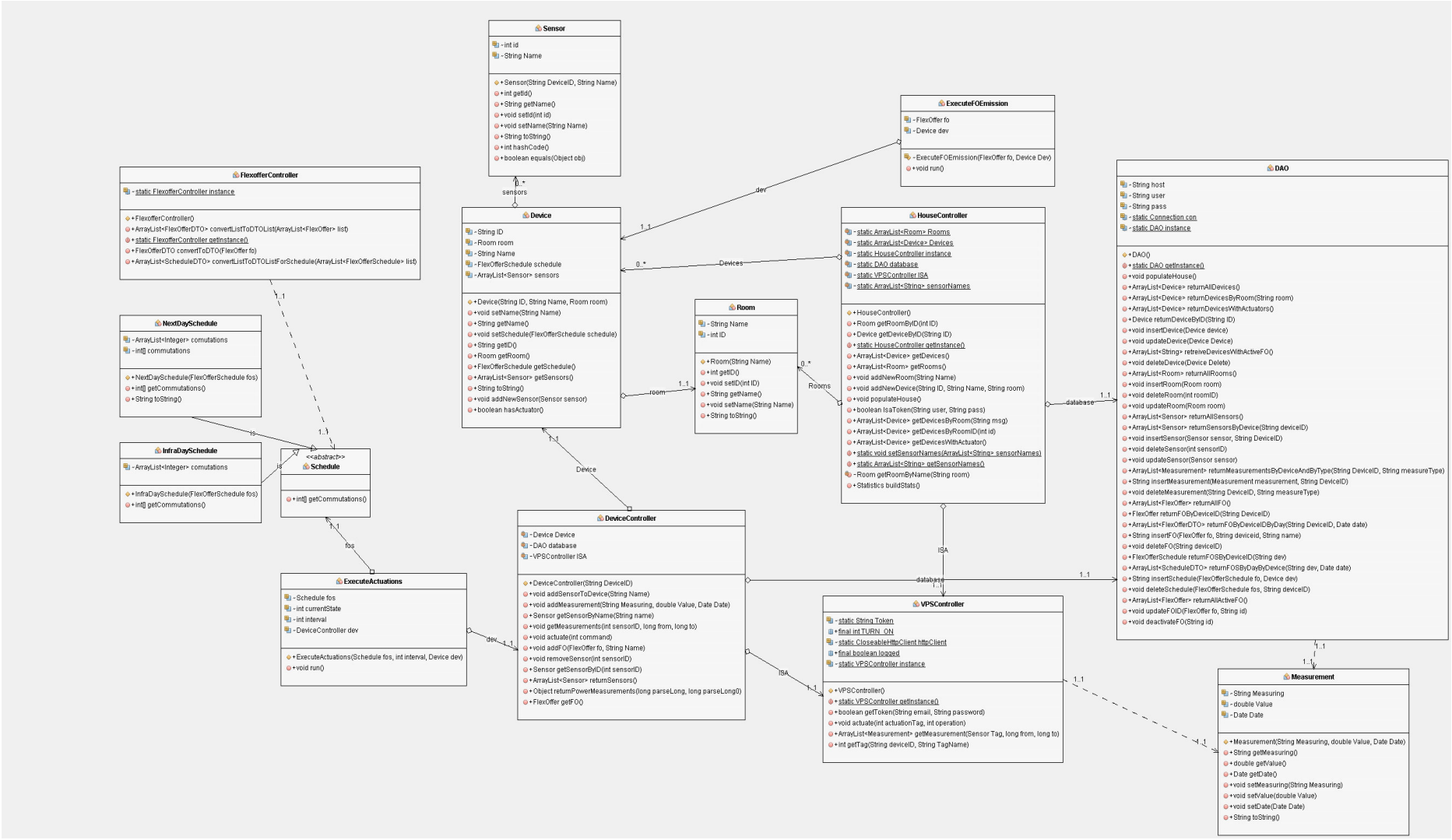

Figure 36 - Class diagram for FHMW

3.5. DATABASE

When an object is altered or added to the system, it is also persisted in the database. The current database is an Apache Derby DB. This section will describe the schema for the database. The full schema can be found depicted in figure 37.

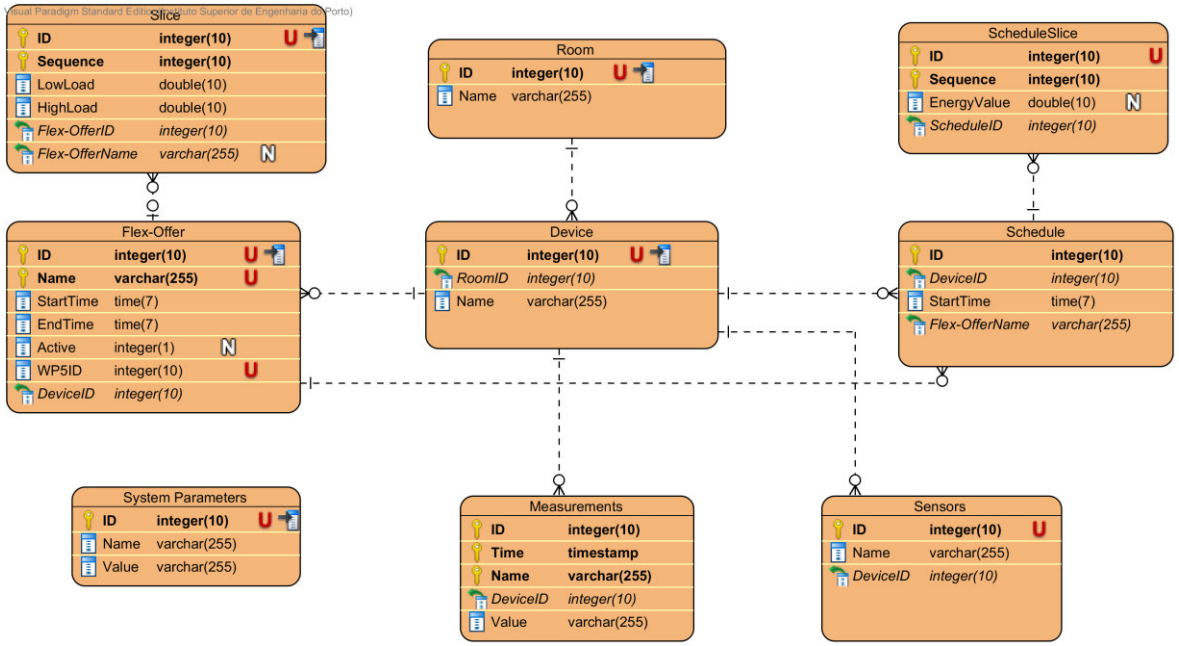

Figure 37 - Schema for the database in the middleware

The most tables in the schema are easy to perceive and only the more complex ones need explaining, which a corresponding table can found in this section, contain the name of each field and a brief description. The schema represents the various object that needs to persist in this system. The way the system saves the data for the flexoffers and for schedules is that we have 2 tables: one for the description (table 28) and another for the slices. The flexoffer has a name, a generated ID, start and end time, the id of device it is attached to and an integer representing the ID of the Flexoffer in the Flexoffer services. When storing the slices, it needs to store the upper and lower bound for the corresponding Flexoffer. To do so, each entry tin the Slice table has 6 columns: the value for both bounds, the name and ID of the flexoffer it belongs to and the ID and sequence number, Sequence number gives us the order of the slice. A similar process is applied to the Schedule and its slices

Column Name	Type	Description
ID	Integer	Auto generated number for the local identification
Name	Varchar	Name of the Flexoffer
StartTime	Time	Date and time of the earliest start of the Flexoffer
EndTime	Time	Date and time of the latest start of the Flexoffer
Active	Boolean	Determines if the flexoffer is still active
WP5ID	Integer	The ID of the Flexoffer in the Flexoffer Services
DeviceID	Integer	ID of the Device it is applied on.

Table 27 - Table description for the Flexoffer

4.2.6 SYSTEM DESIGN DESCRIPTION – FRONTEND

This document defines the System Design Description of the FlexHousing FrontEnd system. FlexHousing FrontEnd, bridging the flexoffer concept and external technologies with the common user. The focus of this document is to describe the implementation of the system architecture, the used components, and the use-cases supported by UML diagram.

1. SYSTEM DESIGN DESCRIPTION OVERVIEW

Name	FlexHousing FrontEnd
Owner	ISEP

The FlexHousing FrontEnd system has been developed by CISTER/ISEP, in the context of the Work Package 5 of the Arrowhead project. The goal of the project was to build a pilot that constituted a proof of concept for the usage of flexoffers as integrated with external technologies. FlexHousing is a system comprised of a middleware (FHMW) and a front-end type Webserver for user interaction (FHFE). FlexHousing Front End (FHFE) is a C# project hosted on a webserver, allowing the user to configure and use the FlexHousing Middleware (FHMW) within his building.
FlexHousing is a system comprised of a middleware (FHMW) and a front-end type Webserver for user interaction (FHFE).
A more abstract description of the FHFE system can be found on the document referenced in Table 2.

System name	**Path**
FlexHousing	`Arrowhead/ SysD FlexHousing FrontEnd.docx`

The web-based front-end FHFE was implemented as a showcase prototype. Its goal is to provide a basic graphical interface to demonstrate the capabilities of the middleware, and as such has been changed with each version of the middleware FHMW. Thus, the FHFE is intended to be an added value to the FHMW, and as such it covers all the services of FHMW itself; on the other hand, the FHFE cannot be considered a mature product by itself and does not cover all possible operations unrelated to FHMW. Incomplete use cases comprise missing CRUD operations over Rooms and Devices, and FHFE to consume data from Use Case 8 (see Section 4.2.1) . Regarding this latter limitation, the FHFE is already able to provide a representation of the data from other Use Cases, and the only missing graphs regards showing a unique graph reporting the Flexoffer schedule and measurements for the same period of time.

2. USE CASES

2.1 NON FUNCTIONAL REQUIREMENTS

To guarantee the non-functional requirements described in the document referenced in Table 2, this section lists the proposed solutions to its corresponding requirement:

- Availability: Deployment on a dedicated server.
- Interoperability & Extensibility: Usage of SOLID software principles, developing a high cohesion and low coupling code.
- Performance: Usage of high performance technologies.

2.2 USE CASES

Each implementation of a Use-case is accompanied by a SSD depicting the workflow, a reference to the implementation of the services the front end uses and an explanation of how it was implemented

and any decisions taken. The SSD will depict the interaction between the different objects within the front end web server and the calls it makes to the FlexHousing middleware.

2.2.1 SEND FLEXOFFER

The workflow of the sending of flexoffers is depicted in Figure 38.

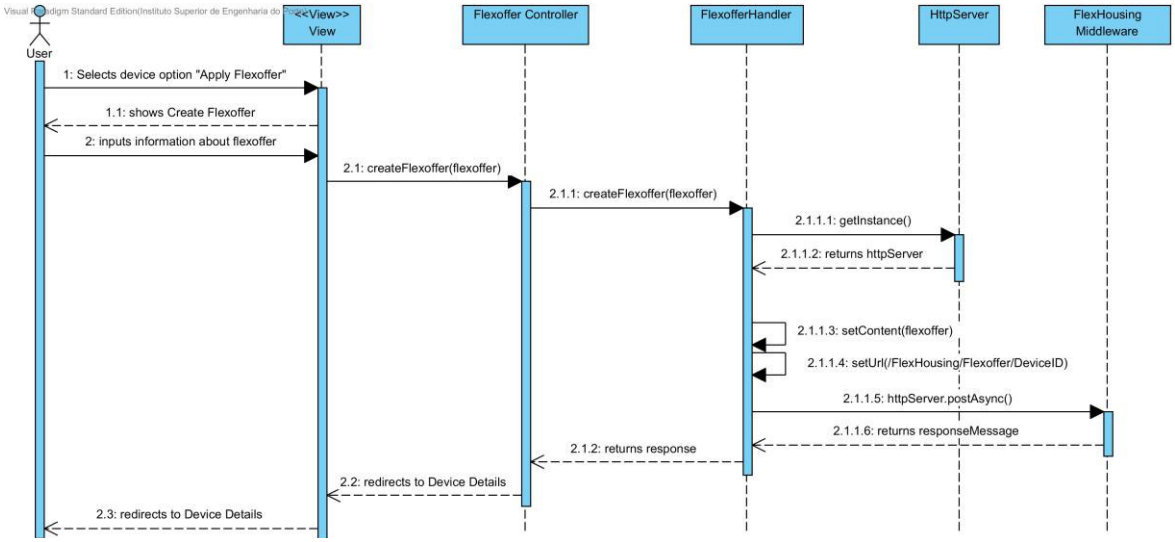

Figure 38- Sequence diagram for UC 1

The user is presented with the list of the devices in the part of the System of Systems that he owns/has access to. The user selects “Apply Flexoffer”. The page is redirected to the `View` and the user is prompted to fill input information on the name of the flexoffer, start and end time, and the pattern for the consumption. The info is collected and sent to the `Flexoffer Controller`. From there, the controller invokes the `Flexoffer Handler`, responsible for the communication with the `FHMW`, more specifically for the Flexoffer interface. The handler executes the request, posting the flexoffer. The handler receives the response from the server and sends it back to controller. If the request was responded with any errors, the user is notified of the success of the operation.

2.2.2 RECEIVE SCHEDULES

The following figure, Figure 2, shows the step the system does to receives schedules.

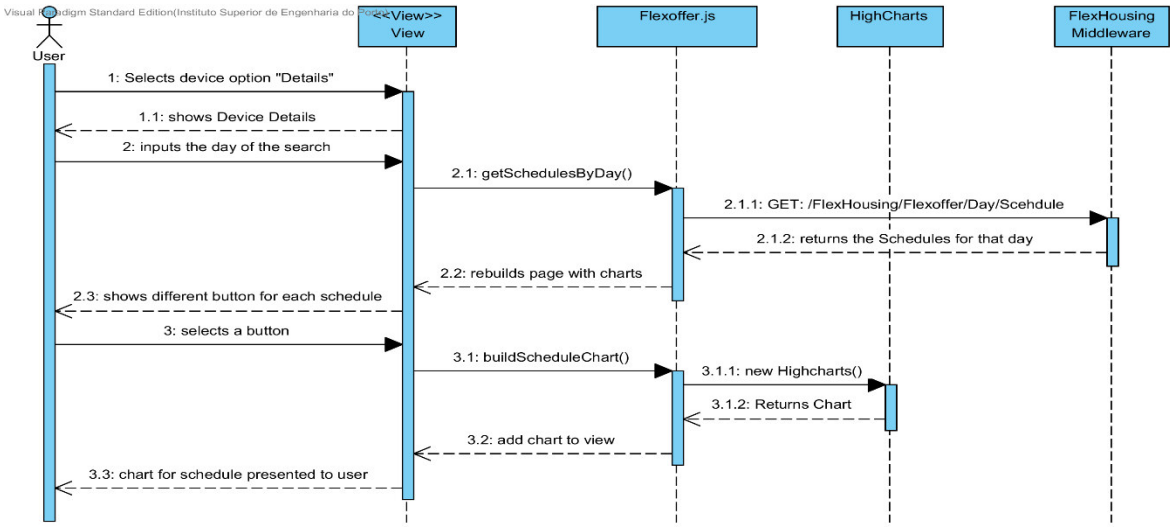

Figure 39 - Sequence diagram for UC 2

The previous figure, Figure 39, shows the step the system does to receives schedules. The user selects the "Details" option of the device. The `View` is loaded and the user is presented with a date chooser html object. When a date is selected, a JavaScript function is executed, retrieving the schedules for that specific date from the `FlexHousing Middleware`. The data is stored and for each object a html button is created, using the name of the flexoffer the schedule originated from as a title. When a button is pressed, a second JS function is invoked, building a `Highchart` object [31], depicting the data from that specific schedule. The chart is put in view for the user to see.

2.2.3 LOGIN

Figure 40 depicts the sequence of steps executed when the user logs into the system.

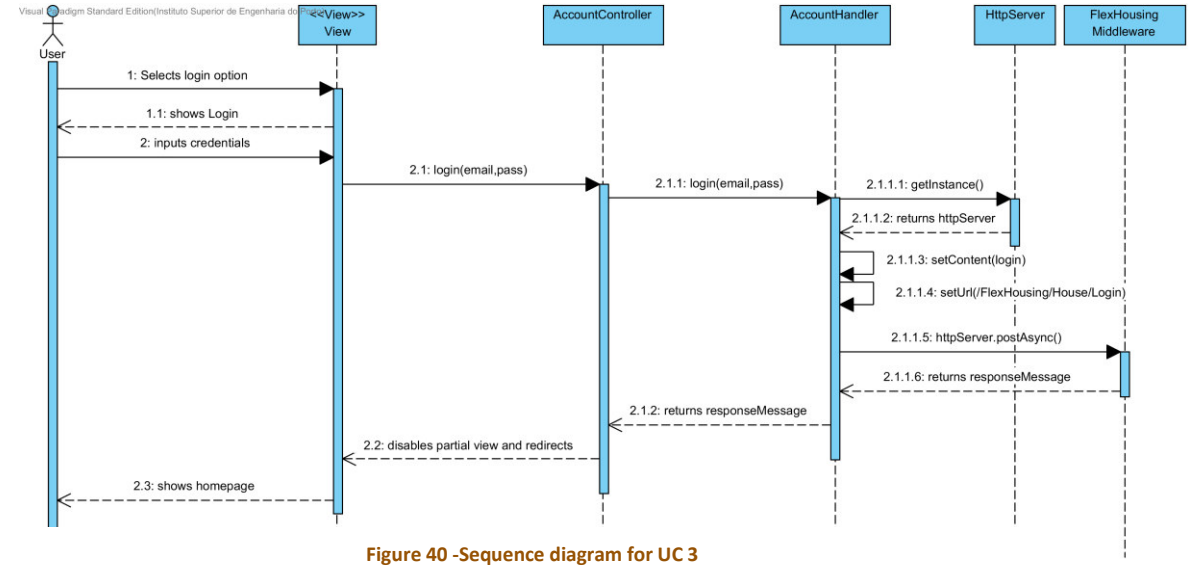

Figure 40 -Sequence diagram for UC 3

The user selects the “Login” option. The user then inputs the email and password. The controller is invoked, capturing the information. The `AccountController` then uses the `AccountHandler` for the execution of the request. The login is sent to FHMW. When the FHMW replies back and if the response was successful, then the user is redirected back to homepage and is now able to perform the rest of the FHFE features.

2.2.4 MANAGE HOUSE

The figure below (Figure 41) depicts the sequence diagrams of the various steps and interactions between the various components, enabling the managing of Rooms.

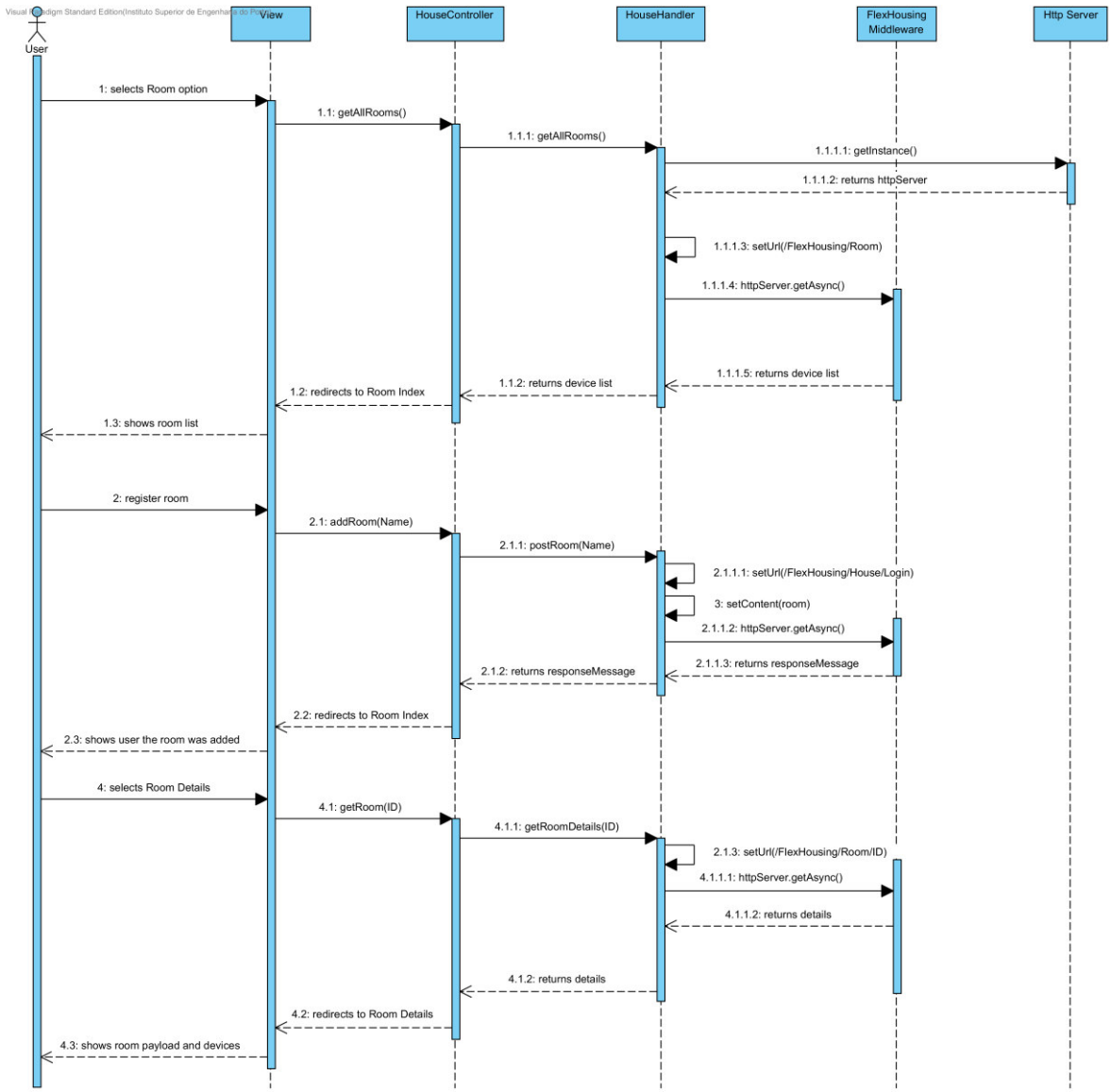

Figure 41- Sequence diagram for UC 4

When selecting the “Room” tab the user can check every room currently registered in system. When the tab is pressed, the `HouseController` is issued the `getAllRooms()` request. The controller proceeds to request execute the function with the same name located in the `HouseHandler`. Its objective is to contact the middleware and request the list of rooms. When the response is sent back, the controller receives the list from and handler a new `View` is built using the list of rooms as Model.

The option within that `View`, "Create Room", allows the user to add a new room to the system. The user is redirected to a new View, where he has to input the name of the new room. When the information has been provided and the user confirms the creation, the `HouseController` captures that action. It will execute the `postRoom()` function of the `HouseHandler` providing as a parameter the name of the room. The handler will contact the middleware and execute a POST function. The room is added. If no exception was raised, the page is redirected to the Index of rooms and the new room is now amongst the list.

2.2.5 MANAGE DEVICES

The following figure (Figure 42) shows a UML representation of the UC 5.

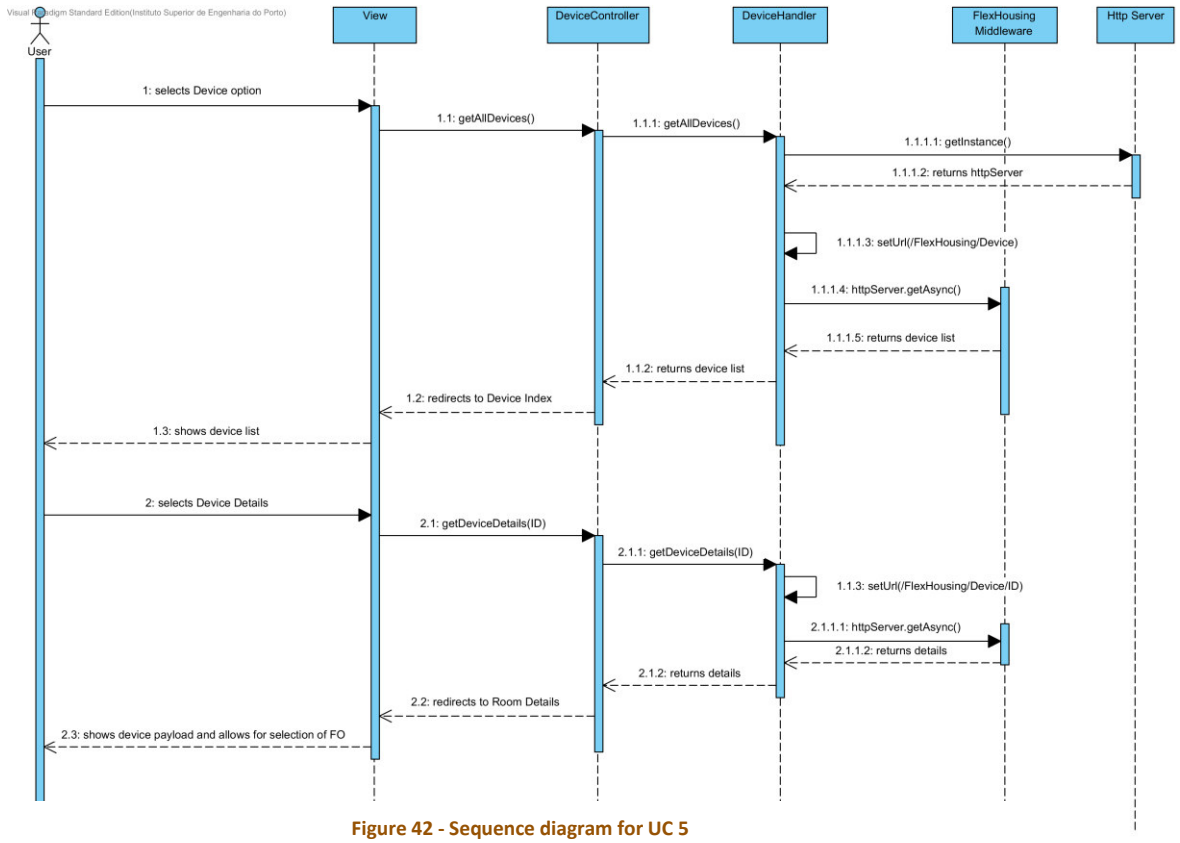

Figure 42 - Sequence diagram for UC 5

The "Device" tab on the main Menu leads to the list of devices currently registered in system. When the option is selected, the `DeviceController` invokes the `getAllDevices()` of the `DeviceHandler`. From there the handler executes a request targeting the middleware. The middleware responds with the list of devices. When the handler sends back the information to the controller. A new View is redirected to user, depicting the list of devices received. When the user is presented with the list of devices one of the option for device is the "Details" option. When selecting that option, the `DeviceController`'s `getDetails()` function is invoked. This triggers the function with the same name in `DeviceHandler`. The handler will execute a request aiming for the middleware, using the ID of the device whose option was clicked as a parameter. The middleware responds with the details of the device. The controller receives that data from the handler and creates a View with the details. The `View` is then presented to the user.

2.2.6 CHECK MEASUREMENTS

Figure 43 represents the sequence diagram for the check measurements use-case.
When presented with list of devices, one of the options for each device is “Check Measurements”.

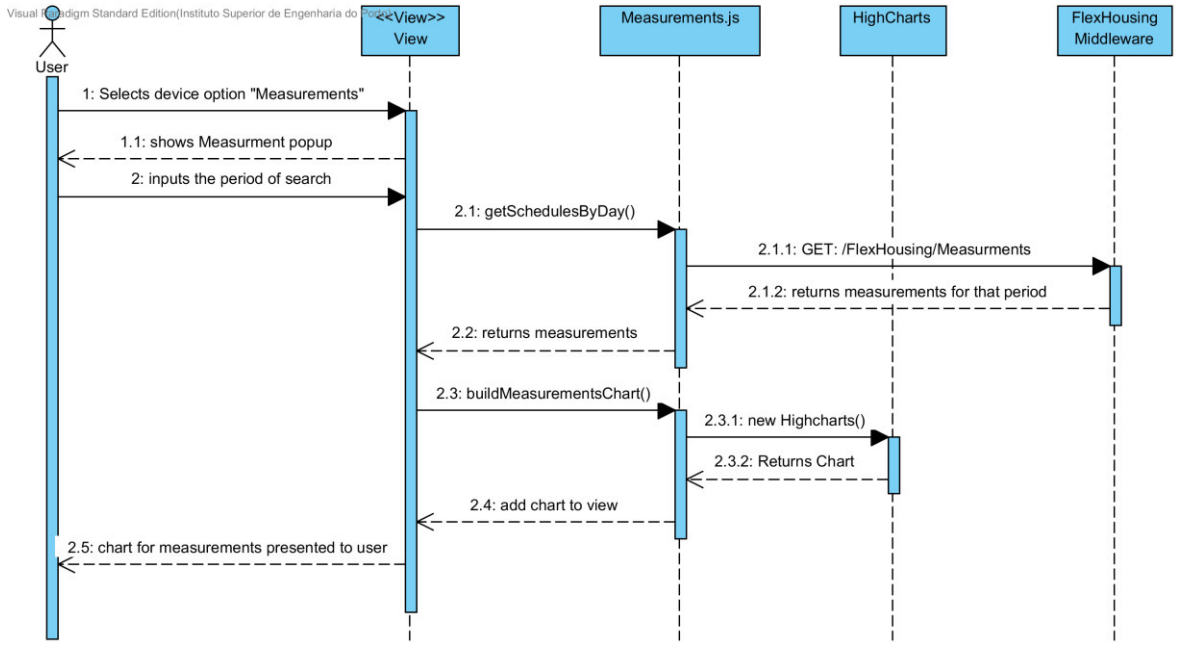

Figure 43- Sequence diagram for UC 6

When the user selects that option, a modal pops up. The modal has 2 fields for input, the start date and end date of the period you want to check the measurements for. The dates are collected and sent as parameters to the `getSchedulesByDay()` function in the `Measurements.js` JavaScript. The function communicates with the middleware requesting the measurements for the period the user has selected. When the response is sent, the function will then create a `Highchart` chart, graphically depicting the data collected. The chart is then added to the View for the user to observe.

2.2.7 ACTUATE ON DEVICES

The following figure (Figure 44) depicts the steps taken by the system when the features for UC 6 are employed.

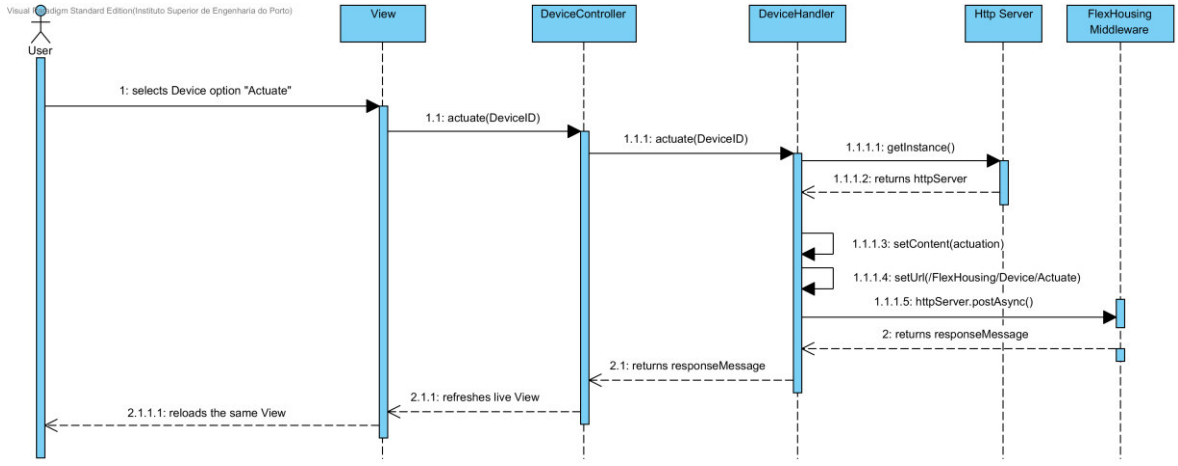

Figure 44- Sequence diagram for UC 7

The device in the device list View have the “Actuate” option. The selection of this option leads to the invocation of the `Actuate()` function in the `DeviceController`. The controller requests the `DeviceHandler` to execute the function with the same name. The handler emits a request towards the middleware with the ID of device in question. When the response is received, it forwarded to the controller, who refreshes the current `View`. The device who had the option executed on had its state toggled.

2.2.8 VERIFY IF FLEXOFFER WAS RESPECTED

Due to time constraints, the implementation of UC8 is an ongoing work, and still has to be completed. The implementation would revolve around the same procedure done for other UC: The user selected the option and a View permitting the selection of the flexoffers the device had had applied. The list would only contain flexoffers with attached schedules: invalid or unscheduled flexoffers wouldn’t appear. From there the user would select a flexoffer. The system would collect the information of that flexoffer and schedule and would request the middleware for the measurements of the same time period, similar to how UC 6 functions. The system would then build a chart, using Highcharts like it did previously, and show the user the bar chart of the flexoffer with a different colored plot line depicting the consumption for each quarter hour within that period of time. The end result would look similar to Figure 45

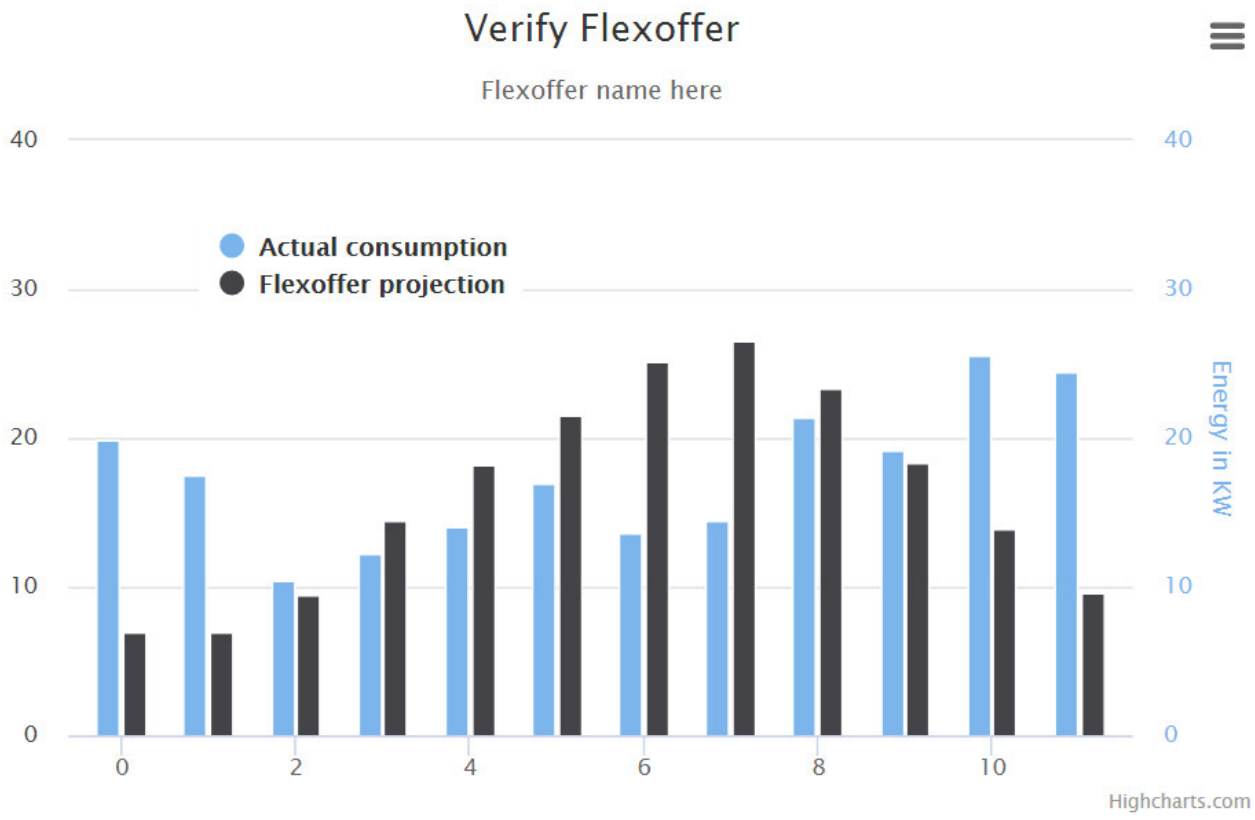

Figure 45 - Chart for the projected implementation of UC 8

3. SOLUTION DESCRIPTION

The purpose of this chapter is to describe the implementation of the solution. Initially, an overview of the system architecture is described with the support of a component diagram, then all the core classes used on the code implementation are explained along with a class diagram.

3.1 COMPONENT DIAGRAM

Typical to web design, a MVC pattern was adopted in the FHFE. More specifically we have ViewModels, Views and Controllers. In order to enforce an even better development methodology, a Handler layer is responsible for the communication with other components. In this particular cases, no DTO are involved because the objects that are exchanged are the same as the Models. This description is depicted in Figure 46.

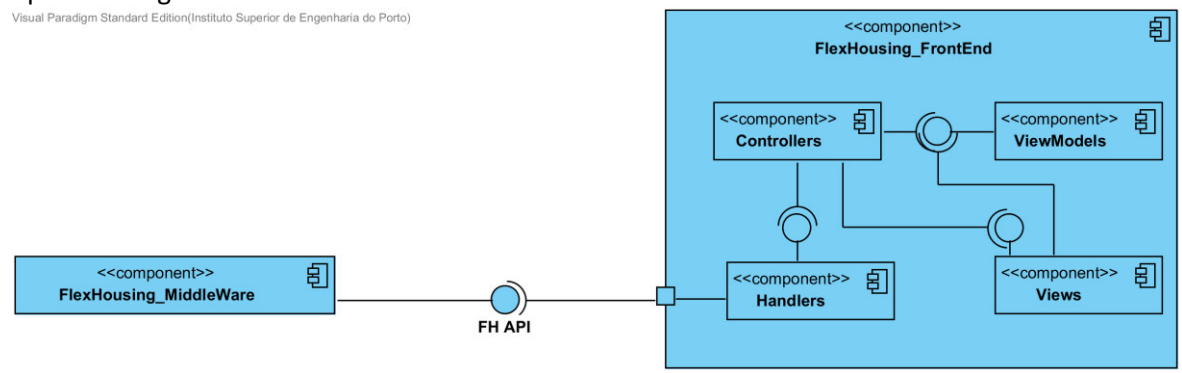

Figure 46 - Component Diagram of the FlexHousing system.

3.2 CLASSES STRUCTURE, AND DESIGN PATTERNS UNDER USE

The MVC pattern is employed in order to facilitated the incremental development. In case it's implemented using ASP.NET MVC. This means the Views use the Models for the building of the Web forms, for each controller method there can a View associated to it and for each major model there's a controller used to execute the operations regarding those models. In this case the Models are actually ViewModels due to the fact they represent the information the system shows the User.
This system has 5 major building blocks:

- The Controllers
- The Views
- The ViewModels
- The Handlers
- The Scripts

3.2.1 CONTROLLERS

Responsible for the logic of any operation. Normally linked to button or option in the view. There is the Device, House and Flexoffer Controller. The controllers are used when the action involved leads to a new View. Figure 29 depicts the Actuate() method.

Table 28 - Sample of code representing the controller method Actuate

```
public async Task<ActionResult> Actuate(string id)
        {
            Actuation act = new Actuation();
            act.ID = id;
            act.command = 3;
```

```
            await DeviceHandler.Actuate(act);
            return RedirectToAction("Index", "Device");
        }
```

3.2.2 VIEWS

Using Html, Css and Razor, the system is able to create web pages integrated with WebForms. These forms allow to easily represent the Models of the system. The" @" annotation starts a razor statement or line of code. That line of code is processed before the page is rendered on the server side. Employing razor enables us to embed controller methods in our options.

Table 29 - Code sample for a View in the FrontEnd

```
@foreach (var item in Model)
    {
        string id = item.ID;
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.ID)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.deviceRoom.Name)
            </td>
            <td>

                @Html.ActionLink("Details", "Details", new { id = item.ID }) |

                @Html.ActionLink("Actuate", "Actuate", "Device", new { id = id },
null) |
                <a id="@Html.Raw(id)" , href="#myModal"
onclick="getLine(this.id)" data-toggle="modal">Measurements</a>
                <br/>
                @Html.ActionLink("Apply Flexoffer", "Flexoffer", new { id =
item.ID })
            </td>
        </tr>
    }
```

The code represented in table 30 shows the usage of razor and html elements. For each object in the Model list this method will create a new row in the table and add those option, which are connected to controller methods.

3.2.3 VIEWMODELS

These classes represent the objects will we operate with. They are normally called Models but in case, because they are also exactly what the system shows the user, they can be called ViewModels. They are used by the razor for the creation of WebForms in the rendering/building of the Views.

Table 30 - The representation of the Flexoffer object

```
public class Flexoffer
    {

        public long startTime { get; set; }

        public long endTime { get; set; }

        public List<double> upperEnergyValues { get; set; }

        public List<double> lowerEnergyValues { get; set; }

        public string name { get; set; }
    }
```

Table 31 depicts the code for the representation of the Flexoffer object. The name of each field is built in order to correctly match the name of the JSON string is received by the handlers. Doing so enables the system to deserialize the string and returns the matching objects.

3.2.4 HANDLERS

These objects have the responsibility of sending the request towards the middleware. They use a HttpServer, whose default URL is the abstract interface of the middleware. From there they build their content, add the correct header and concatenate the rest of the URL in order to reach the service. Then depending on the request they will execute the operation (GET, POST, DELETE, …). When the response is received they will return it to the controller who invoked the handler in the first place.

Table 31 - Code for the GetAllDeviceList function of the device handler

```
public static async Task<ICollection<Device>> GetAllDeviceList(string url)
        {
            ICollection<Device> deviceList = new List<Device>();
            var client = WebClient.GetClient();

            HttpResponseMessage response = await client.GetAsync(url);
            if (response.IsSuccessStatusCode)
            {
                string jsonString = await response.Content.ReadAsStringAsync();
                deviceList =
JsonConvert.DeserializeObject<List<Device>>(jsonString);
            }

            return deviceList;
        }
```

The implementation of the function in table 32 is responsible for the retrieval of the devices in the middleware. Using the HttpClient, that Handler executes the GET request for the list of devices in FHMW.

3.2.5 SCRIPTS

Some of the operations, in order to be more responsive and user friendly, are done on client side, directly using the web browser. Such operations are stored in .js Javascript files. Those functions will retrieve information from the user, execute requests towards the middleware and alter the web page depending to the results.

Table 32 - Code for the appendSlice function

```
function appendSlice() {
    var from = document.getElementById("inputLower").value;
    lower.push(from);
    console.log(lower);
    var to = document.getElementById("inputUpper").value;
    upper.push(to);
    console.log(upper);
    var d1 = document.getElementById('Pattern');
    d1.innerHTML += "<li>From "+from+"kW to "+to +"kW.</li>";
    $('.modal-backdrop').remove();
    $('#myModal').modal('hide');
}
```

The script depicted in table 33 is used when the user is building the pattern for a flexoffer. The function retrieves the info from a modal, puts in the internal object and also adds it to a html object list for the user to keep track of his work.

4.2.7 SERVICE DESCRIPTION AND INTERFACE DESCRIPTION – MIDDLEWARE

This document describes the services hosted by the FlexHousing Middleware. Each service is described, has a sequence diagram of it implementation and an example for a request.

1. OVERVIEW

This document describes the services hosted by the FlexHousing middleware that supports the flexoffer pilot, including its interfaces and its information model. The purpose of the FlexHousing Middleware is to allow the user to perform CRUD (create, read, update and delete) operations on devices and rooms representations, flexoffer creation, measurement verification and automated actuations driven by received schedules.
The service integrates the flexoffer services implemented by Work Package 5 of the Arrowhead Project with external technologies, in this case represented by a smart plug service provided by Virtual Power Solutions.

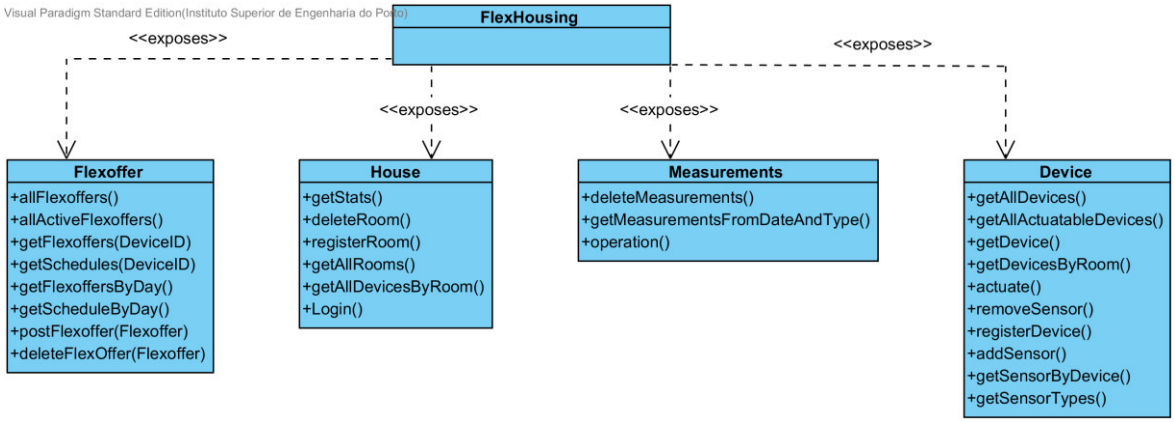

Figure 1 – FlexHousing FlexHousing interface Overview

The FlexHousing service is an application service. It is attached to an Arrowhead cloud containing an Aggregator as such multiple instances can be instantiated on the same network.

2. INTERFACES

The FlexHousing service exposes four interfaces, namely Flexoffer, House, Device and Measurements. In the following, the function names are preceded by the type of request it handles, with its name in capital letters (GET, DELETE, POST, PUT).

This section reports the definition of the functions related to each interface, and examples regarding the messages used in the context of the functions. Section 3 reports information regarding the Information Model, and it documents each field of the messages.

2.1 FLEXOFFER

This interface is used for CRUD operations upon flexoffers, namely creating flexoffers, retrieving flexoffers, with or without filters, and retrieving schedule.

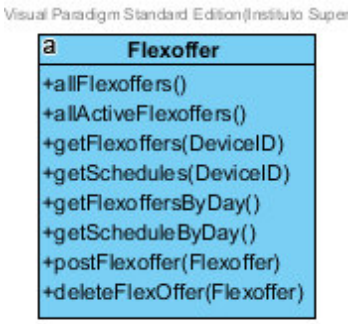

Figure 47 – Flexoffer Interface

Figure 47 depicts the methods attached to the interface.

2.1.1 GET GETALLFLEXOFFERS

The GetAllFlexoffers function is used to retrieve all the flexoffers created by the user. It returns all existing flexoffers, independently of their state, active, deleted or pending scheduling. This function is mainly used for debugging purposes.

Table 33 - Example of a request to get all flexoffers

PATH	
/FlexHousing/Flexoffer/GetAllFlexoffers	
Request	Response
Path Parameters:	Body
None	[{ "name": "Test1", "startTime": 1638912, "endTime": 1638912, "upperEnergyValues": [25], "lowerEnergyValues": [50] }, { "name": "Test2", "startTime": 1639008, "endTime": 1639008, "upperEnergyValues": [12,12], "lowerEnergyValues": [10,10]}, {…..}]
Query Parameters	
None	
Body	
None	

2.1.2 GET GETALLACTIVEFLEXOFFERS

It retrieves all the active flexoffers, namely the ones with an active schedule.

Table 34 - Example of a request to retrieve all active flexoffers

PATH

/FlexHousing /Flexoffer/GetAllActiveFlexoffers	
Request	Response
Path Parameters	Body
None	[{ "name": "Test1", "startTime": 1638912, "endTime": 1638912, "upperEnergyValues": [25], "lowerEnergyValues": [50]}, { "name": "Test2", "startTime": 1639008, "endTime": 1639008, "upperEnergyValues": [12,12], "lowerEnergyValues": [10,10]}, {…..}]
Query Parameters	
None	
Body	
None	

2.1.3 GET {PARAM}

This function retrieves the flexoffer that is currently attached to a device, and whose device ID corresponds to the parameter added to the path.

Table 35 - Example of a request to retrieve the latest flexoffer attached to a device

PATH	
/FlexHousing/Flexoffer/3ZU-VGC-N3J-NWK-9P	
Request	Response
Path Parameters	Body
Device ID – “3ZU-VGC-N3J-NWK-9P”	["name": "Test1", {"startTime": 1638912, "endTime": 1638912, "upperEnergyValues": [25], "lowerEnergyValues": [50]}, {"startTime": 1639008, "endTime": 1639008, "upperEnergyValues": [12,12], "lowerEnergyValues": [10,10]}, {…..}]
Query Parameters	
None	
Body	
None	

2.1.4 GET SCHEDULE {PARAM}

Similar to the function in section 2.1.1.3., it allows for the retrieval of the schedules bound to the device whose ID corresponds to the function path.

Table 36 - Example of a request to retrieve a schedule attached to a specific device

PATH	
/FlexHousing/Flexoffer/Schedule/3ZU-VGC-N3J-NWK-9P	
Request	Response
Path Parameters	Body

Device ID – "3ZU-VGC-N3J-NWK-9P"	[
	{
	"name": "Lamp",
	"Start": "Sep 24, 2016 7:00:00 PM",
	"energyAmounts": [
	15
	]
	}
	]

Query Parameters	
None	
Body	
None	

2.1.5 GET DAY {PARAM}

This function is used to access all the flexoffers created in a day specified by a query param attribute, for the device represented by the ID in the path. Normally requested when the user desires to check the flexoffer history of a given device.

Table 37 - Example of a request to retrieve the flexoffers on a certain day

PATH	
/FlexHousing/Flexoffer/Day/3ZU-VGC-N3J-NWK-9P?Day=2016-09-21	
Request	Response
Path Parameters	Body
Device ID – "3ZU-VGC-N3J-NWK-9P"	[{ "name": "Test1", "startTime": 1474452000000, "endTime": 1474459200000, "upperEnergyValues": [15], "lowerEnergyValues": [12] }, { "name": "Test2", "startTime": 1474452000000, "endTime": 1474459200000, "upperEnergyValues": [80], "lowerEnergyValues": [
Query Parameters	
Day – "2016-09-24"	
Body	
None	

	50] }]

2.1.6 GET SCHEDULE DAY {PARAM}

Similar to the function in section 2.1.1.5., this function returns all the schedules applied to the device specified by the path, in a day specified as the query parameter *param*.

Table 38 - Example of a request to retrieve the schedules for a specific day

PATH	
/FlexHousing/Flexoffer/Schedule/Day/3ZU-VGC-N3J-NWK-9P?Day=2016-09-24	
Request	Response
Path Parameters	Body

Device ID – “3ZU-VGC-N3J-NWK-9P”	[
	{
	"name": "Test1",
	"Start": "Sep 21, 2016 11:00:00 AM",
	"energyAmounts": [
	13
	]
	},
	{
	"name": "Test2",
	"Start": "Sep 21, 2016 11:00:00 AM",
	"energyAmounts": [
	65
	]
	}
	]

Query Parameters	
Day – “2016-09-24”	
Body	
None	

2.1.6 POST {PARAM}

Even though this function shared the same name as the one in 2.1.1.3., it allows the user to create a flexoffer through the POST operation. The body of the request contains a string representing the name of the flexoffer, its startTime and endTime in milliseconds, and two arrays of doubles representing the upper and lower energy values for each slice of the flexoffer.

Table 39 - Example of a request to store a flexoffer

PATH
/FlexHousing/Flexoffer/3ZU-VGC-N3J-NWK-9P

Request	Response
Path Parameters	Body
Device ID – “3ZU-VGC-N3J-NWK-9P”	ResponseCode(200)
Query Parameters	
None	
Body	
{ "startTime":1474452000000.0, "endTime":1474459200000.0, "upperEnergyValues":[25.0], "lowerEnergyValues":[12.0] }"	

2.1.7 DELETE

As the name implies, this function allows the deletion of a flexoffer. It is not supposed to be used often since, when a new flexoffer is created as a replacement for a previous one, the older one will be deactivated.

Table 40 - Example of a request to delete a flexoffer

PATH	
/ FlexHousing/Flexoffer?DeviceID=3ZU-VGC-N3J-NWK-9P	
Request	Response
Path Parameters	Body
None	ResponseCode(200)
Query Parameters	
DeviceID – “3ZU-VGC-N3J-NWK-9P”	
Body	
None	

2.2 DIAGRAMS

2.2.1 GET GETALLFLEXOFFERS

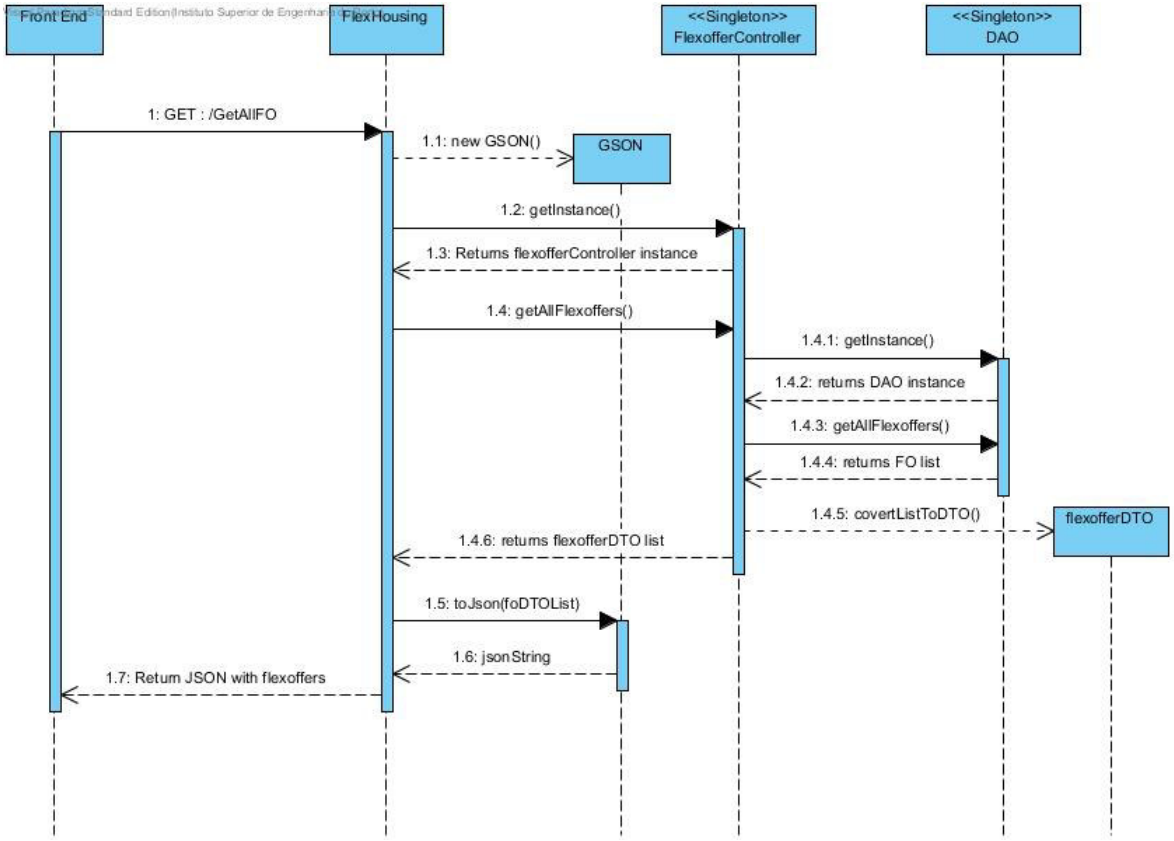

Figure 48 - Sequence diagram for getAllFlexoffers

Figure 48 depicts the steps executed when retrieving all the flexoffers in the system.

2.2.2 GET GETALLACTIVEFLEXOFFERS

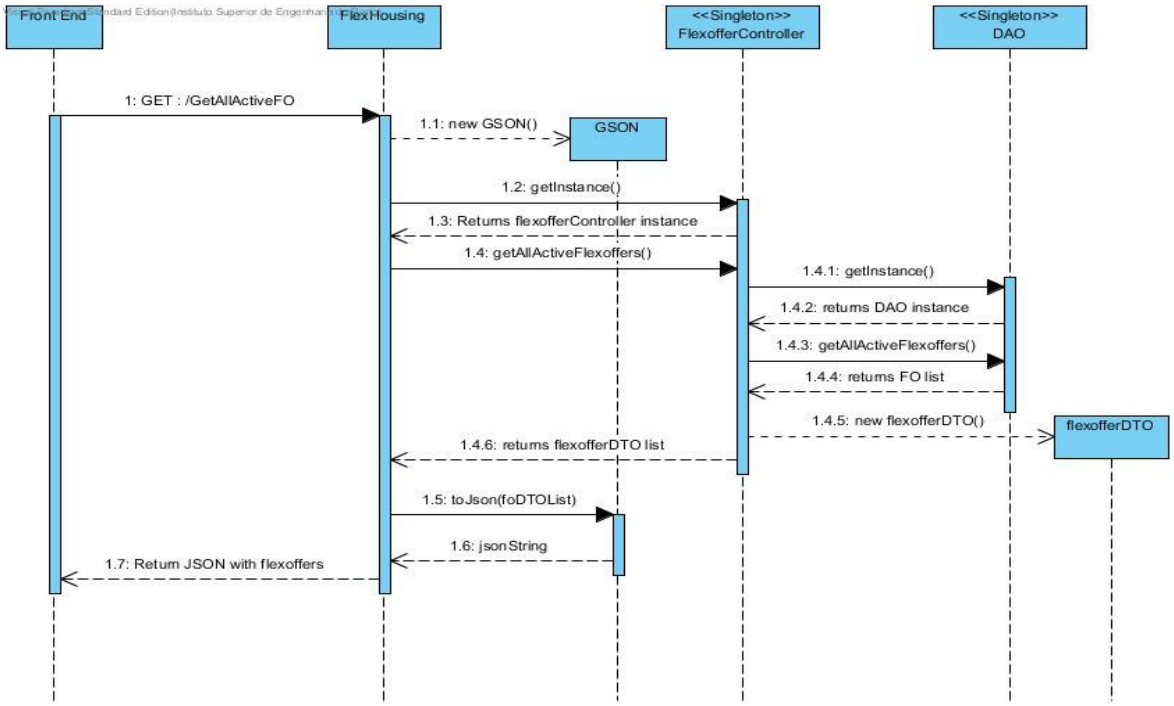

Figure 49 - Sequence diagram for getAllActiveFlexoffers

Figure 49 represents the sequence diagram for the execution of the get all active flexoffers service.

2.2.3 GET {PARAM}

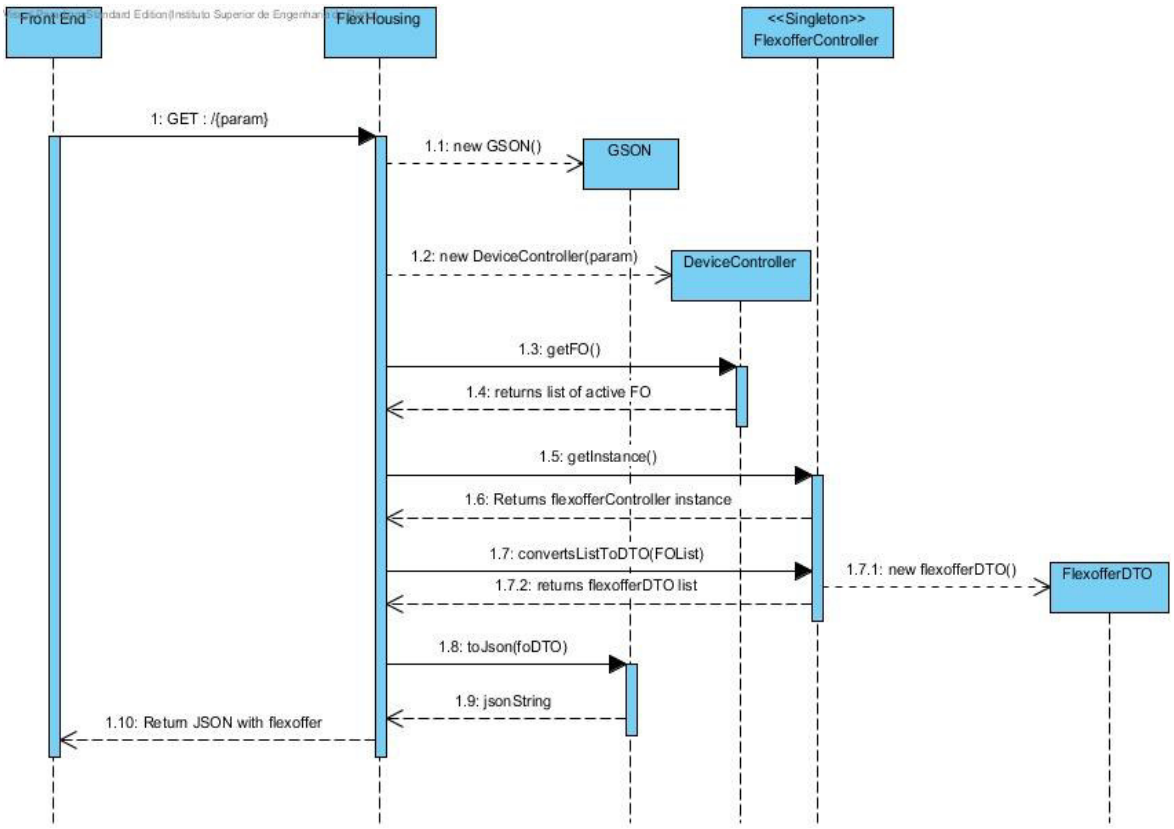

Figure 50 - Sequence diagram for getFlexoffer

Figure 50 represents the sequence diagram for the retrieval of a specific flexoffer

2.2.4 GET SCHEDULE {PARAM}

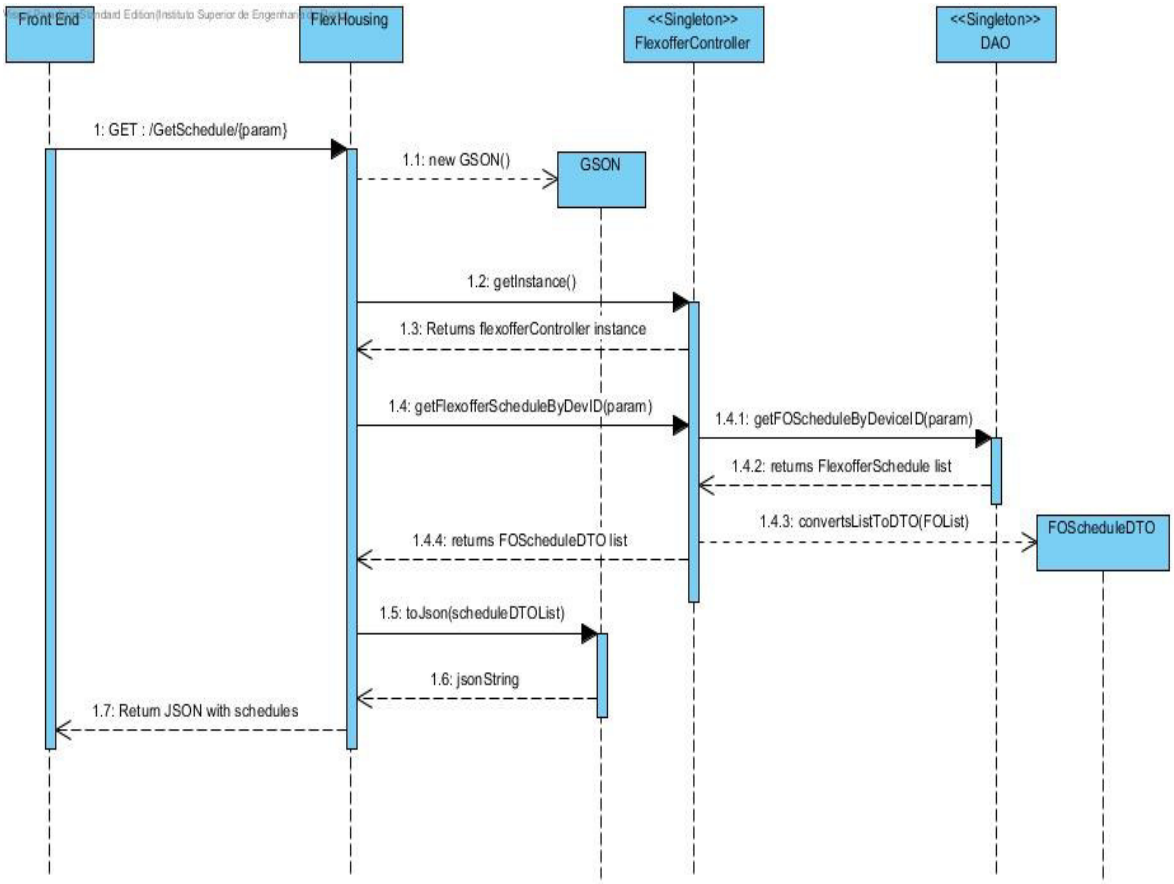

Figure 51 - Sequence diagram for getSchedule

Figure 51 represents the sequence diagram for the retrieval of a specific schedule.

2.2.5 GET DAY {PARAM}

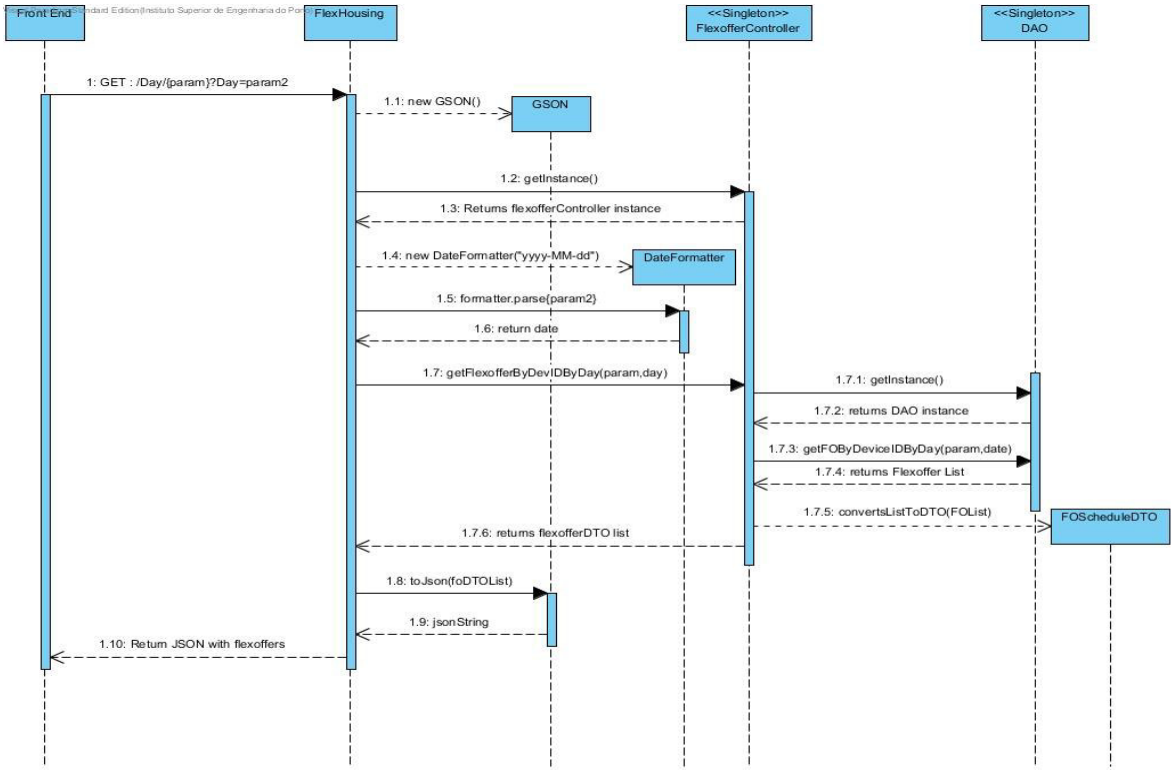

Figure 52 - Sequence diagram for getFObyDay

The diagram depicted in figure 52 represents the steps taken in reiving the flexoffers for a specific day.

2.2.6 GET SCHEDULE DAY {PARAM}

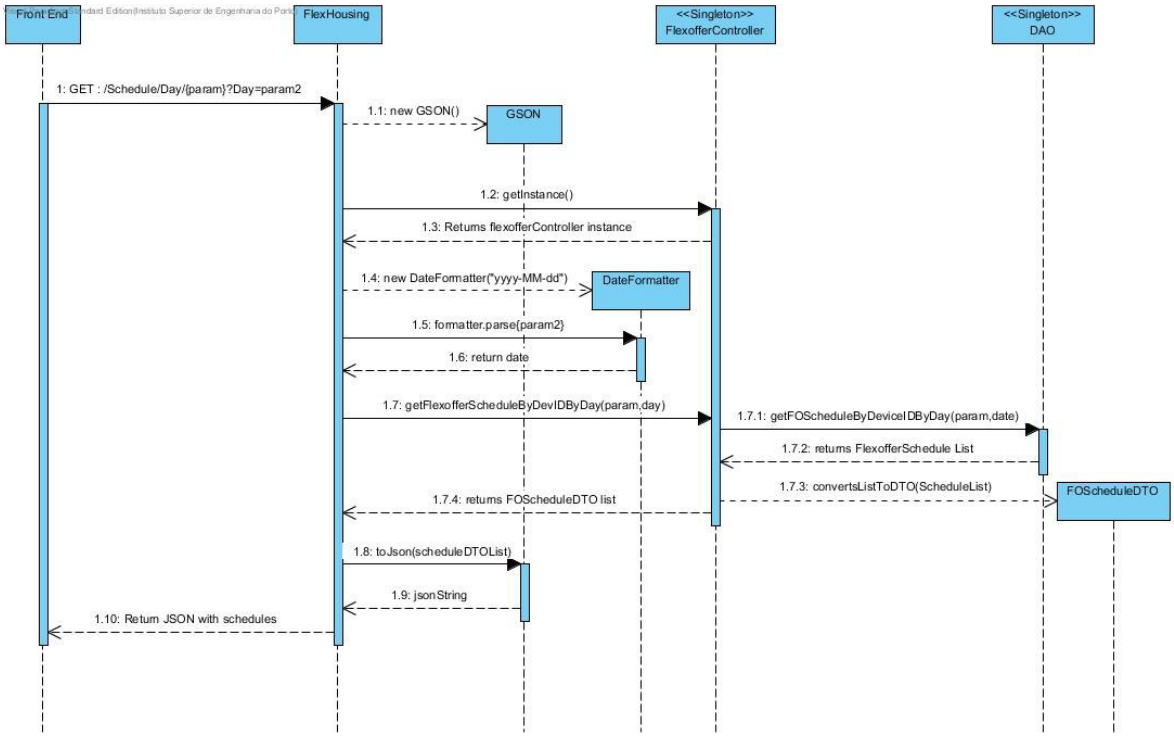

Figure 53 - Sequence diagram for getSchedulesByDay

Figure 53 depicts the sequence diagram for the retrieval of the schedules for a specific day.

2.2.7 POST {PARAM}

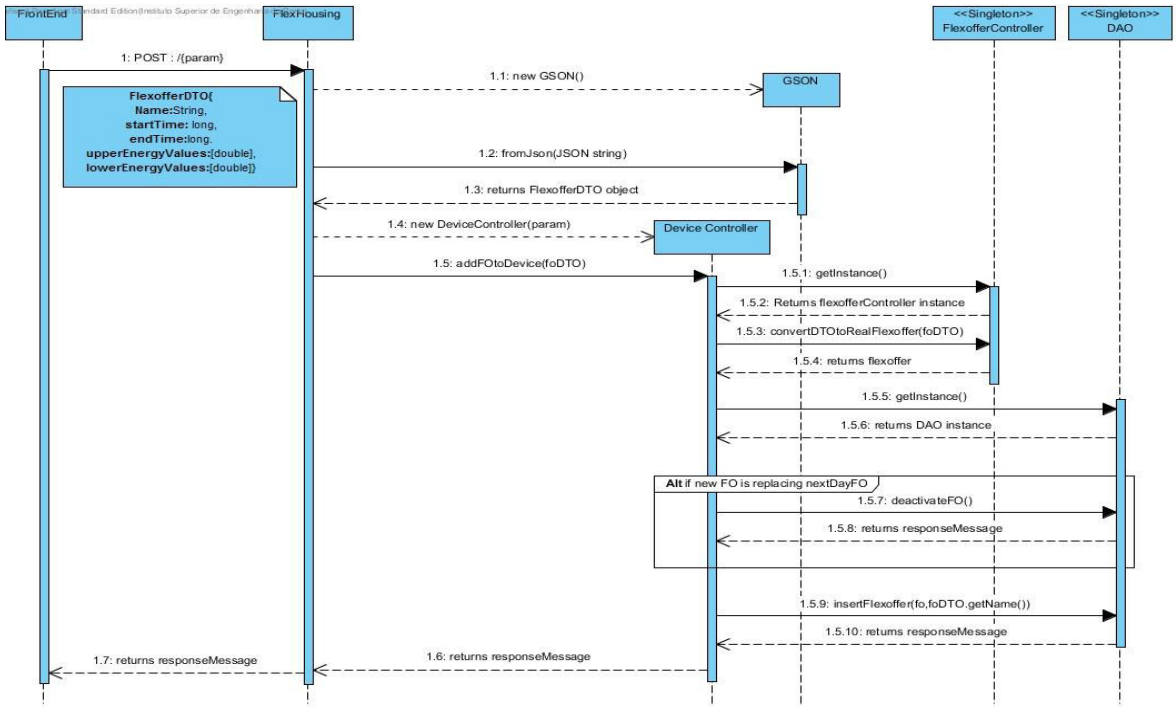

Figure 54 - Sequence diagram for posting a flexoffer

Figure 54 depicts the sequence diagram for the registering of a flexoffer in the system.

2.2.7 DELETE

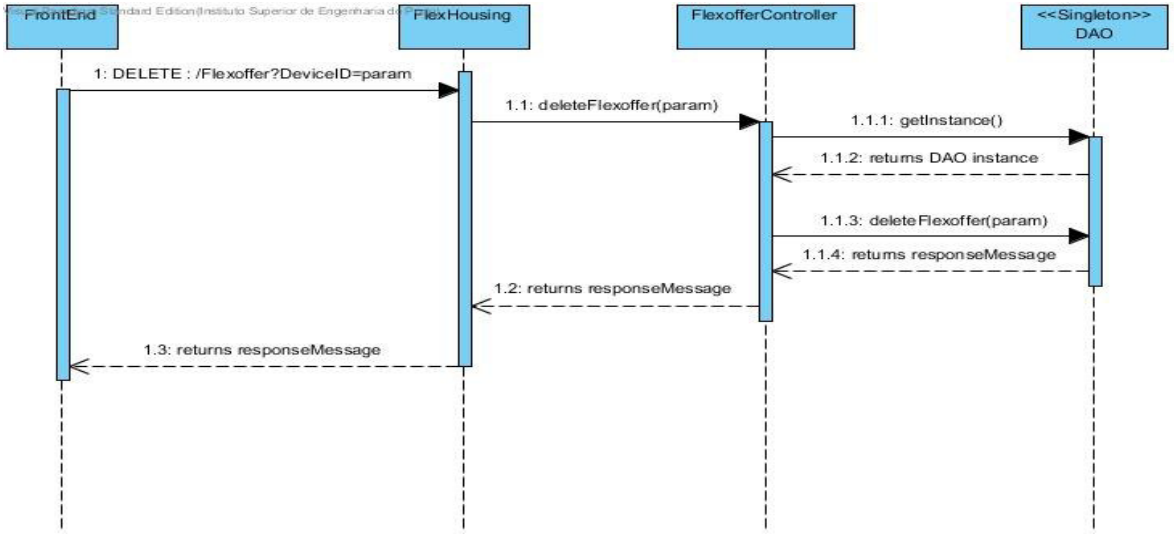

Figure 55 - Sequence diagram for deleting a flexoffer

Figure 55 represents the steps taken when deleting a flexoffer.

2.3 HOUSE

This interface is used to interact with the devices in the building and the login against the VPS services, the latter being required to be able to manage devices from an actuation and measurement retrieval point of view.

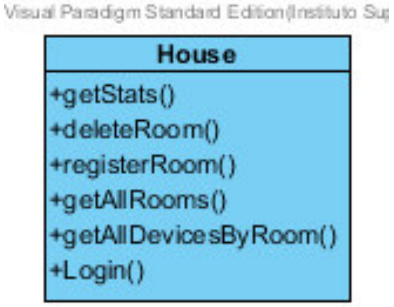

Figure 3 – Flexoffer Interface

2.3.1 GET STATS

The *stats* function is used for the retrieval of statistics relevant to the VPS system, namely the number of flexoffer applied on devices, the total number of flexoffers that had been applied and a value representing the money saved using the FlexHousing service.

Table 41 - Example of a request to get the statistics of a house

PATH	
/FlexHousing/House/Stats	
Request	Response
Path Parameters	Body

None	<pre>{
	"numberOfDevicesWithFO": 1,
	"numberOfFOApplied": 24,
	"moneySaved":
	1.2000000000000002
	}</pre>

Query Parameters	
None	
Body	
None	

2.3.2 DELETE ROOM

As the name implies, it allows the deletion of a room, specified in a query parameter, which has to be the room ID.

Table 42 - Example of a request to delete a room

PATH	
/FlexHousing/House/Room?RoomID=24	
Request	Response
Path Parameters	Body
None	<pre>ResponseCode(200)</pre>
Query Parameters	
RoomID – “24”	
Body	
None	

2.3.3 POST ROOM

This function allows the creation and addition of the room with the name in the body of the request. The service will automatically assign a unique ID when the room is added to the list. The ID is obtainable when requesting the details of the room.

Table 43 - Example of a request to register a room

PATH	
/FlexHousing/House/Room	
Request	Response
Path Parameters	Body
None	ResponseCode(200)
Query Parameters	
None	
Body	
{ "Name": “test” }	

2.3.4 GET ROOM {PARAM}

When consumed, this function will return all the devices that are currently in the room whose name was sent as a path parameter.

Table 44 - Example of a request to retrieve a specific room

PATH	
/FlexHousing/House/Room/24	
Request	Response
Path Parameters	Body
RoomID – “24”	[{ "ID": "3ZU-VGC-N3J-NWK-9P", "room": { "Name": "Sala", "ID": 24 }, "Name": "test", "sensors": [{ "id": 9454, "Name": "Active Power" }] }]
Query Parameters	
None	
Body	
None	

2.3.5 GET ROOM

This function allows for the retrieval of every room in the system. The returned data are a list of rooms comprising name and unique ID.

Table 45 - Example of a request to get all the rooms

PATH	
/FlexHousing/House/Room	
Request	Response
Path Parameters	Body

None	[
	{
	"Name": "Sala",
	"ID": 24
	},
	{
	"Name": "Quarto Principal",
	"ID": 25
	},
	{
	"Name": "Garagem",
	"ID": 26
	},
	{
	"Name": "Cozinha",
	"ID": 27
	},
	{
	"Name": "Sala de Estar",
	"ID": 28
	},
	{
	"Name": "Teste",
	"ID": 29
	}
	]

Query Parameters	
None	
Body	
None	

2.3.6 POST LOGIN

This function accepts a body with an email address and a password. The system will then communicate with the VPS services in order to verify if the email/password combination that was sent is registered in their system. If the response is a success code, it will also include an authorization token, used in any operation involving their services.

Table 46 - Example of a request to execute the login

PATH	
/FlexHousing/House/Login	
Request	Response
Path Parameters	Body
None	ResponseCode(200)
Query Parameters	
None	
Body	
{ "Login":1120527@isep.ipp.pt, "Password":"cister"}	

2.4 DIAGRAMS

2.4.1 GET STATS

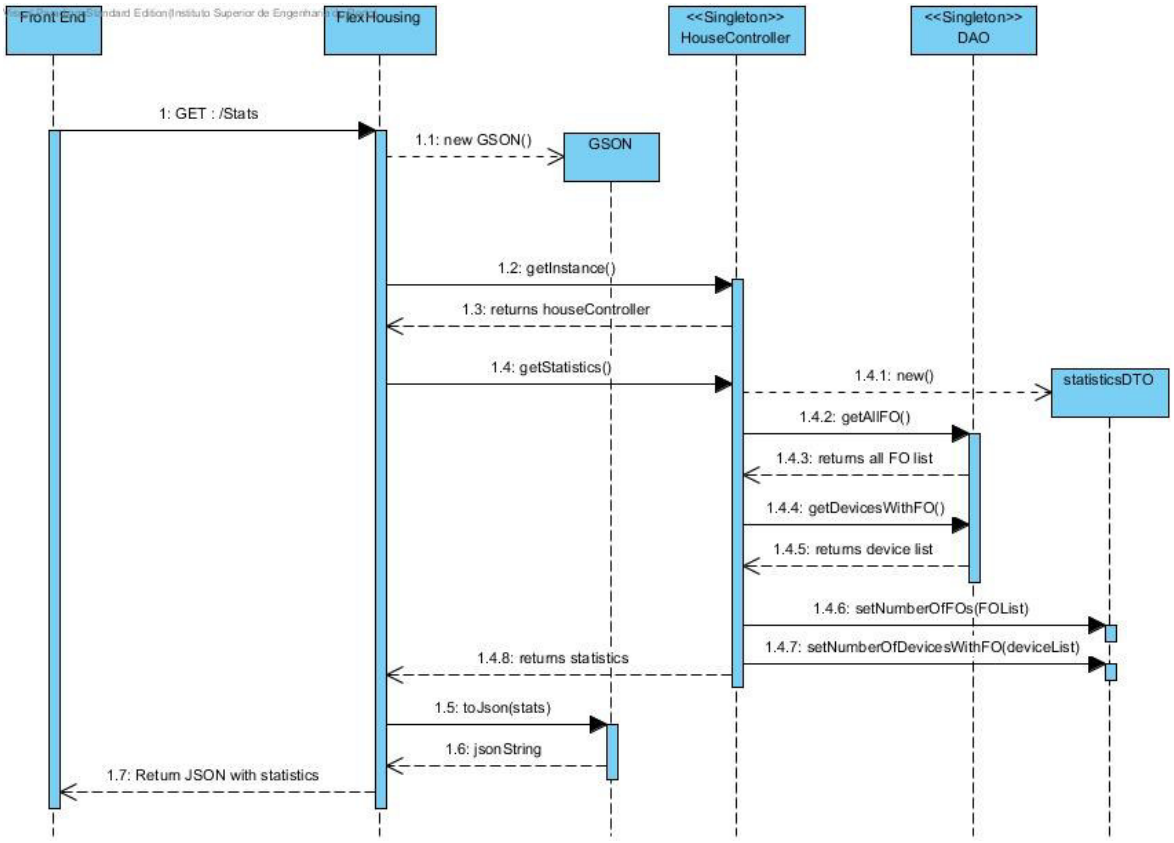

Figure 56 - Sequence diagram for retrieving the statistics of the house

Figure 56 depicts the sequence diagram for retrieval of the statistics of the house.

2.4.2 DELETE ROOM

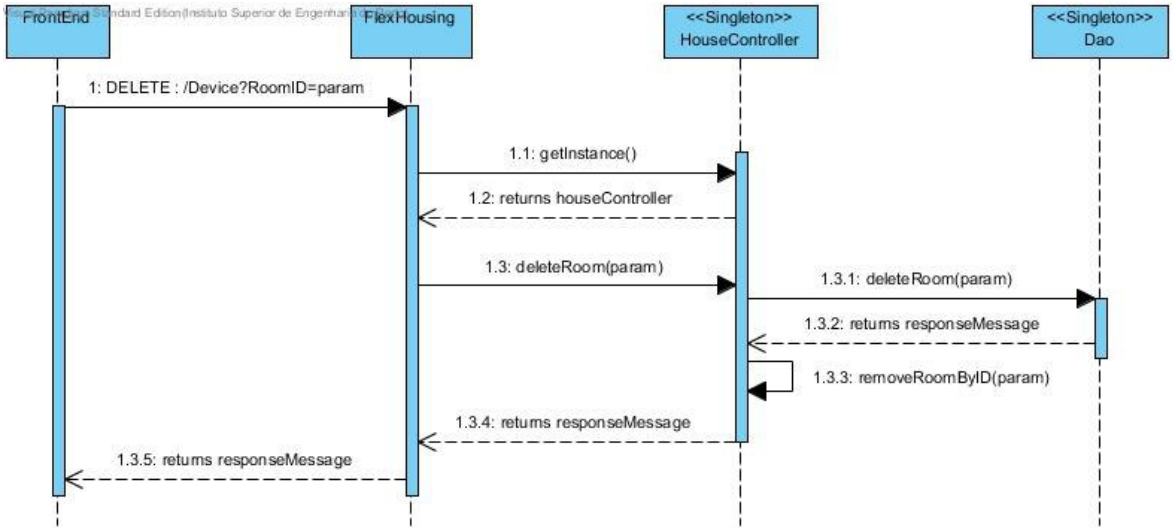

Figure 57 - Sequence diagram for deleting a room

Figure 57 depicts the sequence diagram for deleting a room.

2.4.3 POST ROOM

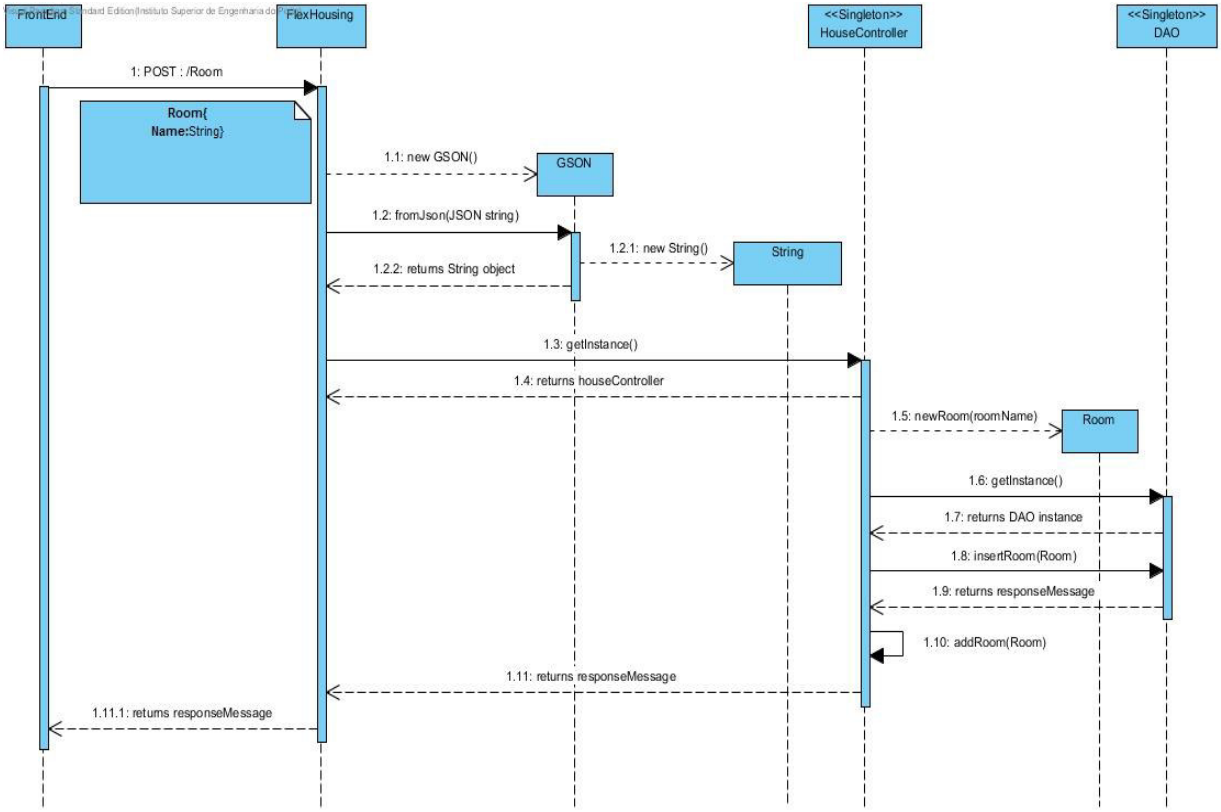

Figure 58 - Sequence diagram for registering a room

Figure 58 depicts the sequence diagram for insertion of a room into the system.

2.4.4 GET ROOM {PARAM}

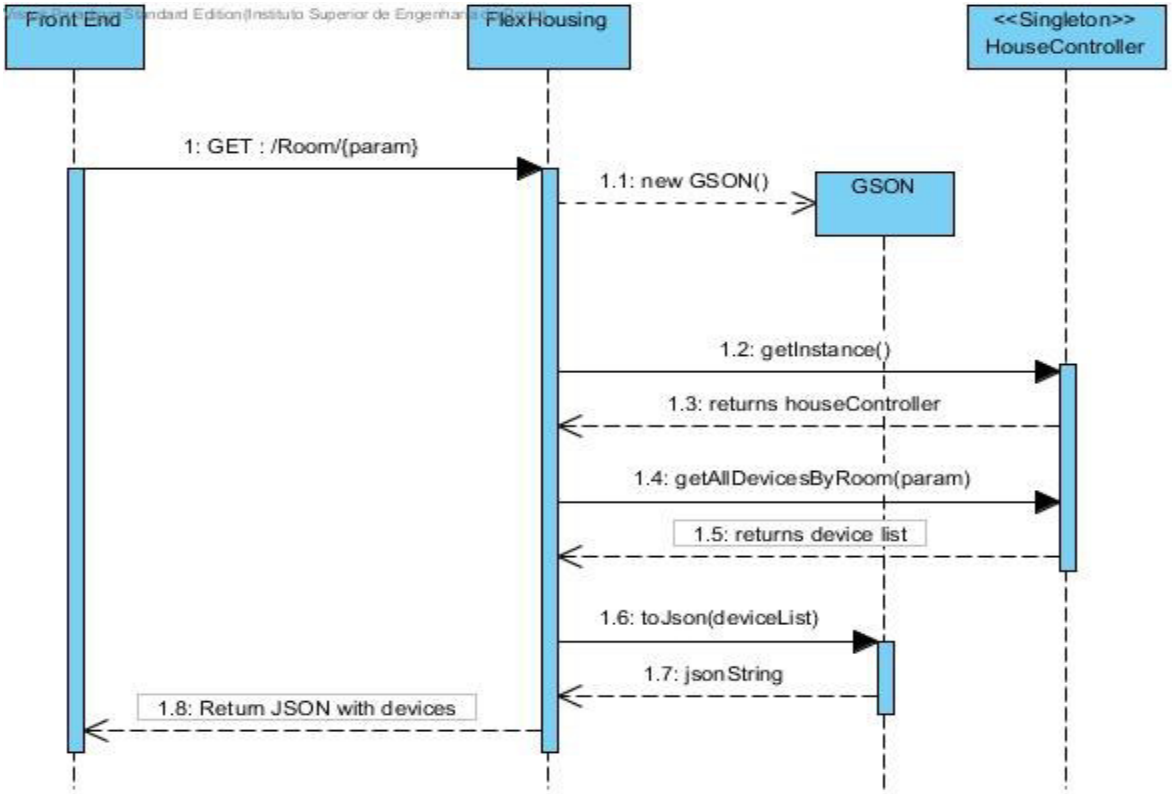

Figure 59 - Sequence diagram for retrieving all the devices in a room

Figure 59 depicts the sequence diagram for obtaining the devices attached to a room.

2.4.5 GET ROOM

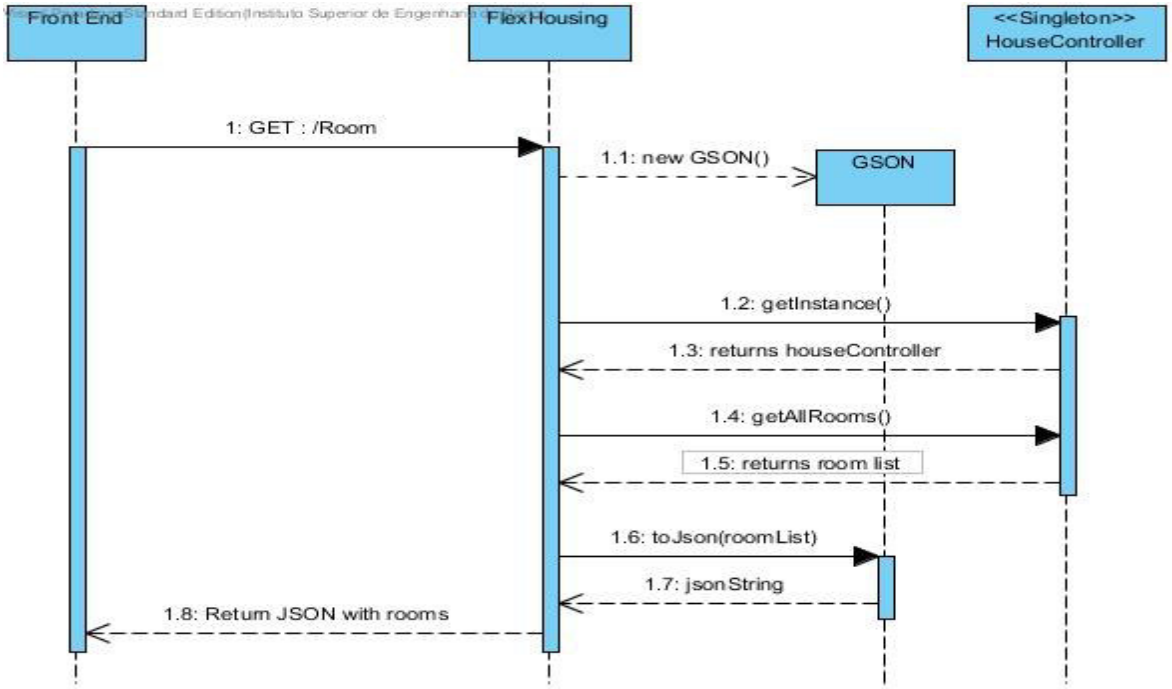

Figure 60 - Sequence diagram for retrieving the details for a room

Figure 60 depicts the sequence diagram for the retrieval of the details of a room.

2.4.6 LOGIN

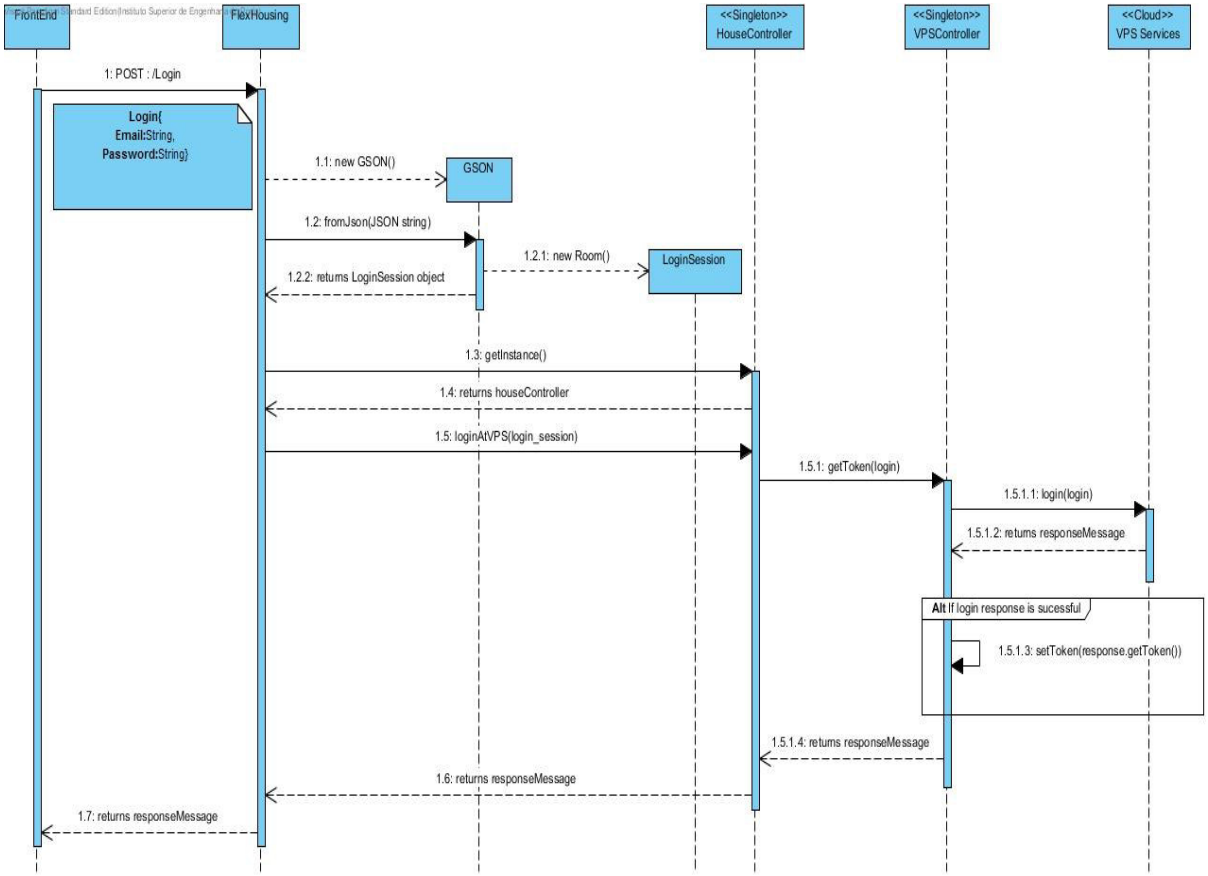

Figure 61 - Sequence diagram for the login in the system

Figure 61 depicts the sequence diagram the steps executed when the Login request is received

2.5 DEVICE

This interface gives the possibility managing the devices: register, update, check details, register sensors and delete them. Any function that interacts with the VPS services will fail if the authorization step wasn't previously executed. The sensors are embedded in the devices but for the sake of usage, they need to be added to the FlexHousing system. The methods attached to interface are depicted in Figure 62.

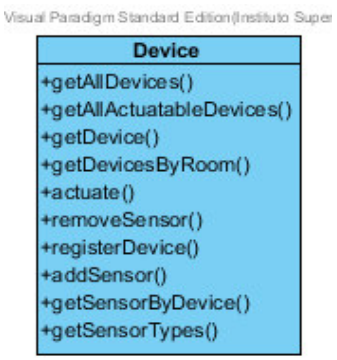

Figure 62 - Flexoffer Interface

2.5.1 GET

The function with the GET request at the same path as the interface(/Device) returns a list with all the devices currently registered in the application.

Table 47- Example of a request to retrieving all the devices from the system

PATH	
/FlexHousing/Device	
Request	Response
Path Parameters	Body

None	<pre>[
	  {
	  "ID": "3ZU-VGC-N3J-NWK-9P",
	  "room": {
	  "Name": "Sala",
	  "ID": 24
	  },
	  "Name": "test",
	  "sensors": [
	  {
	  "id": 9454,
	  "Name": "Active Power"
	  }
	  ]
	  }
	]</pre>

Query Parameters	
None	
Body	
None	

2.5.2 GET GETACTUATABLE

Similar to the function described in section 2.3.1.1, with the exception of only returning devices with the actuators attached to them. All the sensors have sensors embed, but some devices are capable to actuate because they lack the actuation sensor. Such devices aren't flexoffer compliant.

Table 48- Example of a request to retrieve all the actuable devices

PATH	
/FlexHousing/Device/GetActuatable	
Request	Response
Path Parameters	Body

None	<pre>[
	  {"ID": "3ZU-VGC-N3J-NWK-9P",
	  "room": {</pre>

Query Parameters	
None	
Body	

None	<pre>"Name": "Sala",
	"ID": 24
	},
	"Name": "test",
	"sensors": [
	{
	"id": 9454,
	"Name": "Active Power"
	},
	{
	"id": 9456,
	"Name": "Actuator"}]
	}
	]</pre>

2.5.3 GET {PARAM}

Invoking this function with the interface path with and an extra path parameter will return the details of the device whose ID is in the said parameter. This will return name, ID, Room and sensors currently linked to that specific device.

Table 49- Example of a request to retrieve a specific device

PATH	
/FlexHousing/Device/3ZU-VGC-N3J-NWK-9P	
Request	Response
Path Parameters	Body

DeviceID – “3ZU-VGC-N3J-NWK-9P”	<pre>{
	"ID": "3ZU-VGC-N3J-NWK-9P",
	"room": {
	"Name": "Sala",
	"ID": 24
	},
	"Name": "test",
	"sensors": [
	{
	"id": 9454,
	"Name": "Active Power"
	},
	{
	"id": 9456,
	"Name": "Actuator"
	}
	]
	}</pre>

Query Parameters	
None	
Body	
None	

2.5.4 POST ACTUATE

This function is the backbone of the FlexHousing system: it allows to issue a command to a specific device. The body of the request contains the ID of the device and the command to be issued: 0 for turning off, 1 for turning on and 3 for commutation (from 1 to 0 or vice versa). When received, the request will trigger FlexHousing into sending a second request to the VPS services, issuing the demanded command to that specified device. This function can be used with any device, but will only have an effect with devices equipped with actuators.

Table 50 - Example of a request to request an actuation on a device

PATH	
/FlexHousing/Device/Actuate	
Request	Response
Path Parameters	Body
None	`ResponseCode(200)`
Query Parameters	
None	
Body	

{
"ID":"3ZU-VGC-N3J-NWK-9P",
"command":3
}	

2.5.5 DELETE SENSOR

This function is used when a sensor is to be deleted. The request is sent specifying the device ID and sensor ID, representing which sensor to delete from the system

Table 51 - Example of a request to delete a sensor

/FlexHousing/Device/Sensor?DeviceID=3ZU-VGC-N3J-NWK-9P&SensorID=9454	
Request	Response
Path Parameters	Body
None	`ResponseCode(200)`
Query Parameters	
DeviceID – “3ZU-VGC-N3J-NWK-9P” SensorID – “9454”	
Body	
None	

2.5.6 POST

Using this function, attached to path of the interface, allows the user to register a device. The only information the user has to input is the ID of the device, normally placed directly on the device, and a name, used for a human readable traceability within the system. FlexHousing is capable to link the device to its sensors and measurements with the physical ID of the device.

Table 52 - Example of a request to register a device

PATH	
/FlexHousing/Device	
Request	Response
Path Parameters	Body
None	ResponseCode(200)
Query Parameters	
None	
Body	

{
"ID": "3ZU-VGC-N3J-NWK-9P",
"room": {
"Name": "Sala",
"ID": 24
},
"Name": "test"
}	

2.5.7 POST SENSOR

This function allows for the attachment of sensors to the device. Essentially the sensors are already attached to the device but only in the VPS services and have their own local ID. This function retrieves the ID of that tag and allow the traceability and responsiveness of the said sensor, for either measurements or in more specific case, the actuation. The body of the request contains the name of the sensor meant to be added and the ID of the target device.

Table 53 - Example of a request to register a sensor

PATH	
/FlexHousing/Device/Sensor	
Request	Response
Path Parameters	Body
None	ResponseCode(200)
Query Parameters	
None	
Body	
{ "ID": "3ZU-VGC-N3J-NWK-9P", "Name": "Active Power"}	

2.5.8 GET SENSOR

The usage of this function allows the user to retrieve the sensor currently attached to a device, specified by its ID, attached by a query parameter

Table 54 - Example of a request to retrieve the details of a sensor

PATH	
/FlexHousing/Device/Sensor/3ZU-VGC-N3J-NWK-9P	
Request	Response
Path Parameters	Body

DeviceID – "3ZU-VGC-N3J-NWK-9P"	<pre>[
	  {
	  "id": 9454,
	  "Name": "Active Power"
	  },
	  {
	  "id": 9456,
	  "Name": "Actuator"
	  }
	]</pre>

Query Parameters	
None	
Body	
None	

2.5.9 GET SENSOR TYPES

This function enables the user to check which kind of sensors is available to be added. It returns a list of strings, containing the names of the sensors.

Table 55 - Example of a request to retrieve the sensor types in the system

PATH	
/FlexHousing/Device/Sensor/Types	
Request	Response
Path Parameters	Body

None	<pre>[
	  "Voltage RMS",
	  "Frequency",
	  "Active Power",
	  "Active energy+",
	  "Actuator",
	  "Power Factor",
	  "Power Factor",
	  "RSSI",
	  "LQI"
	]</pre>

Query Parameters	
None	
Body	
None	

2.6 DIAGRAMS

2.6.1 GET

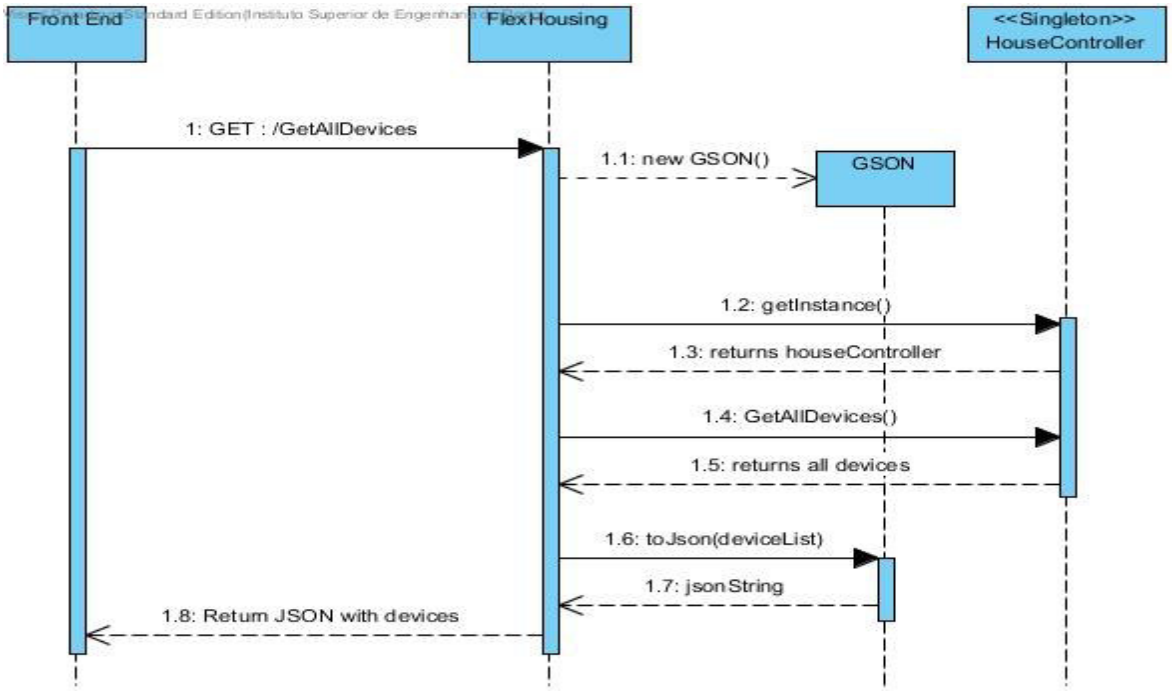

Figure 63 - Sequence diagram for retrieving all the devices in the system

Figure 63 depicts the sequence diagram for retrieving the device that are registered in the system.

2.6.2 GET GETACTUATABLE

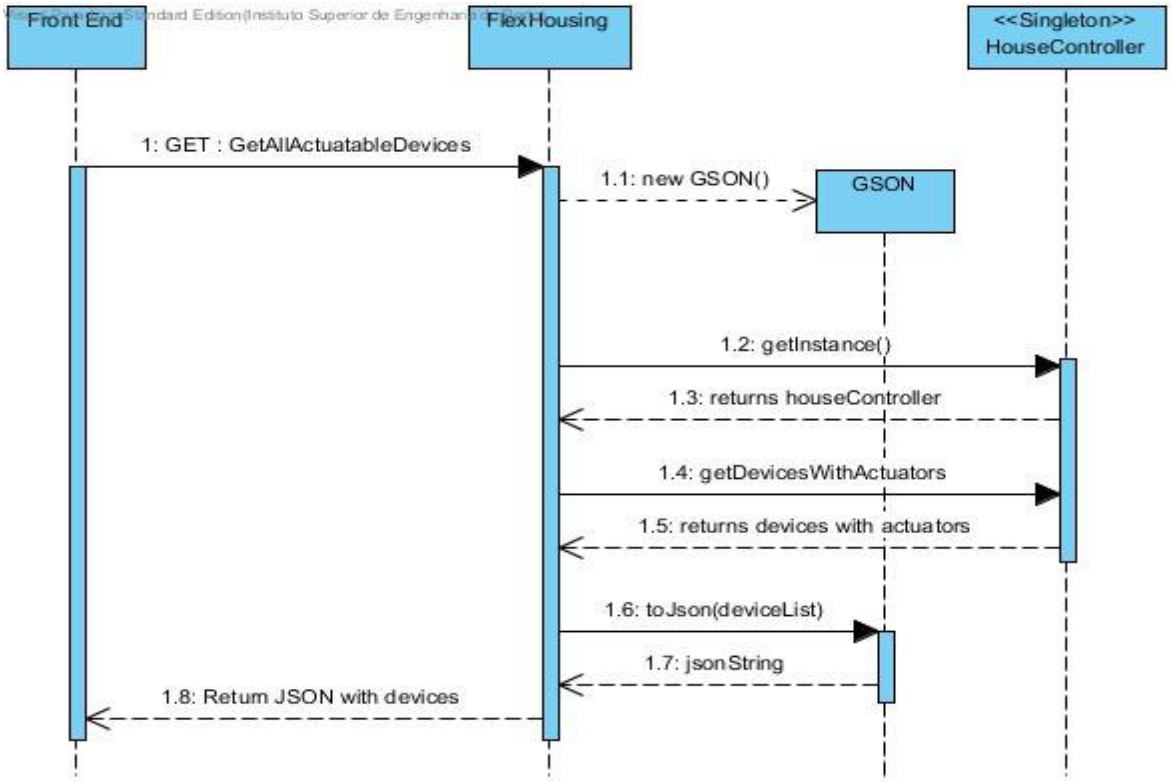

Figure 64 - Sequence diagram for retrieving all the actuable devices

Figure 64 depicts the sequence diagram for retrieving the devices that have the Actuator sensor linked to them.

2.6.3 GET {PARAM}

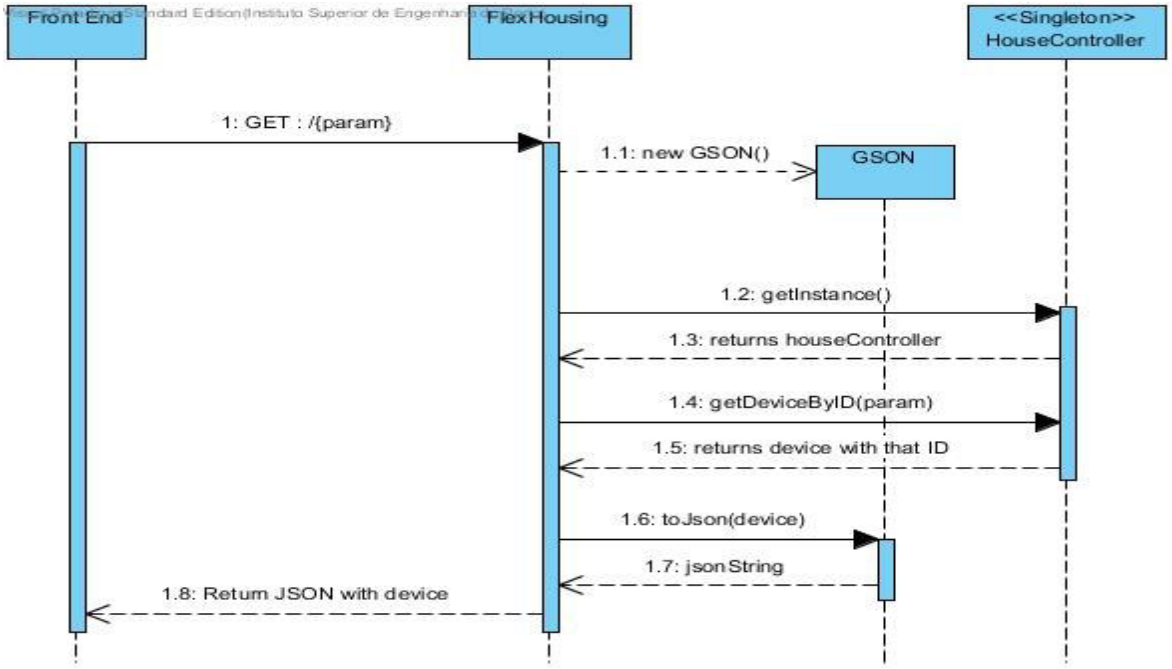

Figure 65 - Sequence diagram for retrieving the details for a specific device

Figure 65 depicts the sequence diagram for obtaining the details a device.

2.6.4 POST ACTUATE

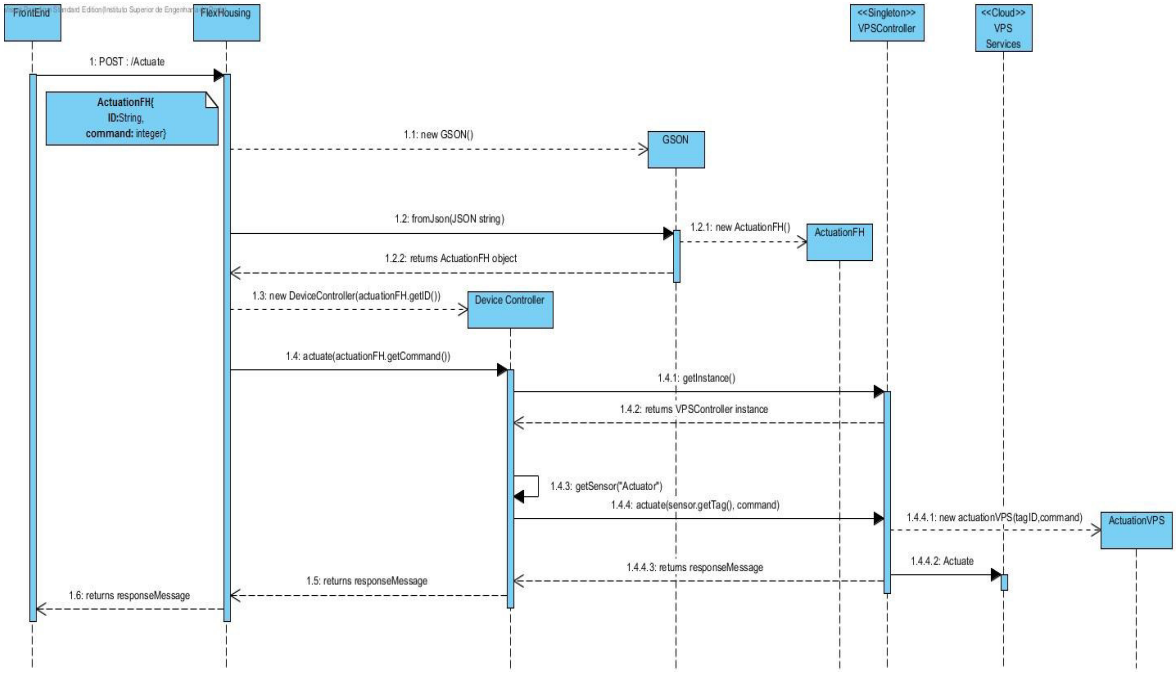

Figure 66 - Sequence diagram for requesting an actuation on a device

Figure 66 depicts the sequence diagram for the operations executed when a request for actuating on a device is received.

2.6.5 DELETE SENSOR

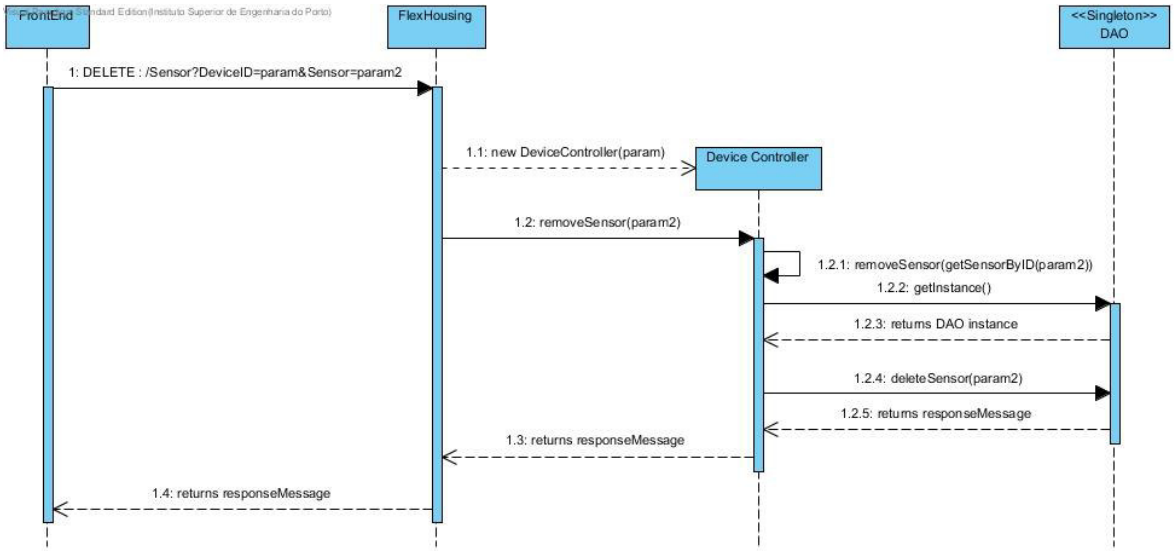

Figure 67 - Sequence diagram for deleting a sensor

Figure 67 depicts the sequence diagram for the deletion of a sensor.

2.6.6 POST

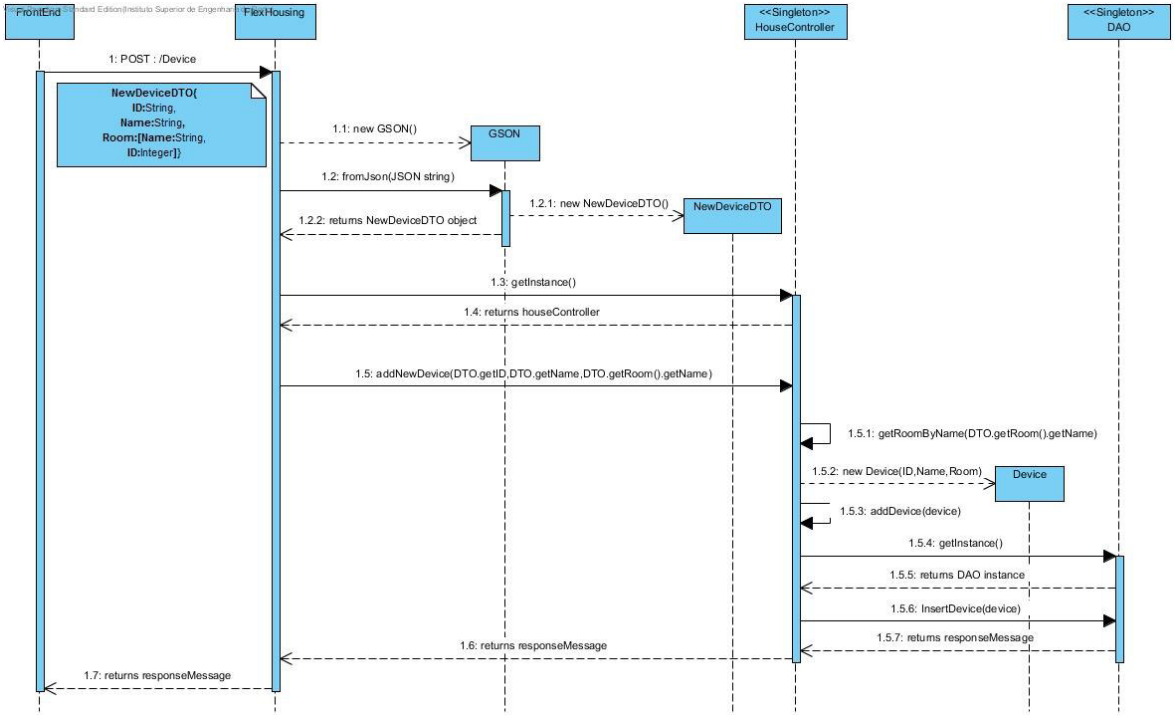

Figure 68 - Sequence diagram for registering a device

Figure 68 depicts the sequence diagram the registration of a device into the FlexHousing system.

2.6.7 POST SENSOR

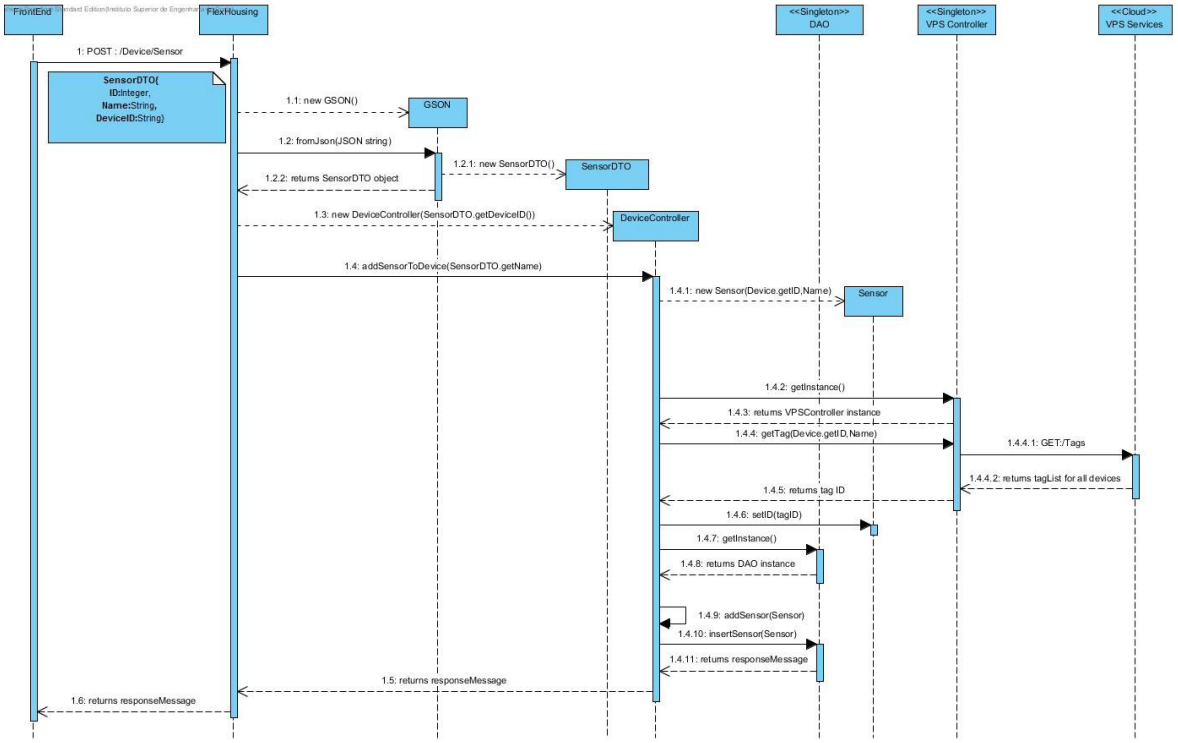

Figure 69 - Sequence diagram for adding a sensor in the system

Figure 69 depicts the sequence diagram for adding a sensor to the FH system.

2.6.8 GET SENSOR

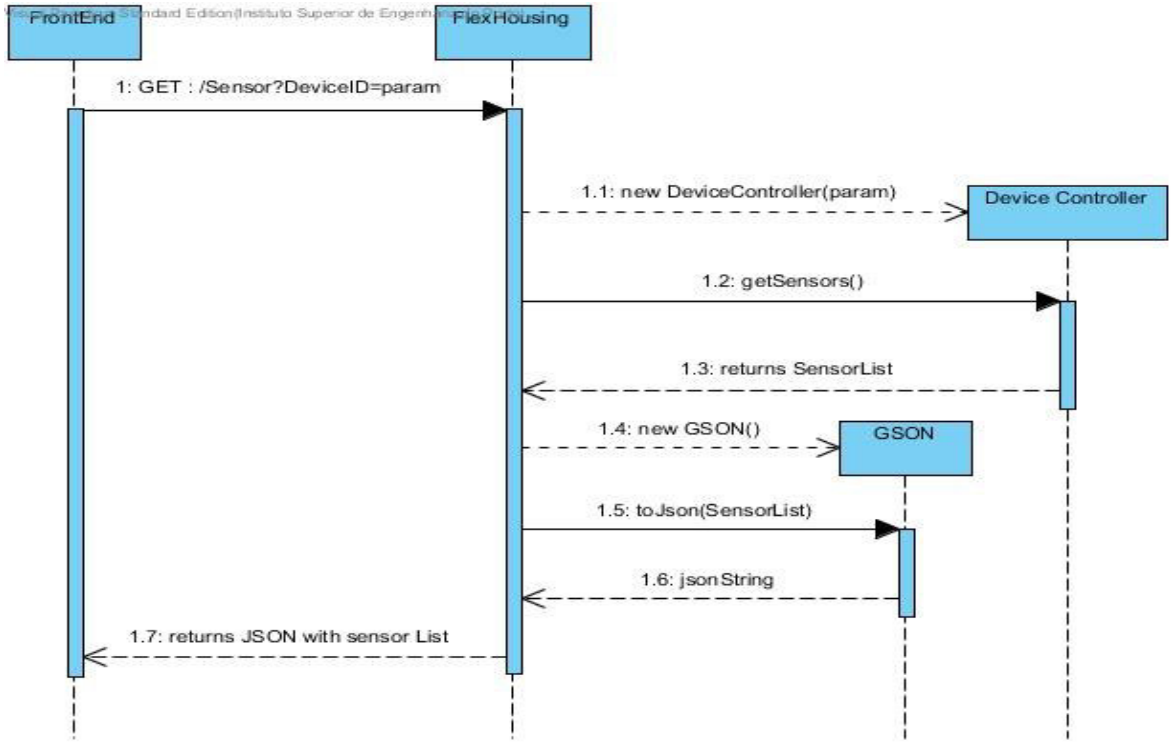

Figure 70 - Sequence diagram for obtaining the details for a sensor

Figure 70 depicts the sequence diagram for retrieving the details from a sensor.

2.6.9 GET SENSOR TYPES

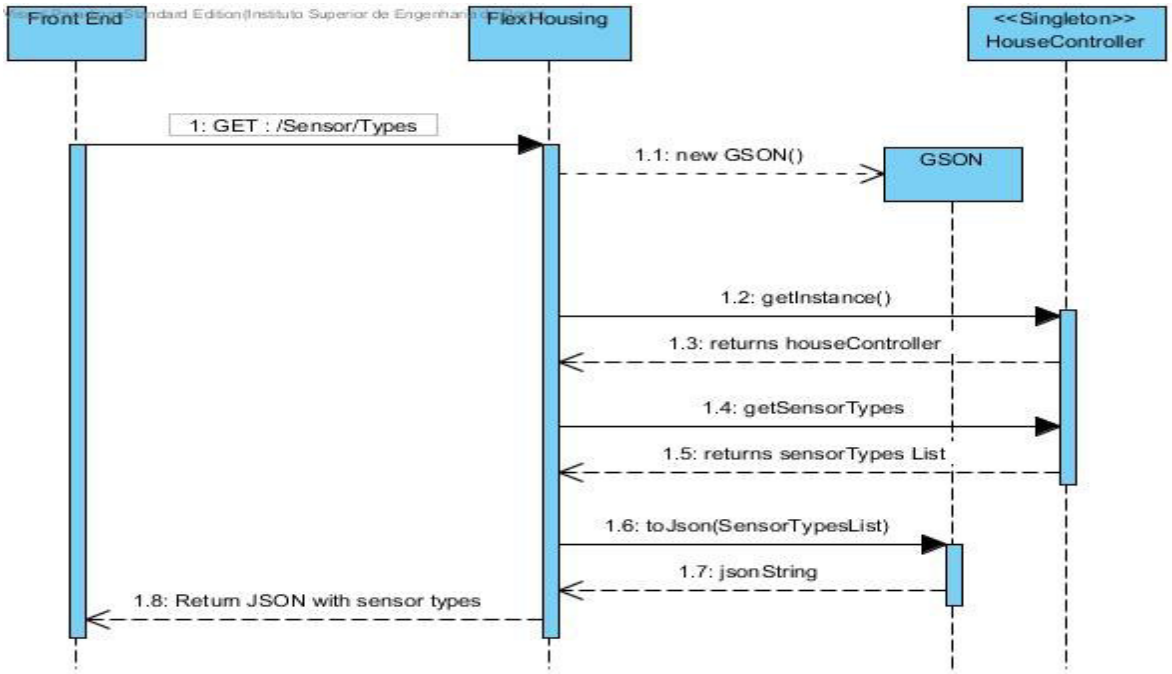

Figure 71 - Sequence diagram for retrieving the types of sensors from the system

Figure 71 depicts the sequence diagram retrieving the different types of sensors.

2.7 MEASUREMENTS

This interface is used to query the measurements of the sensors embedded in the devices. Used for verification of values, flexoffer establishment and to check if a flexoffer was respected energy-wise. The services accept GET and DELETE operations, and their path have query parameters acting as filters. Figure 72 depicts the methods attached to the Measurements interface.

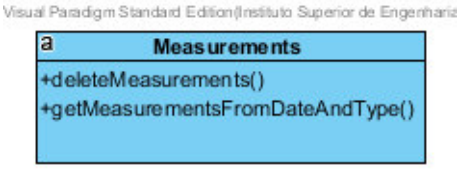

Figure 72 - Measurements Interface

2.7.1GET

The GET function applied to a path allows the user to retrieve information about the measurements from a device's sensor. Every sensor has measurements although they can in specific cases only return some stock values (The actuator will only return 0 or 1, depending whether the device was actuated during that time). The measurements are stored at VPS servers, so the FlexHousing system acts as an intermediate, with the option of storing specific information locally if the user desires.

Table 56 - - Example of a request to retrieve the data collected by a sensor between dates

PATH	
/FlexHousing/Measurements?DeviceID=3ZU-VGC-N3J-NWK-9P&from=1473951600000&to=1473962400000&Sensor=Active+Power	
Request	Response
Path Parameters	Body
None	[{ "Measuring": "Active Power", "Value": 0.0178, "Date": "Sep 15, 2016 3:00:00 PM" }, { "Measuring": "Active Power", "Value": 0.0204, "Date": "Sep 15, 2016 3:15:00 PM" }, { "Measuring": "Active Power", "Value": 0.0218, "Date": "Sep 15, 2016 3:30:00 PM" }, { "Measuring": "Active Power", "Value": 0, "Date": "Sep 15, 2016 3:45:00 PM" }, …..
Query Parameters	
DeviceID – “3ZU-VGC-N3J-NWK-9P” From – “1473951600000” To – “1473962400000” Sensor – Active Power	
Body	
None	

2.7.2 DELETE

The DELETE function allows the user to clear the database of the measurements that were stored locally. The query parameters allow the system to pinpoint from which sensor to delete the information and of which device

Table 57- Example of a request to delete the data collected by a sensor

PATH	
/FlexHousing/Measurements?DeviceID=3ZU-VGC-N3J-NWK-9P&Type=Active+Power	
Request	Response
Path Parameters	Body
None	ResponseCode(200)
Query Parameters	
DeviceID – “3ZU-VGC-N3J-NWK-9P”	

Type – “Active Power”	
Body	
None	

2.8 DIAGRAMS

2.8.1 GET

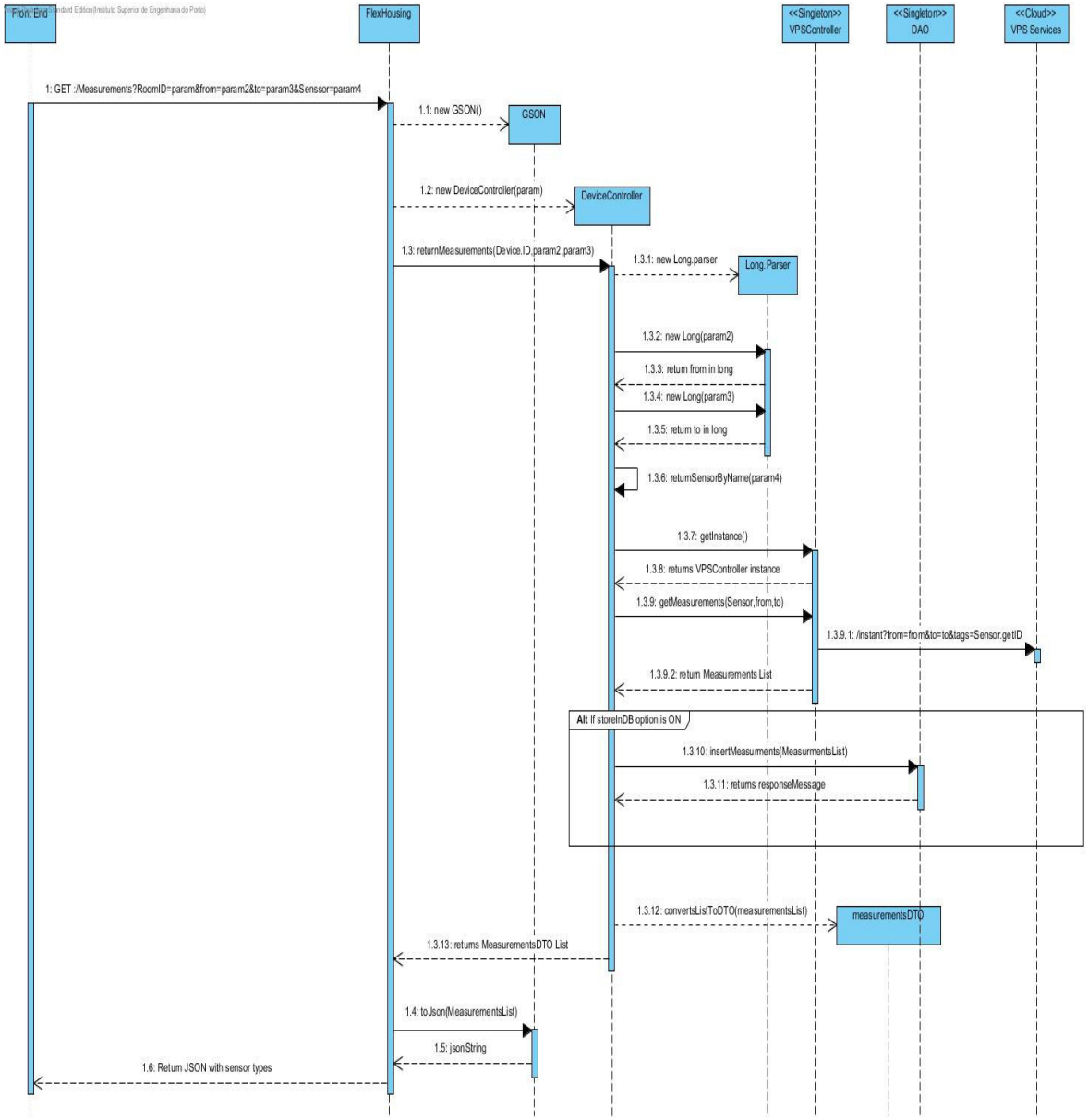

Figure 73 - Sequence diagram for retrieving the measurements collect between 2 dates

Figure 73 depicts the sequence diagram for gathering the measurements from a sensor for a time period

2.8.2DELETE

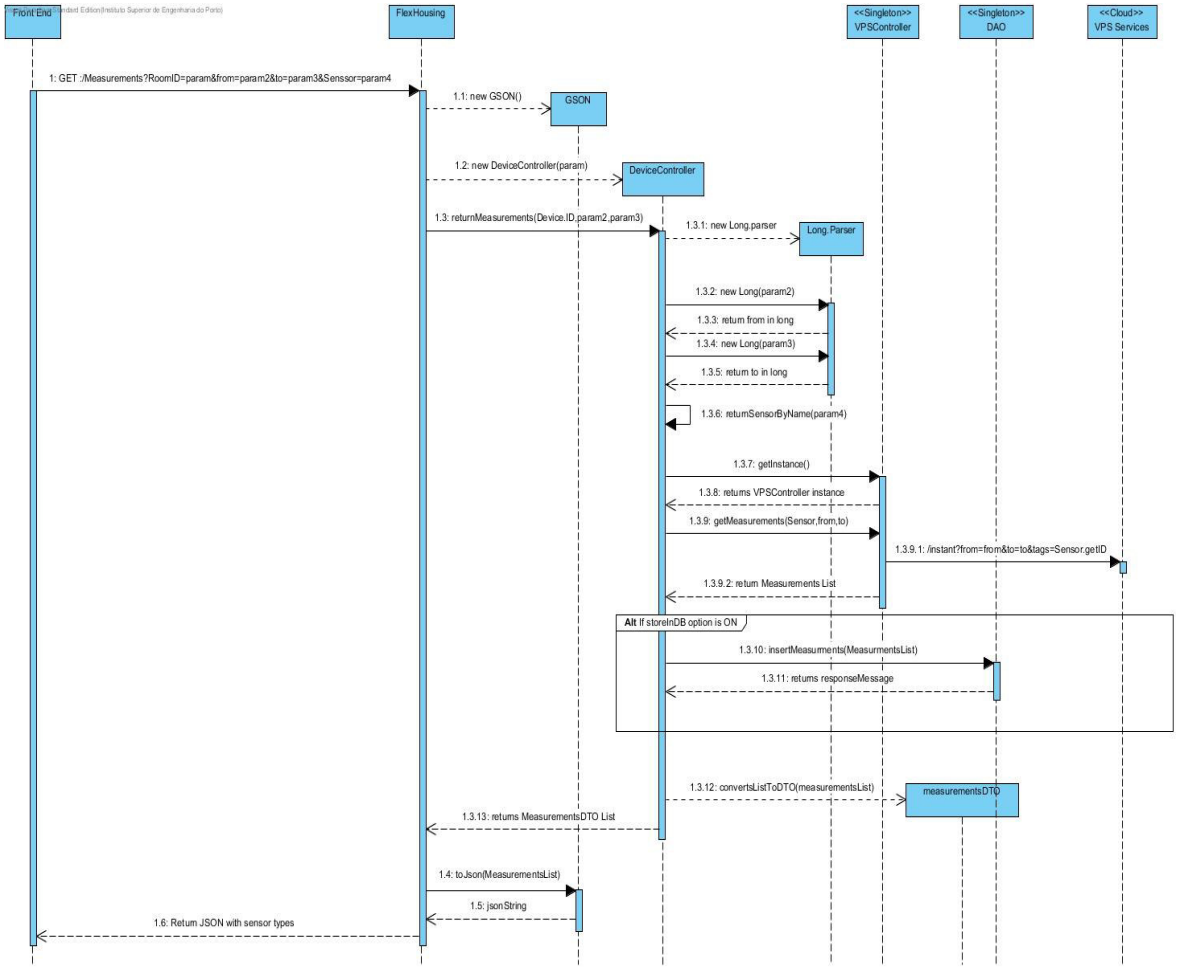

Figure 74- Sequence diagram for deleting the measurements from a sensor

Figure 74 depicts the sequence diagram for the deletion of the measurements stored in the middleware from a device and type of sensor.

3. INFORMATION MODEL

This section explains what each parameter and field represent for each interface of the FlexHousing service.

3.1 FLEXOFFER

Table 58 - Description of the parameters used in the flexoffer interface

Parameter	Format	Description
DeviceID	String	The physical ID of the device
Flexoffer	FlexofferDTO	Contains the payload of the flexoffer
Schedule	ScheduleDTO	Contains the information about the schedule
Day	yyyy-MM-dd	The date of a certain day
startTime/endTime	long	A specific moment in time in milliseconds since 1st of January 1970

upper/lowerEnergyValues	List of Doubles	Each double represents an amount of energy in kW
Name	String	Name of the flex. Normally allusive to the device it was applied to.

3.2 HOUSE

Table 59 - Description of the parameters used in the house interface

Parameter	Format	Description
RoomID	Integer	The local ID of the room
DeviceID	String	The physical ID of the device
Email	Email (aaaaa@aaaa.aa)	The email use to register at the VPS services
Password	String	The password use in conjuction with the email for the VPS account
Room	Room	Contains the details of the room
Device	NewDeviceDTO	Contains the details of the device
Name	String	Sent to create a room with that name

3.3 DEVICE

Table 60 - Description of the parameters used in the device interface

Parameter	Format	Description
DeviceID	String	The physical ID of the device
Device	NewDeviceDTO	Contains the details of the device
Sensor	SensorDTO	Contains information about the sensor
ActuationFH	ActuationFH	Contains the ID and command of the actuation
SensorTypes	List of String	Represents the List of names of the sensors that may be present in the devices
SensorID	Integer	Represents the ID of sensor using the VPS local ID

3.4 MEASUREMENTS

Table 61 - Description of the parameters used in the measurement interface

Parameter	Format	Description
DeviceID	String	The physical ID of the device
Sensor	String	Name of the sensor
from/to	long	A specific moment in time in milliseconds since 1st of January 1970

4.3 TESTING

The realization of software tests is crucial to ensure that it meets all of its requirements and has the expected behavior. This process should be incremental and implemented alongside the project. If some errors aren't picked up earlier on, it might lead to some serious flaws [32].

In fact, there are two different approaches of development tests, "Black-Box" and "White-Box" testing. "Black-Box" testing is done without knowing the internal parts of the software, the tester knows what the program should do but does not know how it works. This testing approach applies in different levels of software testing such as integration testing, system testing or acceptance testing. Whereas the White-Box testing focus on the internal parts of the software, forcing the tester to know the structure of the program. Most commonly, the White-Box testers have programming skills and had already studied the implementation code. This testing approach applies in three different level of software testing: integration, unit and system.

For the purpose of this project, seeing that a distributed system was developed, standard Unit Testing would be lackluster. As such, a big emphasis was given to the acceptance tests [33].

4.3.1 ACCEPTANCE TESTS

Acceptance tests are developed focusing on specific scenario. Normally involving the usage of the system as a whole. Each acceptance test tests a features, and features in this case translate into Use Cases.

So for each Use Case, an acceptance test was developed.

Test for UC:	1
Goal:	Send Flexoffer
Criteria	Criteria Result
1. Input the required static information 2. Able to retrieve pattern from measurements 3. Manually create flexoffer 4. Send Flexoffer 5. Flexoffer received 6. Flexoffer stored in database	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success
Final Result	Sucess

Table 62 - Acceptance test for UC 1

Test for UC:	2
Goal:	Receive the schedules
Criteria	Criteria Result
1. Retrieve Flexoffer from database 2. Execute Flexoffer emission 3. Receive Schedule 4. Store Schedule into database 5. Ask for specific schedule 6. Retrieve schedule from database 7. Send Schedule 8. Create Chart for Schedule	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success 7. Success 8. Success
Final Result	Success

Table 63 - Acceptance test for UC 2

Test for UC:	3
Goal:	Login
Criteria	Criteria Result
1. Send credentials 2. Receive credentials 3. Send credentials to VPS Services 4. Retrieve response Token 5. Attach token to Controller 6. Return response	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success
Final Result	Success

Table 64 - Acceptance test for UC 3

Test for UC:	4
Goal:	Manage House
Criteria	Criteria Result
1. Send request for room list 2. Receive request 3. Return the room list 4. Receives room list 5. Creates new room 6. Receive new room information 7. Returns response 8. Request room details 9. Receives request 10. Return room details 11. Receives room details	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success 7. Success 8. Success 9. Success 10. Success 11. Success
Final Result	Success

Table 65 - Acceptance test for UC 4

Test for UC:	5
Goal:	Manage Device
Criteria	Criteria Result
1. Send request for device list 2. Receive request 3. Return the device list 4. Receives device list 5. Request device details 6. Receives request 7. Return device details 8. Receives device details	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success 7. Success 8. Success
Final Result	Success

Table 66 - Acceptance test for UC 5

Test for UC:	6
Goal:	Check Measurements
Criteria	Criteria Result
1. Send request 2. Receive request 3. Send request to the VPS Services 4. Receive response 5. Send measurements back 6. Receive measurements 7. Build chart for measurements	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success 7. Success
Final Result	Success

Table 67 - Acceptance test for UC 6

Test for UC:	7
Goal:	Actuate on Device
Criteria	Criteria Result
1. Send request 2. Receive request 3. Send Request to VPS services 4. Receive response 5. Return response	1. Success 2. Success 3. Success 4. Success 5. Success
Final Result	Success

Table 68 - Acceptance test for UC 7

In conclusion, one can verify that all the tests were successful. The is further evidence by the formal demonstration done in an official Arrowhead Meeting held in Turin, in October. All the features were

demonstrated to the various partners present. Dr Jerker Delsing, Director of the Artemis European project in which Arrowhead is included, was pleased by the demonstration. In addition to being one of 2 partners who had live demonstration, all the features were demonstrated successfully.

5. CONCLUSION

This chapter resumes all the developed work, taking into account what went best and worst during all the internship.

Section 5.1 does a recap of the problem and how it was dealt with, then Section 5.2 enumerates and describes the objectives that were accomplished. The limitations of the developed solution along with some future work perspectives are discussed in Section 5.3. Finally, Section 5.4 describes the opinion of the student towards the internship as a whole.

5.1 SUMMARY

This project's goal was to implement a pilot for the integration of the Flexoffer Concept. As described in section 1, the Flexoffer concept revolves around the exposure of the demands in electrical power of the users to the energy market. The offer with the flexibility the user has is sent to an aggregator, which sends back the schedule that meets the best prices (lowest price), but still matching the user's needs.

This pilot integrates with external cloud technologies from VPS, turning it into a house energy automation application. The user would install smart plugs between the appliance and the electrical outlet. The smart plugs allow for the control and monitor of the energy flow towards the appliance. The company that supplied the plugs, Virtual Power Solutions, also provided us with the API that allow for the remote control of the plugs.

The implemented pilot worked structured upon 2 major components: the Middleware and the FrontEnd. The middleware was responsible to connect the Flexoffers services and the VPS services. Capable of persisting any operation that the user made, it allowed for managing of the House. To facilitate the user's work, each smart plug is connected to a room and each room is linked to the user's home/building. The middleware provides services for various features including application of flexoffers, retrieval of schedules, control of measures gathered from the smart plugs. The mechanism that solidly integrates the flexoffer concept in the house is the automatic emission of flexoffers and actuation on the devices depending on the schedule that is received. When a schedule is received, using that mechanism, the middleware will operate (e.g. turn or turn off) each device, in order to enforce the schedule. The FrontEnd was developed with the goal of providing the User with user friendly interface. Using the services that the middleware provides, the FrontEnd allows the user to execute all the task regarding the pilot. Using its user friendly graphical interface, the user can customize his house, from the automation of the usage of its appliances to the control and monitor of any individual smart plug. Built using a web server the FrontEnd application can be accessed on any platform with internet access.

The pilot was demonstrated in an official Arrowhead meeting in Turin, in October of 2016. The demonstration was successful, even getting some comments from the Project Leader, Dr Jerker Delsing. This project also lead to the publishing of a communication article and corresponding poster at the national Informatics symposium, INForum 2016, that took place in September. The paper was published and presented in the Distributed and Embed systems session, which had Dr Luis Pinho as lead chair.

5.2 ACCOMPLISHED OBJECTIVES

The pilot was successfully developed. The FlexHousing system, comprised of the Middleware and the FrontEnd integrates the Flexoffers and VPS Services while supplying a user friendly interface for the end-user The architecture relies on a middleware linking the Flexoffers services with the VPS Services. Table 70 represents the various goals that the pilot had and whether or not they were achieved.

Goal	Degree of implementation
Integrate the Flexoffer Service	Done
Integrate the VPS Services	Done with the exception of the registration of devices from scratch
Build Middleware with persistence	Done
Host middleware services	Done
Build FrontEnd for user interaction	Done
Allows user to apply Flexoffers	Done
Allows user the check the schedules for his devices	Done
Allow user to manage his house and devices	Done
Allow user to check the measurements gathered by the smart plugs	Done
Allow user to actuate remotely on devices	Done
Allow system to verify if the flexoffers were respected	Incomplete

Table 69 - Table describing the goals for the project

5.3 LIMITATIONS AND FUTURE WORK

The FrontEnd could still be improved. The usability isn't optimal for a commercial deployment and not customizable for different markets or companies. The aesthetics also need some innovation and creativity. The implementation of the Front in C# on a ISS webserver isn't the best approach, as the execution can put a heavy strain on any resource constrained and low cost hardware. The communication between the FrontEnd and the Middleware isn't encrypted, thus revealing a vulnerability.

There was an issue with the VPS Services, as their services weren't highly documented, leading to struggles in its integration with the middleware and support from VPS was not commonly available. As such, the registration of the device on VPS cloud, was left incomplete – the solution was to register the device manually, using VPS interfaces to the cloud. The service was available but the request was incomplete: some of the fields required were IDs that referred to objects that were local to the VPS Services, and thus unknown to the public.

One of the other issues was that every command or request that the pilot executed to operate a device had to be sent to the VPS Services. The Services would then relay the request through the internet towards the gateway in the house. Since the Gateway communicates with the smarts plug, a simpler implementation would be to communicate directly with the gateway, thus decreasing the latency of the system.

With the further development of the Flexoffers, the integration might suffer due to the implementation of new functionalities or the deletion of old ones. As such, until the Flexoffer Services are considered complete, the pilot must keep track of the change the external systems might suffer.

The future work might also revolve around the implementation of more features or even the integration of new systems, transforming FlexHousing into a system capable of controlling any smart device inside the house-hold the user. VPS also has system for Business monitor and control. Its integration could lead to the creation of a new platform, also capable of dealing with an industrial environment

The pilot could also benefit with the implementation of a manager application. The application would be responsible to manage various FlexHousing system that are registered in the Arrowhead Framework. The manager would be able to control the various instances of FlexHousing, interacting directly with it, thus minimizing the work of the end-user. Especially useful for condos or apartment buildings.

5.4 FINAL APPRECIATION

An internship at CISTER is one of best experiences one could ask for. The work environment is very flexible and any request for equipment or other demands met within a very short time.

The project allow for growth has it required the developers to deal with new technologies and new ways or programming. Any problem that was tackled was met with a solution. That solution was then discussed and revised with the project managers.

Considering the difficulties, the end result and time available, the outcome is pretty positive: 2 systems were developed, 1 communication paper was written and published in a conference, another paper is being written and the knowledge and experience of the developers was increased.

BIBLIOGRAPHY

[1] "IoT Expansion," [Online]. Available: http://www.cloudoye.com/blog/cloud-hosting/how-big-data-and-cloud-prowess-can-accelerate-iot-expansion. [Accessed August 2016].

[2] "ArrowHead FW Wiki," [Online]. Available: https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Technical_architecture. [Accessed August 2016].

[3] "Cister," [Online]. Available: http://www.cister.isep.ipp.pt/info/. [Accessed August 2016].

[4] L. L. e. a. Ferreira, "Arrowhead compliant virtual market of energy," in *Proc. of the 19th IEEE Intl. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2014.

[5] "OpenADR Alliance," [Online]. Available: http://www.openadr.org/. [Accessed 6 June 2016].

[6] "BIND9," [Online]. Available: https://www.isc.org/downloads/bind/. [Accessed 6 June 2016].

[7] M. P. D. W. R. a. S. Z. D. B. Terry, "The Berkeley Internet Name Domain Server," in *Proceedings of USENIX Summer Conference*, 1984.

[8] R. N. T. R. T. Fielding, "Principled Design of the Modern Web Architecture," in *ACM Transactions on Internet Technology (TOIT)*, 2002.

[9] "XMPP," [Online]. Available: https://xmpp.org/about. [Accessed 15 June 2016].

[10] "ISO/IEC/IEEE P21451-1-4 Standard for a Smart Transducer Interface for Sensors, Actuators, and Devices based on the eXtensible Messaging and Presence Protocol (XMPP) for Networked Device Communication," [Online]. Available: http://wiki.xmpp.org/web/Tech_pages/IoT_Sensei. [Accessed April 2014].

[11] "Internet History," [Online]. Available: http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet. [Accessed August 2016].

[12] "IoT Growth," [Online]. Available: http://www.businessinsider.com/how-the-internet-of-things-market-will-grow-2014-10. [Accessed August 2016].

[13] "IBM-IOT," [Online]. Available: http://www.slideshare.net/ArrowECSMarketing/internet-of-things-and-ibm. [Accessed 18 August 2016].

[14] "Lighting Smart Grids," [Online]. Available: http://www.directcurrent.eu/en/projects/public-lighting-on-dc-smart-grid. [Accessed August 2016].

[15] "Last mile smart grid," [Online]. Available: http://www.cs.rug.nl/aiellom/publications/PaganiAielloIsgt13.pdf. [Accessed August 2016].

[16] "Last mile smart grid topology," [Online]. Available: http://www.cister.isep.ipp.pt/docs/convergence_of_smart_grid_ict_architectures_for_the_last_mile/1039/view.pdf. [Accessed August 2016].

[17] "Neur.io," [Online]. Available: http://neur.io/products/. [Accessed 29 August 2016].

[18] "Canary device," [Online]. Available: https://canary.is/how-it-works/. [Accessed 29 August 2016].

[19] "Ivee Sleek," [Online]. Available: http://helloivee.com/. [Accessed 29 August 2016].

[20] "IoT house-hold applications," [Online]. Available: http://www.iotphils.com/solutions/smart-home/. [Accessed 18 August 2016].

[21] C. Teixeira, ""Convergence to the European energy policy in European countries: Case studies and comparison"," in *J. Soc. Technol*, 2014.

[22] "VPS Cloogy," [Online]. Available: http://www.cloogy.com/en/about/. [Accessed 20 May 2016].

[23] "VPS Devices," [Online]. Available: http://www.vps.energy/#!solutions/x8d7y. [Accessed 20 May 2016].

[24] "RUP," [Online]. Available: http://www.agilemodeling.com/essays/agileModelingRUP.htm. [Accessed August 2016].

[25] "RUP," [Online]. Available: https://en.wikipedia.org/wiki/Rational_Unified_Process. [Accessed 23 August 2016].

[26] "Java," [Online]. Available: https://java.com/en/download/faq/whatis_java.xml. [Accessed 15 June 2016].

[27] H. schildt, Java the Complete Reference Ninth Edition book, oracle press, 2014.

[28] "C#," [Online]. Available: : https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx. . [Accessed June 2016].

[29] "MVC explanation," [Online]. Available: http://www.tomdalling.com/blog/software-design/model-view-controller-explained/. [Accessed 18 May 2016].

[30] L. L. Ferreira, "The Arrowhead Aproach for SOA Application Development and Documentation," Arrowhead, EU, 2014.

[31] "Highcharts," [Online]. Available: http://www.highcharts.com/. [Accessed August 2016].

[32] M. E. Khan, *Importance of Software Testing in Software Development Life.*

APPENDIXES