|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | |  |  | | --- | --- | | |  | | --- | | **ESCUELA POLITÉCNICA SUPERIOR DE MONDRAGON UNIBERTSITATEA**  *MONDRAGON UNIBERTSITATEKO GOI ESKOLA POLITEKNIKOA*  MONDRAGON UNIVERSITY FACULTY OF ENGINEERING | | |  |
|  | **Trabajo presentado para la obtención del título de**  *Titulua eskuratzeko lana*  Final degree project for taking the degree of |  |
|  | **GRADO EN INGENIERÍA EN INFORMÁTICA** *INFORMATIKAKO INGENIARITZA GRADUA* DEGREE IN COMPUTER ENGINEERING |  |
|  |  |  |
|  | |  | | --- | | **Título del Trabajo** *Lanaren izenburua* Project Topic | | RUNTIME VERIFICATION FOR SPATIO-TEMPORAL PROPERTIES OVER IOT NETWORKS | |  |
|  |  |  |
|  |  |  |
|  | |  |  | | --- | --- | | **Autor** *Egilea* Author   **Curso** *Ikasturtea* Year | OIHANA GARCIA ANACABE   2021/2022 | | |

|  |
| --- |
| **Título del Trabajo** *Lanaren izenburua* Project Topic |
| **RUNTIME VERIFICATION FOR SPATIO-TEMPORAL PROPERTIES OVER IOT NETWORKS** |

**Nombre y apellidos del autor**   
*Egilearen izen-abizenak*   
Author's name and surnames   
GARCIA ANACABE, OIHANA   
  
**Nombre y apellidos del/los director/es del trabajo**   
*Zuzendariaren/zuzendarien izen-abizenak*   
Project director's name and surnames   
EZIO BARTOCCI   
ILLARRAMENDI, MIREN   
  
**Lugar donde se realiza el trabajo**   
*Lana egin deneko lekua*   
Company where the project is being developed   
TU WIEN   
  
**Curso académico**   
*Ikasturtea*   
Academic year   
2021/2022

**El autor/la autora del Trabajo Fin de Grado, autoriza a la Escuela Politécnica Superior de Mondragon Unibertsitatea, con carácter gratuito y con fines exclusivamente de investigación y docencia, los derechos de reproducción y comunicación pública de este documento siempre que: se cite el autor/la autora original, el uso que se haga de la obra no sea comercial y no se cree una obra derivada a partir del original.**

*Gradu Bukaerako Lanaren egileak, baimena ematen dio Mondragon Unibertsitateko Goi Eskola Politeknikoari Gradu Bukaerako Lanari jendeaurrean zabalkundea emateko eta erreproduzitzeko; soilik ikerketan eta hezkuntzan erabiltzeko eta doakoa izateko baldintzarekin. Baimendutako erabilera honetan, egilea nor den azaldu beharko da beti, eragotzita egongo da erabilera komertziala baita lan originaletatik lan berriak eratortzea ere.*

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Laburpena

(Laburpen amaieran ipini dokumentuaren amaierarantz informazio gehiago dagoela euskaraz \*Erreferentzia bat sartu atal horretara)

Resumen

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Table of contents

[1. Introduction 1](#_Toc101457140)

[1.1. Problem definition 1](#_Toc101457141)

[1.2. Objectives 2](#_Toc101457142)

[1.3. Project phases 3](#_Toc101457143)

[1.3.1. Introduction to the project and department 3](#_Toc101457144)

[1.3.2. Product development 3](#_Toc101457145)

[1.3.3. Other tasks 3](#_Toc101457146)

[1.3.4. Gantt Chart 4](#_Toc101457147)

[1.4. Product requirements 4](#_Toc101457148)

[1.4.1. Software 4](#_Toc101457149)

[Moonlight 4](#_Toc101457150)

[GitHub Actions and SonarCloud 4](#_Toc101457151)

[Programming languages 5](#_Toc101457152)

[Gradle 5](#_Toc101457153)

[Zephyr 5](#_Toc101457154)

[Integrated Development Environment (IDE) 5](#_Toc101457155)

[1.4.2. Hardware 5](#_Toc101457156)

[Thingy52 5](#_Toc101457157)

[Segger J-Link 5](#_Toc101457158)

[ESP32 6](#_Toc101457159)

[2. State of the art 7](#_Toc101457160)

[2.1. Cyber Physical Systems 7](#_Toc101457161)

[2.2. Monitors 7](#_Toc101457162)

[2.3. Spatio-Temporal Logics 8](#_Toc101457163)

[2.4. Middleware 9](#_Toc101457164)

[2.5. Communication protocols 9](#_Toc101457165)

[2.5.1. MQTT 9](#_Toc101457166)

[2.5.2. Bluetooth Low Energy 10](#_Toc101457167)

[2.5.3. Wi-Fi 10](#_Toc101457168)

[3. Development 11](#_Toc101457169)

[3.1. Monitor 11](#_Toc101457170)

[3.1.1. Data transformation 11](#_Toc101457171)

[Recording the received data 11](#_Toc101457172)

[Joining the values 12](#_Toc101457173)

[3.1.2. Features of the monitor 12](#_Toc101457174)

[3.2. Middleware 14](#_Toc101457175)

[3.3. Services SOA 14](#_Toc101457176)

[3.4. Design pattern 14](#_Toc101457177)

[3.5. Buffer 14](#_Toc101457178)

[3.6. Robustness 14](#_Toc101457179)

[3.7. Kodea egiteko sistemak / Pattern 15](#_Toc101457180)

[3.8. Physical system 15](#_Toc101457181)

[3.8.1. Thingy 15](#_Toc101457182)

[3.8.2. ESP 01 15](#_Toc101457183)

[3.9. MQTT 15](#_Toc101457184)

[3.10. BLE 15](#_Toc101457185)

[3.11. WIFI 15](#_Toc101457186)

[4. Problems and solutions 15](#_Toc101457187)

[How to manage the time and the buffer 15](#_Toc101457188)

[Dealing with missing values / Imprecise Signals 15](#_Toc101457189)

[Coordinate the sensors 16](#_Toc101457190)

[Duplicated values from sensors -Maybe DELETE- 16](#_Toc101457191)

[Other minor problems 16](#_Toc101457192)

[5. Use cases 16](#_Toc101457193)

[5.1. Office use case 16](#_Toc101457194)

[5.2. Wiener linien use case (?¿?¿?¿?¿?¿?¿) 17](#_Toc101457195)

[6. Economic memory 17](#_Toc101457196)

[7. Conclusions and future lines 17](#_Toc101457197)

[7.1. Conclusions 17](#_Toc101457198)

[7.2. Future lines 17](#_Toc101457199)

[8. Personal evaluation of the experience(?) and the project 17](#_Toc101457200)

[9. Sarrera, ondorioak eta etorkizuneko ildoak 18](#_Toc101457201)

[9.1. Sarrera 18](#_Toc101457202)

[9.2. Ondorioak 18](#_Toc101457203)

[9.3. Etorkizuneko ildoak 18](#_Toc101457204)

[10. Appendix A STREL 19](#_Toc101457205)

[11. Appendix B MQTT 20](#_Toc101457206)

[12. Appendix C BLE - Beacon 21](#_Toc101457207)

[13. Appendix D Data bus 22](#_Toc101457208)

[14. Appendix E Gantt chart 23](#_Toc101457209)

[15. Bibliography 24](#_Toc101457210)

Table of figures

[Figure 1‑A Project outline 2](#_Toc101457211)

[Figure 1‑B Hardware 6](#_Toc101457212)

[Figure 3‑A Online Monitoring Model 11](#_Toc101457213)

[Figure 3‑B Tuple.java usage example 12](#_Toc101457214)

[Figure 3‑C Features of the monitor 13](#_Toc101457215)

Table of tables

[Table 2‑A Software monitors for RV 7](#_Toc101457262)

[Table 2‑B Number of specification coverage on 1000 requirements [13] 8](#_Toc101457263)

# Introduction

This chapter is the introduction to the Bachelor’s Degree Final Project “Runtime verification for spatio-temporal properties overt IoT networks”. In this section, the concepts involved in the project are defined. Additionally, the project definition, scope of the project, planification and the product specification and requirements are explained.

## Problem definition

IoT (**Internet of Things**) is the area of computer science that collects the challenges of connecting millions of smart devices and sensors and making them accessible via the internet. This field is growing fast. The forecast is that the number of connected devices by 2030 will be 25.44 billion worldwide [1]. These devices are already part of several fields (e.g., e-health services, smart cities, e-farm, and intelligent transportation systems (ITS)), being a big part of the digitalization of society to build a smart world.

Among the systems that can exploit an IoT infrastructure, a noteworthy category is **Cyber Physical Systems** (CPS), where physical systems are monitored and/or controlled by a computational core. They interact with physical processes through sensors and actuators. The increasing numbers of IoT devices and intelligent systems made CPS influence society. They can be found in different sectors such as self-driving cars, home equipment and medical devices [2] [3]. The following definition is the most famous one for the term “Cyber Physical Systems”:

“Cyber-Physical Systems are engineering, physical and biological systems whose operations are integrated, monitored, and/or controlled by a computational core. Components are networked at every scale. Computing is deeply embedded into every physical component, possibly even into materials. The computational core is an embedded system, usually demands real-time response, and is most often distributed. The behaviour of a cyber-physical system is a fully-integrated hybridisation of computational (logical) and physical action."

(Helen Gill, US National Science Foundation) [4]

Monitoring is an activity related to the wider category of **Runtime Verification** (RV), which purpose is to observe information from a system while it is operating and analyse the behaviour to detect if it satisfies or violates certain properties. Monitoring the status of a CPS at runtime can give precise information to ensure reliability, safety, robustness and security [5] [6].

This project focuses precisely on the challenges when doing monitoring on CPS over IoT and provides an implementation of a service to monitor data collected by sensors at runtime. It is closely related to some aspects of Helen Gill’s definition. The IoT devices are in the physical part where they are spatially distributed and networked. The data will be collected, both across space and time. One main task of the project is to connect the sensors with the monitor so they can share information (i.e., networking). Finally, this data will be sent to MoonLight to monitor everything in real-time.

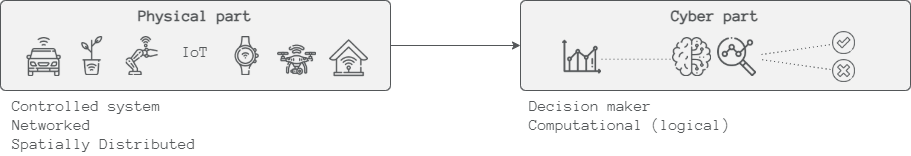


Figure 1‑A Project outline

## Objectives

The objective of this project is to feed MoonLight (the provided monitor) with live data. Moonlight previously has been tested with a Matlab interface since this tool has several CPS models and analysis tools available [7]. With that being said, this is the first time for the monitor to receive real data, so the purpose of this project is to help MoonLight be able to monitor different use cases and see how it works with actual real time data.

Therefore, the main objective is to implement a middleware. This middleware will enable the monitor to communicate with different applications in a distributed network. To achieve this, the middleware not only will provide some services but also will be capable expand by adding more services.

To implement this, several and diverse goals must be met. On the one hand, the monitoring of spatio-temporal properties will be carried out using logic-based specification languages, so the language used, STREL, must be comprehended in order to implement it on the use-cases defined in the project. On the other hand, the communication of the sensors with the monitor has to be established, to achieve this, two types of protocols will be used, MQTT and Bluetooth Low Energy.

The sensors are spatially distributed collecting the surrounding data. Using the Internet access, the communication between the physical world and the middleware must be enabled. With this connection, the middleware must be able to monitor the systems behaviour at runtime. Runtime monitoring analyzes the current state of the physical part to detect if it satisfies the specified properties.

Finally, the middleware has to adapt to the different use cases. The middleware is a software which offers and communicates different services. The services that must be present are the monitor, the data collector service and the dashboard. Furthermore, it should be easy to add more services and use cases, making it flexible and accept different upcoming data will add more value to the framework.

## Project phases

The project duration is of eight months, from November 2021 to June 2022. To achieve the objectives, the project has been divided into some tasks and scheduled to manage the work. The development has been divided into the next phases:

### Introduction to the project and department

The project is held in Technische Universität Wien (TU Wien), Austria's largest research and educational institution in the field of technology and natural sciences. The group is the Cyber-Physical System department. This phase has consisted of learning the subjects involved in the project and the tools employed during the development. The principal tool used was Moonlight, therefore, most of the time was dedicated to this, to learn the concepts to comprehend it (i.e., CPS and STREL) and how to use it.

### Product development

After studying the project bases and making some trial examples with the moonlight framework, the product development started. Starting from the middleware, continuing with the networking and the sensors, and finishing with the layout/framework, the development was carried out during the entire process.

A planification was scheduled to make sure that the objectives were achieved before the deadline. There was a beta release planned for the beginning of April. The beta release consisted in a product that was close in look, feel and function to the final version of the project. This beta release happened one week after the planned date, however, the product had more fancy stuff. The product not only had the main functions of the final product done (i.e., all the flow, beginning in the sensors until the moonlight monitor) but also it was able to fully support SOA (Service Oriented Architecture). The services implemented in the Middleware were the one in charge with the MQTT connections and the Online Moonlight service, able to monitor the input data. Besides, the Data Bus did …

Luckily, the middleware I was developing was as general as possible and the tests done had a good coverage, so to change the architecture of the software hadn’t take so long and I was able to extend the project.

In the resting month HTPP, … wre implemented in the middleware. In terms of hardware, the Thingy52 and the ESP were able to do … (Previously, I had some demo programs with the Thingy)

Some objectives of the project had changed or added from the beginning to the end. During the development of the project, the supervisors proposed new ideas, like the use of the dashboard. At the beginning the objective was to develop the way from the sensors to the monitor, but finally, adding another step, the middleware became bi-directional. It was not only able to take the data and monitor it, but also to get the results and communicate them to the client through the dashboard. To add this and more services to the middleware easily, it has a SOA architecture (Service Oriented Architecture).

### Other tasks

* **Tests**: Software testing has a high importance in the process of developing and evaluating a product. It has several benefits like precenting bugs and reducing development costs. Tests were used for several purposes, to validate that the units perform as expected, to verify the functions work well together and, in some cases, to do a TDD (Test Driven Development) where the test cases were developed before the software. Lastly, with the continuous integration, the tests where automated every time the commits were pushed to the repository.
* **Documentation**: During the entire project the details of the product were recorded in this document.
* **Meetings and more**: Every week I had a meeting with the supervisors to keep a track of the project, make some questions and do or ask for suggestions. Every now and then I had other extra meetings if needed. Besides, in the first months, I was given the opportunity to attend the IoT master’s course, that helped me with the sensors and the MQTT protocol.

### Gantt Chart

To organize the development, a Gantt chart was created at the beginning of the project, \_\_ is the summarized version of the planification. Some tasks took longer than expected due to the changes in the product and other unexpected issues, nevertheless, the project was successfully completed. As time passed, the track of the project was being recorded, in the chart of \_\_\_ appears the detailed actual development of the whole project.

GANTT CHART TXIKITO

## Product requirements

This thesis consists largely in developing a software project, due to this, the main resources used are software, nevertheless, some hardware devices were used too. Down below there is a list of the requirements needed during this project divided into two groups, software and hardware:

### Software

#### Moonlight[[1]](#footnote-1)

This is the principal resource, a lightweight Java-tool monitor, it can monitor temporal, spatial and spatio-temporal properties of distributed complex systems, like Cyber-Physical Systems. It has two different monitoring approaches, offline and online. Finally, supports the specification of properties written with the Reach and Escape Logic (STREL).

#### GitHub Actions and SonarCloud

GitHub Actions is a continuous integration and continuous delivery platform that allows for automating the build, test and deployment pipeline. This project is in a GitHub repository, so is possible to run a workflow. Every time a push is done, the Middleware is built, tested and analysed in SonarCloud.

SonarCloud automatically analyzes branches, the code quality and code security. By knowing the bugs, code smells, security hotspots and coverage and the fast feedback it was possible to maintain the code cleaner.

#### Programming languages

Nearly all the project is developed in Java. As previously mentioned, Moonlight is implemented as a Java program and so is the developed Middleware. The version used is Java 17, the last Long Term Support release.

C is used to develop the program of the devices of this project. A language that can be considered one of the most widely used programming languages in IoT.

#### Gradle

Gradle is a build automation tool. It is open-source and it is focused on flexibility and performance. Its features include the management of the dependencies.

#### Zephyr

The Zephyr Project is a scalable open-source real-time operating system (RTOS) supporting multiple hardware architectures (more than 350 boards), between them Thingy52 can be found. The Zephyr projects are CMake-based. CMake is an open-source, cross-platform family of tools designed to build, test and package software.

#### Integrated Development Environment (IDE)

First, for the middleware development, IntelliJ IDEA was used. This tool provides code completion, static code analysis and refactoring. This makes the development workflow experience smoother.

Then, Visual Studio Code was for the Thingys. This IDE provides just the tools a developer needs for a quick code-build-debug.

Finally, Arduino IDE was also used. This makes it easy to write code and upload it to the board, in this case to the ESP32.

### Hardware

#### Thingy52

The Nordic Thingy52 board is an easy-to-use prototyping platform. It is designed for helping to build IoT demos, with power optimization and several sensors. It is based on the nRF52832 Bluetooth 5 system on chip (SoC). In this project, the Bluetooth Low Energy (BLE) and environmental sensors such as temperature, humidity and air quality (CO2 and TVOC) are used.

#### Segger J-Link

SEGGER J-Links are the most widely used debuggers on the market thanks to its many supported CPUs and compatibility with the popular environments. The J-Link EDU Mini is a reduced version to allow students and educational facilities to debug different devices. USB-powered, can communicate at high speed.

#### ESP32

ESP32 is low-cost, ultra-low power consumption system on a chip microcontroller. ESP32 can interface with other systems to provide Wi-Fi and Bluetooth functionality through its SPI/SDIO or I2C/UART interfaces. In this project case, uses Bluetooth to communicate with the Thingy sensors and the Wi-Fi to publish it in the MQTT broker.



Figure 1‑B Hardware

# State of the art

In the state of the art chapter, the technologies used in the project are analysed to study the current level of development by searching and reading the previous researches done.

## Cyber Physical Systems

CPS plays an important role in Industry 4.0. The concept of Fourth Industrial Revolution was introduced in 2010 by the German government. This is an interpretation of Klaus Schwab, the executive chairman of the Word Economic Forum (WEF): “Now a Fourth Industrial Revolution is building on the Third, the digital revolution that has been occurring since the middle of the last century. It is characterized by a fusion of technologies that is blurring the lines between the physical, digital, and biological spheres.”[8]. The mechanical systems by their own are not relevant anymore, that is why CPS is widely applied in various fields. By providing the physical objects with computing and communication capabilities, they can turn into intelligent objects and surpass the previous systems. CPS can learn from the physical world and in some cases even interact with the world, turning environments into Smart Environments [9] [10]. In conclusion, CPS are deeply influencing the society and reshaping the perception and interaction with the physical world [3].

## Monitors

CPSs are susceptible to faults so there is a need to detect the status of the systems. Monitoring is an activity to observe systems behaviour, and enabling the runtime verification (RV) techniques, monitors can track systems while they are operating.

As previously mentioned, the monitor used in this project is **Moonlight**. Moonlight is a software monitor solution, in the next table some software monitors are presented:

Table 2‑A Software monitors for RV

|  |  |  |
| --- | --- | --- |
| Monitor | Specification Languages | Programming Language |
| Moonlight | STREL (Spatio-Temporal Reach and Escape Logic) | Java |
| RV-Monitor | LTL (Lineal Temporal Logic) | Java |
| jSSTL | SSTL (Signal Spatio-Temporal Logic) | Java |
| Kieker | - | Java |

* Moonlight: This monitor is a java-tool for monitoring properties of distributed complex systems.
* RV-MONITOR: This monitor enables the runtime verification, in this case, monitors are integrated into the original system to check its behaviour during execution [11].
* jSSTL: Is a java-based tool to monitor trajectories coming from simulations or from measurements of real systems. But it is an offline monitoring, so it does not support RV [12].
* Kieker: This framework is able to monitor at runtime. It allows developers to replace or add framework components [11].

Moonlight and its interfaces are still under development; however, it is sufficiently qualified to apply it in real environments, and therefore in this project.

Comparing the monitors previously mentioned, Moonlight is the most suitable one for this project. In the first place, it can be used for distributed systems, which means that it can connect with various components. Afterwards, it supports online monitoring, and the state of the system can be observed at runtime. Finally, with the STREL specification language, it can monitor the system both across space and time.

In conclusion, Moonlight has the desired features to monitor spatio-temporal properties of distributed systems. The other monitors have some interesting attributes, but they lack of some elements that are necessary for this project (e.g., jSSTL does not support online monitoring).

## Spatio-Temporal Logics

The monitors should include a language to specify the requirements needed to monitor the system efficiently. Signal Temporal Logic (STL) is widely used for analyzing programs in CPSs. However, STL is not expressive enough for real world cases, because they do not only require temporal information but also spatial needs. Most specification languages and tools available for CPS supports only the monitoring of temporal properties (e.g., MTL, STL and TRE). Thereby, there are some existing spatial extensions of STL (e.g., SSTL, SpaTeL and STREL) [13].

In the following table there is a comparison of the specification coverage on 1000 quantitatively-specified real city requirements between STL, SSTL, STREL and SaSTL:

Table 2‑B Number of specification coverage on 1000 requirements [13]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | SaSTL | STREL | SSTL | STL |
| Number of covered requirements | 950 | 431 | 431 | 184 |
| Covered key elements | Temporal & Spatial Range & Aggregation, Counting, Percentage | Temporal & Spatial Range | Temporal & Spatial Range | Temporal |

As we can see in the Table 2‑B Number of specification coverage on 1000 requirements [13]the specification language with the most coverage is SaSTL, nevertheless, STREL, the language used with Moonlight, is expressive enough to cover the most important requirements.

In spite of that, STREL is more convenient for the following reasons:

* STREL can handle online monitoring, this solution is preferable because it permits to take immediate action during execution and is computationally cheaper [14].
* STREL can handle mobile or dynamic CPS. In SSTL the topology is assumed to be static, it is impossible to monitor nodes changing locations [15].

In Appendix A **STREL** is more information about STREL and how to express the spatio-temporal behaviours in this specification language.

## Middleware

A distributed system is composed of several hardware and software elements which must be integrated, the element that facilitates the management of such environments is the middleware. For example, IoT requires integrating and working with data and different algorithms distributed in other applications, as well as operating in real time with a wide variety of processes.

Distributed systems are not only applicable to computers and physical components, but also a logical dimension. A distributed system can be defined as a set of independent processes or applications that interact with each other. The middleware is in charge of enabling the communication and data management of the distributed applications [16].

The middleware developed for this project communicates different services with each other in real time; these services can be found locally or distributed.

## Communication protocols

A communication protocol is required to exchange messages between computing systems. These are the communication standards used in the project:

### MQTT

MQTT (Message Queue Telemetry Transport) is a standard messaging protocol widely used for the Internet of Things. It is ideal for connecting remote devices with low computational power and a minimal network bandwidth. It is designed as an extremely lightweight publish/subscribe messaging transport and thanks to this, it has become one of the most used IoT protocols. Today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications… [17] (More information in Appendix B **MQTT**).

### Bluetooth Low Energy

Bluetooth Low Energy (BLE) is one of the most widely applicable low-power connectivity standards. It can be used in different situations, such as in the use of wireless headphones, hands-free calling and file transfer. This wireless personal area network technology was integrated into Bluetooth from version 4.0 in 2009 and is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range [18].

### Wi-Fi

Wi-Fi is one of the most widely used wireless technologies. It allows all kind of different devices to interface with the Internet. Internet connectivity occurs through a wireless router, when the Wi-Fi compatible devices connect to exchange information with one another, creating a network [19].

# Development

This chapter is dedicated to the development of the project. Here the processes to achieve the project are explained.

## Monitor

As previous mentioned, the monitor used is Moonlight. This project is not focused in developing this component, but it must incorporate it, so numerous activities were closely related to Moonlight.

To begin, Moonlight is a monitoring algorithm for STREL. The spatio-temporal logic STREL operates over a weighted graph representing the spatial arrangement of spatially distributed entities (i.e., maps each node and time instant into a vector of values, describing the internal state of each location). Hence, the monitor must be provided with the system’s spatial model and size. When setting up the monitor the desired formulas have to be specified too, to decide the satisfaction/violation of the observed events [7].

The monitoring typed used in this project is the online one. The offline method monitors the stored traces generated during the execution time. The online monitoring approach is performed incrementally, when a new piece of data is available the monitor can handle and observe it, giving the possibility to take immediate action.

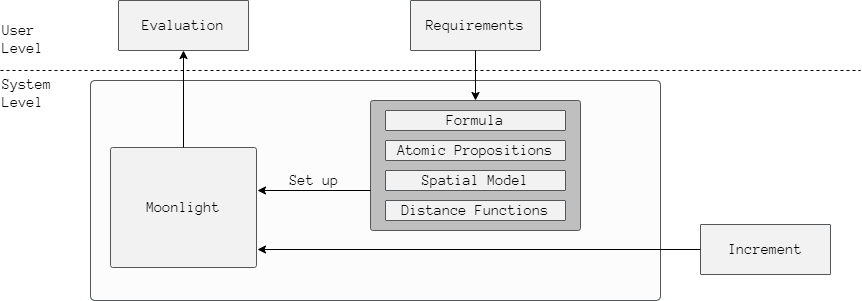


Figure 3‑A Online Monitoring Model

### Data conversion

The data that arrives from the sensors is not adequate to monitor before transforming it. Due to this reason the new information of the system goes throw two types of changes.

#### Recording the received data

The first change receives each data of each sensor. The developed software is expecting a JSON file whose values are extracted to create a *Tuple.java* and save important variables like the time and the sensor ID. It is worth mentioning that this class is only declared just in the necessary classes, whenever possible generic types are used to make future changes easier (i.e., if Tuple or other classes need to be replaced, there will be few changes). *Tuple.java* is a moonlight class to record immutable objects. It is suitable for this project because the number and the type of elements specified inside can be modified depending on the needs of the use cases, making it versatile. Figure 3‑B is an example of how to use a *Tuple.java*.

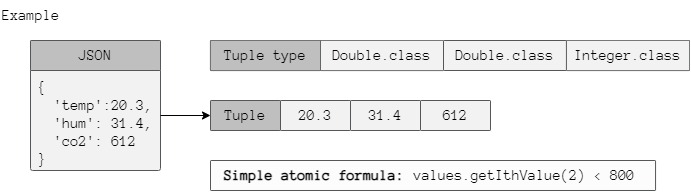


Figure 3‑B Tuple.java usage example

To conclude, the received sensors messages are divided in three parts, **ID**, **time** and **values** (in this case, Tuple). With this separation, a *Message.java* is created and the information is ready for the next step.

#### Joining the values

The purpose of Moonlight is to monitor the increments. The online monitor can monitor two types of data: *Update.java* and *TimeChain.java*. Both java classes are composed of a list of values (i.e., Moonlight needs all the nodes of the spatial model to monitor the increment). Consequently, there is a need to join all the sensors values together. To do this, the middleware needs a **buffer** (\_) to save temporally the sensors values and a converter to **create increments**. The process of joining the sensors messages has some complications that are explained in apartado. Finally, after joining the values the monitor is executed and the evaluation is achived.

### Features of the monitor

Moonlight combined with the middleware is capable of monitoring at runtime. In this section some elements that define the monitor are explained [20].

1. **Monitored objects – Distributed**: The software entities used during the project such as programs and services are distributed. The program receives different stream of events during execution and the process consists of communicating with different services such as sensors and a dashboard.
2. **Monitoring Access Methods**: This feature is presented in the following two aspects.
   1. **Monitoring Code Instrument Methods – Automatic (Static)**: For the initialization of the monitor, the formulas and the requirements are inserted using an external service (…?), without any intervention of programmers. Once the requirements for the monitor are set up, the online monitor service is initialized, and the values cannot be inserted or changed again.
   2. **Response Mechanisms – None**: The software developed during this project does not contain any technique to change the behaviour of the monitored system. Nevertheless, the results of the analysis are displayed in the dashboard to the user.
3. **Monitoring Mechanism Implementation – Logic**: Moonlight employs the human understandable logic STREL as a method to monitor. This framework can also handle mobile/dynamic CPS.
4. **Execution Relationships**: The execution relationship feature consists of two parts.
   1. **Monitoring Execution Models – Multi-Process Model**: The monitor is a separate service from the system being monitored. The behaviour and state of the monitored objects are communicated to the monitor externally. Moonlight works in a single thread for now, the parallelization of the monitor is expected to be added in the near future [7].
   2. **Interaction Methods – Middleware**: The interaction method defines the communication ways between the Moonlight monitor and the monitored system. In this case the middleware is used for the interaction. The monitor is not related to the monitored objects and there is no need to know where the observed system is. The Middleware is also used to communicate with different applications, so it can provide a distributed monitoring facility.

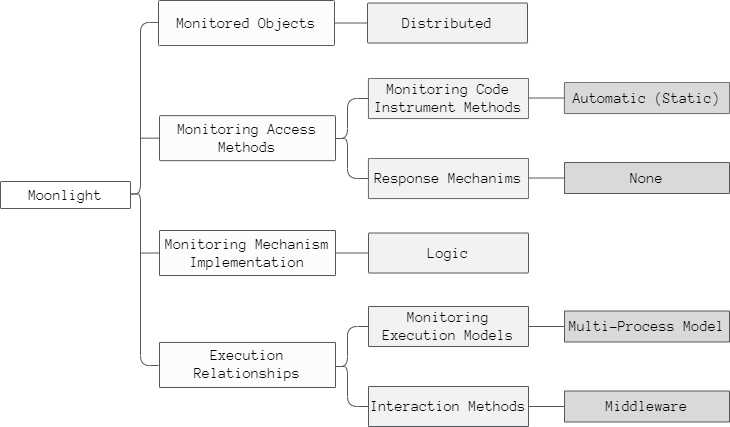


Figure 3‑C Features of the monitor

## Middleware

The middleware is the main developed component in this project. Middleware is the software that connects different components or applications with each other. The middleware developed for this project supports distributed networks and provides services like messaging. This is the architecture of the middleware:

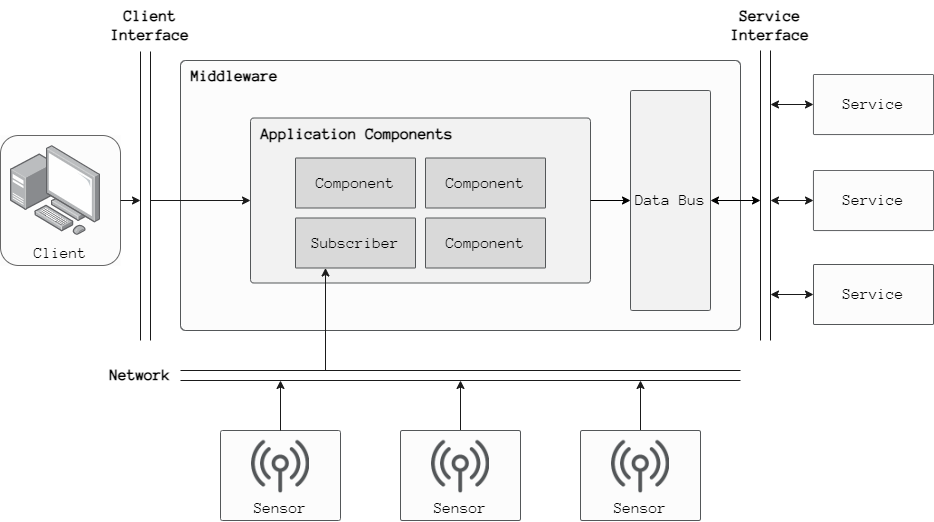


Figure 3‑D Middleware Architecture

Middleware technologies are increasingly important due to the growth of use of network-based applications. It acts like the connective tissue between applications, data, and users [21].

There is a broad category of different kinds of middleware depending on the main functionalities. On the one hand, the developed middleware works as an integration tool, connects external and internal applications with each other, sharing and streaming data among the components and the services in an asynchronous way. On the other hand, the middleware can support runtime for the use cases of this project, working agilely across services and the internal components, making the middleware behave as a new application by its own.

The middleware integrates the custom components as well as own and purchased services (Software as a Service, SaaS). Data management, application services and messaging are all handled by the middleware [22]. This are the services provided:

1. **Sensor service (\_)**: The objective of this service is to facilitate the communication between the sensors and the monitor.
2. **Monitor service (\_)**: This is the largest service of the middleware. Includes methods that save messages, convert the data and monitors.
3. **Result service (\_)**: This service is in charge of getting the results of the monitor and communicating them to the user.

## Service Oriented Architecture (SOA)

SOA is an important function for the software evolution, development and integration. Using the service interfaces makes the software components reusable and interoperable. The services use common pattern, this makes new services to be rapidly incorporated. Moreover, the internal functionality of a service doesn’t affect other services, they are independent. This means that the service interfaces provide loose coupling (i.e., services do not need to know how the services are implemented underneath). The service interface works between the services, some services provide some data and others consume.

In this way, SOA represents an important stage in the evolution of application development and integration over the last few decades. Before SOA emerged, connecting an application to data or other functionality that was in another system required complex point-to-point integration for each new developer project. SOA makes the integration of applications be easier, thereby the scalability and the maintenance of the application are improved [23].

The next diagram represents the architecture used to build this middleware, the services, data bus and other components are explained bellow.

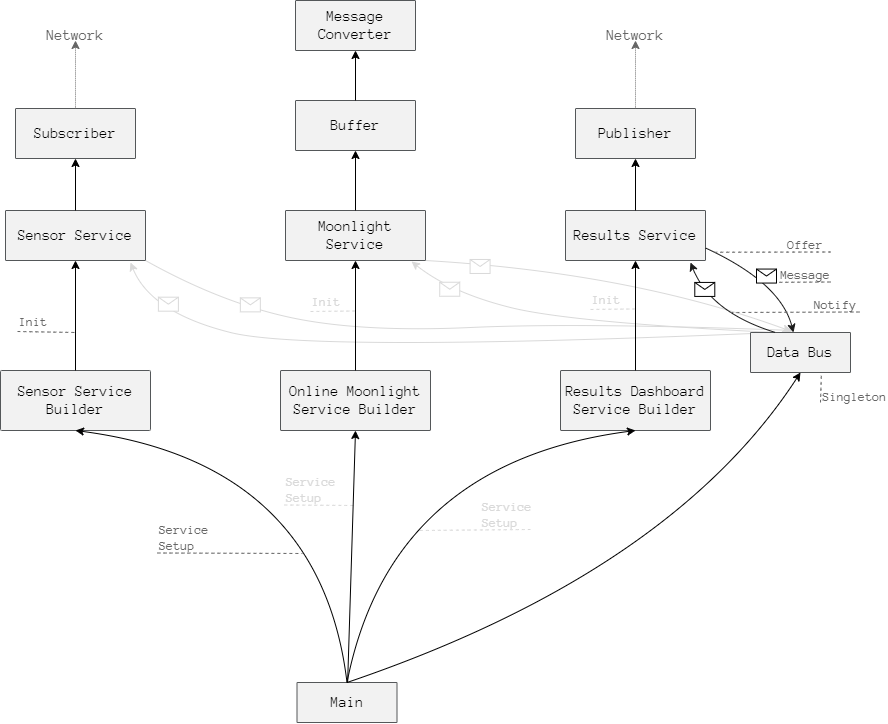


Figure 3‑E SOA Diagram

### Sensor service

This service facilitates the communication between the sensors and the monitor service. The sensor service contains a subscriber, this component listens to the MQTT broker and receives the data to monitor. This information is transformed to a *Message.java*.

The service is as general as possible, so the instructions are passed previously from the *Sensor Service Builder*. This service needs the broker and the topic to subscribe for the communication. For the communication with the MQTT broker, Eclipse Paho is implemented, an open-source library for lightwaight publishing/subscribing messages.

Then the service needs the necessary information for the transformation of the data to a Message.java, as explained above in Recording the received data. This is achieved in very few steps thanks to Gson. Gson is a Java open-source library that can be used to convert Java Objects into their JSON representation and vice versa. It does not need any Java annotations in the classes and provides simple *toJson()* and *fromJson()* methods for the conversions, which makes a very useful tool [24]. From the builder it just receives one class that implements the interface to set up common sensors message; for the use case (\_) looks like this:

1. public class OfficeSensorMessage implements Message, CommonSensorsMessage<Tuple>{
2. private int id;
3. private double time;
4. private double temp;
5. private double hum;
6. private int co2;
7. private int tvoc;

10. @Override
11. public int getId() { return id; }
13. @Override
14. public double getTime() { return time; }
16. @Override
17. public Tuple getValue() {
18. TupleType tupleType = TupleType.of
19. (Double.class, Double.class, Integer.class, Integer.class);
21. return Tuple.of(tupleType, temp, hum, co2, tvoc);
22. }
23. }

Table 3‑A Example of a sensor message class

So for this use case, with just this *OfficeSensorMessage.class* the conversion from JSON to the class is done in a single line. jsonMessage being a string and messageClass the desired java class (e.g., *OfficeSensorMessage.class*).

1. Message message = new Gson().fromJson(jsonMessage, (Type) messageClass);

Table 3‑B JSON to Message conversion

After transforming the data, the last step for this service is to offer this message to the data bus.

### Service 2

Moonlight is placed here and apart of this monitor, all the other components are created from scratch. Buffer, converter, timechains, monitor, sent to the database.

### Service 3

### Service Builders

### Data Bus

## Design pattern

Builder 🡪 <https://refactoring.guru/>

## Buffer

Collecting binary data bits into groups that can then be operated on as a unit,

automatic buffering.

It helps devices to manipulate data before sending or receiving.

## Robustness

Error handling

Maintainability

## Kodea egiteko sistemak / Pattern

Again “Visitor Pattern Considered Pointless - Use Pattern Switches Instead“-I buruz hitz egin eta Java 17. Again Duplikazioen eta polymorfismoari buruz hitz egin

## Physical system

### Thingy

Kconfig Json importatu ahal izateko 🡪 zephyrrena

CMakeList

Prj.conf 🡪 sensoreak enable egin ahal izateko

<https://github.com/google/eddystone/blob/master/protocol-specification.md>

### ESP 01

## MQTT

DR1 Lightweight communication methods

DR2 Interoperability.

DR3 Non-blocking event propagation. Events may arrive at unknown rates

DR4 Scalability(??)

Edge-based Runtime Verification for the Internet of Things

QoS -> WHY

## BLE

## WIFI

# Problems and solutions

#### How to manage the time and the buffer

Overriding problems, how to save them

#### Dealing with missing values / Imprecise Signals

The CPS must be able to overcome the system uncertainty, scalable and tolerant to threats

In this case, the uncertainty related to the absence of information must be taken into account

1. The very first values

Discard it directly

1. The missing values during the program

Time Chain splitter

#### Coordinate the sensors

Moonlight is prepared to monitor starting from the time 0

Time table to coordinate the sensors

Global reference time is one of the important architecture criteria in CPS components. The global reference time helps to ensure the real-time communication performance are achived. Global reference time is the basis CPS component to confirm the communication between physical and cyber world will work properly. CPS able to conduct a synchronous or asynchronous interaction with the physical world. Cyber-Physical System (CPS) State of the Art

#### Duplicated values from sensors -Maybe DELETE-

#### Other minor problems

* MoonlightRecord had some problems: Escape + online monitor + MoonlightRecord = infinite loop

The null values didn’t throw errors, wrong error handling.

I reported this issues -> Ennio created Tuple.class, that had everything MoonlightRecord was offering but with these errors fixed: I just used Tuple from that moment on.

* I was having problems with Windows + Zephyr project -> I used Linux for the coding of the sensors

for the Thingy sensors, I used a Linux virtual machine. IntelliJ IDEA does not support C/C++ officially and it requires quite a lot memory, that’s why I used Visual Studio Code.

# Use cases

## Office use case

## Wiener linien use case (?¿?¿?¿?¿?¿?¿)

# Economic memory

The majority of the cost of a software project is in long-term maintenance. [clean code liburua]

Intellij IDEA: unibertsitateko kontuari esker dohainik

# Conclusions and future lines

This is thechnical

## Conclusions

a. Reflexiones técnicas: relacionadas con los objetivos del proyecto

Data Bus pattern and its associated centralized team of integration specialists can become a bottleneck. (IBM SOA)

b. Reflexión sobre las implicaciones sociales, de salud y seguridad, medioambientales, económicas e industriales   c. Reflexión sobre la aplicación de conocimientos relativos a cuestiones económicas, organizativos de gestión (gestión del riesgo y del cambio) en el contexto industrial y comercial.

## Future lines

“Smart Home Automation System Using on IoT” dokumentuan rosas dagoenari begirada bat bota /!\

Legal aspects: General Data Protection Regulation (GDPR):

Error processing: Maybe there are things that are assumed that can fail and they are not handled, for example, the times of the sensors are always ascendant.

# Personal evaluation of the experience(?) and the project

Proiektua egiten nola sentitu naizen aipatu

Esperientziari dagokionez: A) Unibertsitatea: nola sentitu naizen, IoTko kurtsoak, astero egiten diren hitzaldietara joaten utzi, liburua utzi irakurteko… B) Beste herrialde batera joan: Leku berriak ezagutu, bertoko kulturatik ikasi, bakarrik bizitzea eta independentzia.

# Sarrera, ondorioak eta etorkizuneko ildoak

Atal honetan sarrera, ondorioak eta etorkizuneko ildoak atalen laburpen bat egingo da euskaraz.

## Sarrera

## Ondorioak

## Etorkizuneko ildoak

# Appendix A **STREL**

Titulua aldatzerako orduan kontuz! Formatua galdu gabe/!\ Aurkibidean polit ikusteko

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Tabla A1 …

# Appendix B **MQTT**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# Appendix C **BLE - Beacon**

BLE was created with IoT applications in mind, which has particular implications for its design. For example, IoT devices tend to be constrained and require extended battery use, so BLE favors low power consumption over continuous data transfer. In other words: when not in use, it goes into sleep mode to conserve energy. (<https://www.avsystem.com/blog/bluetooth-low-energy-ble/>)

# Appendix D **Data bus**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# Appendix E **Gantt chart**

# Bibliography

|  |  |
| --- | --- |
| [1] | T. Insights, «Statista,» December 2020. [En línea]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/. |
| [2] | L. Nenzi, E. Bartocci, L. Bortolussi, M. Loreti y E. Visconti, «Monitoring Spatio-Temporal Properties (Invited Tutorial),» de *Runtime Verification*, Springer International Publishing, 2020. |
| [3] | D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique y E. Bartocci, «A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems,» *IEEE Access,* vol. 7, pp. 13260-13283, 2019. |
| [4] | H. Gill, US National Science Foundation, 2006. |
| [5] | C. Tsigkanos, M. M. Bersani, P. A. Frangoudis y S. Dustdar, «Edge-based Runtime Verification for the Internet of Things,» *IEEE Transactions on Services Computing,* 2021. |
| [6] | M. Illarramendi, L. Etxeberria, X. Elkorobarrutia, J. Perez, F. Larrinaga y G. Sagardui, «MDE based IoT Service to enhance the safety of controllers at runtime,» 2019. |
| [7] | E. Bartocci, L. Bortolussi, M. Loreti, L. Nenzi y S. Silvetti, «MoonLight: A Lightweight Tool for Monitoring Spatio-Temporal Properties,» *ArXiv,* vol. abs/2104.14333, 2020. |
| [8] | K. Schwab, «The Fourth Industrial Revolution: what it means, how to respond,» World Economic Forum, 2016. [En línea]. Available: https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/. |
| [9] | J. Jamaludin y J. M. Rohani, «Cyber-Physical System (CPS): State of the Art,» *2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube),* 2018. |
| [10] | G. Garatu, «Que son los Sistemas Ciber-Físicos (CPS) en la nueva Industria 4.0,» 2017. [En línea]. Available: https://grupogaratu.com/que-son-sistemas-ciber-fisicos-cps/. |
| [11] | M. Illarramendi, «Runtime Observable and Adaptable UML State Machine-Based Software Components Generation and Verification: Models@Run.Time Approach,» 2019. |
| [12] | L. Nenzi, L. Bortolussi y M. Loreti, «jSSTL - A Tool to Monitor Spatio-Temporal Properties,» 2017. |
| [13] | M. Ma, E. Bartocci, E. Lifland, J. Stankovic y L. Feng, «SaSTL: Spatial Aggregation Signal Temporal Logic for Runtime Monitoring in Smart Cities,» 2020. |
| [14] | E. Visconti, E. Bartocci, M. Loreti y L. Nenzi, «Online Monitoring of Spatio-Temporal Properties for Imprecise Signals,» 2021. |
| [15] | E. Bartocci, L. Bortolussi, M. Loreti y L. Nenzi, «Monitoring mobile and spatially distributed cyber-physical systems,» 2017. |
| [16] | A. Rivas, A. Pérez García Osvaldo y J. Hernández-Palancar, «Estado del arte sobre aplicaciones del middleware ROS,» 2016. |

1. [GitHub - MoonLightSuite/MoonLight: a light-weight framework for runtime monitoring.](https://github.com/MoonLightSuite/MoonLight) [↑](#footnote-ref-1)