UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DE COMPUTAÇÃO

LAURA BRAGANTE CORSSAC

A Digital Twin-based Smart Home: A Proof-of-Concept Study

Porto Alegre
2021

LAURA BRAGANTE CORSSAC

A Digital Twin-based Smart Home: A Proof-of-Concept Study

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Advisor: Prof. Dr. Juliano Araújo Wickboldt

Porto Alegre
2022

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Juliano Wickboldt, for all the mentorship during the elaboration of this thesis. Also, for introducing me to the concept of the Digital Twin, which gave birth to the article. I also thank the Institute for parallel and distributed systems (IPVS) of the University of Stuttgart and the university itself for giving me the opportunity to learn many concepts I used in this work during an amazing research visit in Germany. Special thanks to my supervisors there, Pascal Hirmer and Ana Cristina Franco da Silva, who assisted me the whole year with mentorship and support. I also extend this acknowledgment to all the professors part of my student journey, who gave me the knowledge and instructions required for a Bachelor's degree.

Moreover, I thank my university colleagues, who I also proudly call my friends, for their support in the ups and downs of my university life. They were my agendas, coaches, free private tutors, role models, and psychologists many times. This includes all members of the (original) Whatsapp group CiC da Ousadia, but specially, Aline Weber, Catarina Nogueira, Guilherme Haetinger, and Marlize Ramos.

Regarding my experiences outside the university, I kindly thank my instructors, leaders, colleagues, and friends of the Apple Developer Academy and do Sicredi. With them, I developed not just applications but also myself professionally and personally.

Thank you to the friendships that remained from school and the exchange. Grateful to have shared many moments and feelings with them. Thanks especially to those who are or were in the same situation writing thesis, Diana Farje, Eduarda Mello, Lúcio Franco and Sofia Wender. Happy for this collegiality, even if unofficial, and for experiencing together (or almost that) this special moment that is graduation.

To conclude, special thanks to my brother, mother, and father. They were my safe harbor during my whole life and are always gonna be. Thank you for all the love, protection, support, and education it required to get here.

Thank you for standing with me when my code didn't work and celebrating with me when it did.

AGRADECIMENTOS

Antes de tudo, gostaria de agradecer ao meu orientador, Juliano Wickboldt, por toda a mentoria durante a elaboração desta tese. Também, por me ter apresentado o conceito do Gêmeo Digital, que deu origem ao artigo. Agradeço também ao Instituto de Sistemas Paralelos e Distribuídos (IPVS) da Universidade de Stuttgart e à própria universidade por me terem dado a oportunidade de aprender muitos conceitos que utilizei neste trabalho durante um intercambio incrível na Alemanha. Agradecimentos especiais aos meus supervisores de lá, Pascal Hirmer e Ana Cristina Franco da Silva, que me acompanharam durante todo o ano. Também estendo este reconhecimento a todos os professores que participaram na minha jornada de estudante, que me forneceram os conhecimentos e instruções necessárias para a obtenção do diploma de Bacharelado.

Além disso, agradeço aos meus colegas universitários, a quem também chamo orgulhosamente meus amigos, pelo seu apoio nos altos e baixos da minha vida universitária. Foram muitas vezes as minhas agendas, coaches, professores particulares gratuitos, fontes de inspiração e psicólogos. Isto inclui todos os membros do grupo (original) Whatsapp CiC da Ousadia, mas especialmente, Aline Weber, Catarina Nogueira, Guilherme Haetinger, e Marlize Ramos.

Quanto às minhas experiências fora da academia, agradeço gentilmente aos meus instrutores, líderes, colegas e amigos da Apple Developer Academy e Sicredi. Com eles, desenvolvi não só aplicativos, mas também a mim própria pessoa e profissionalmente.

Obrigada às amizades que ficaram da escola e do intercâmbio. Grata por ter compartilhado muitos momentos e sentimentos com eles. Obrigada especialmente àqueles que estão ou estavam na mesma situação escrevendo tese, Diana Farje, Eduarda Mello, Lúcio Franco e Sofia Wender. Agradecida por este sentimento de coleguismo, mesmo que não oficial, e por podermos compartilhar juntos (ou quase isso) este momento especial que é a formatura.

Para concluir, agradecimentos especiais ao meu irmão, à minha mãe, e ao meu pai. Eles foram o meu porto seguro durante toda a minha vida e sempre o serão. Obrigada por todo o amor, protecção, apoio e educação necessários para chegar até aqui.

Obrigada a todos por estarem comigo nos momentos em que o meu código não compilou e por comemorarem comigo quando funcionou.

ABSTRACT

Digital Twins have gained much attention from academia and industry. They are virtual systems that can fully describe a physical one by constantly receiving data from its sensors. The digital twin also fetches data from other sources such as simulators to make analyses, predictions, and decisions sent back to its physical counterpart. Another trending topic is Domotics, or Smart Home Technologies, which brings intelligence and comfort to a house by automating some of its functions. Although the research on both themes is vast, there are few implementations of Smart Homes based on Digital Twin technologies, and this work aims to prove that residences can also benefit from this concept. We implement two different use cases showing that a two-way connection between a home and its virtual counterpart can provide its owners with analyses and simulation-based automation. The first study case allows the user to visualize their home appliances' past and present state concerning their energy consumption. Based on heating simulations, the second determines the best time to turn on a heater in order to increase the thermal comfort of the home and reduce energy usage.

Keywords: Digital Twin. Smart Home. Domotics. Internet of Things. MQTT. Kafka.

Uma casa inteligente baseada em um gêmeo digital: um estudo de prova de conceito

RESUMO

Os gêmeos digitais ganharam muita atenção do meio acadêmico e da indústria. São sistemas virtuais que podem descrever completamente um sistema físico através do recebimento constante de dados de seus sensores. Os gêmeos digitais também buscam dados de outras fontes, como simuladores para fazer análises, previsões e decisões enviadas de volta para sua contraparte física. Outro tópico de tendências é a Domótica, ou automação residencial, que traz inteligência e conforto a uma casa, automatizando algumas de suas funções. Embora a pesquisa sobre ambos os temas seja vasta, há poucas implementações de casas inteligentes, ou Smart Homes, baseadas em tecnologias de Gêmeos Digitais, e este trabalho visa provar que as residências também podem se beneficiar deste conceito. Implementamos dois casos diferentes de uso mostrando que uma conexão bidirecional entre uma casa e sua contraparte virtual pode fornecer a seus proprietários análises e automação baseada em simulação. O primeiro caso de estudo permite ao usuário visualizar o estado passado e presente de seus eletrodomésticos no que diz respeito ao seu consumo de energia. Com base em simulações de aquecimento, o segundo determina o melhor momento para ligar um aquecedor a fim de aumentar o conforto térmico da casa e reduzir o consumo de energia.

Palavras-chave: Smart Home. Domótica. Internet das Coisas. MQTT. Kafka.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BIM	Building Information Model/Modeling
DT	Digital Twin
DTH	Digital Twin Healthcare
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
RGB	Red Green Blue
UI	User Interface
UX	User Experience

CONTENTS

1 INTRODUCTION

A Digital Twin (DT) is a virtual replica of a physical system bidirectionally connected to it (GRIEVES, 2016). All data from the physical environment, collected mainly by sensors, is transferred to the digital one. The latter uses this data for playing its two core roles: the predictive and the interrogative. The first is about predicting the future state of the physical system, including but not limited to anomaly detection, fault prevention, or the prediction of a given property (e.g., temperature or pressure) of an object. The virtual replica may also make some decisions from the conclusion of its simulations and data analysis. It can send instructions back to the real world, such as firing an alarm in case of danger or turning off a device after noticing its faulty operation. In turn, the interrogative role is about displaying the current state of the physical system. Both systems have to be connected in real-time so that the user can remotely verify the values read by the sensors, and these must precisely reflect those of the physical system.

Another trending subject, which helps the DT in its real-time data collection, is the Internet of Things (IoT). It proposes that computers sense the physical world by adding sensors, electronics, devices, and internet connectivity to objects, i. e., things (NEGRI; FUMAGALLI; MACCHI, 2017). IoT, like the DT, can act in a variety of fields. One of them is Domotics, or Home Automation, which involves controlling and monitoring appliances of a home, called Smart Home, in a unified system (MIORI; RUSSO, 2014). IoT technologies can, for example, make homes more secure (KEAT; CHUAH, 2018) and more energetically efficient (MOTLAGH et al., 2018).

The Digital Twin is a concept that is increasingly gaining the attention of academia and industry. Its applications vary in their goals, domains, and size. For example, NASA states that the DT will be crucial to its future vehicles (GLAESSGEN; STARGEL, 2012). It will enable higher safety and reliability with high fidelity simulators of the behavior of a space object from before and during its flight. A twin of the Bobsleigh building used in China's Winter Olympics is also possible. It could detect the presence of burglars, the risk of fire, and overcrowding (LIU; ZHANG; WANG, 2020). However, there is not so many studies of a DT for a residential place to provide its inhabitants with more convenience on daily activities.

With this in mind, the goal of this work is to implement a proof of concept digital twin of a residence. It aims to make the house more pleasant to its owners by making it more intelligent, safer, and more economical in different ways. It is different from

traditional Smart Home technologies because its aims go beyond automation, including simulation, event prediction, and past plus present data analysis.

We evaluate our digital twin with two use cases. The first one focuses on the interrogative role of the DT and works as a data analyzer of the energy consumption of a residence. This scenario gives people more information about their habits, enabling them to rationalize their power usage. The second, in turn, focuses on the predictive role of the concept. It decides the best time to turn on a room's heater so that its temperature equals a target value at a determined time. It makes the decision consulting simulation data to predict the heating of the space and successfully provides better thermal comfort to the users.

In the next chapter, we detail the definition of the Digital Twin and explain related concepts, such as Internet of Things (IoT) and Smart Homes. We also explain some of the technologies used for our implementation, i. e., Apache Kafka, and the Message Queuing Telemetry Transport (MQTT) protocol. In Chapter 3, we propose an architecture of a residence's Digital Twin. Later, we describe our implementation and, in sequence, the use cases. We then finish this work exposing some conclusions reached during its elaboration.

2 BACKGROUND INFORMATION

This chapter provides the background information necessary for understanding the proposal and implementation of this project. We also present related work about the concepts we cover, i.e., IoT, Smart Homes, and Digital Twins.

2.1 Internet of Things

The internet of Things (IoT) is a trending concept of Industry 4.0, which is a synonym of the fourth and most recent industrial revolution. The IoT term proposes that objects, i. e., *things*, should be connected to the internet so that computers can sense the physical world autonomously. This is done by embedding sensors, electronics, devices, and internet connectivity to these elements. (FLEISCH, 2007) (ASHTON et al., 2009) (NEGRI; FUMAGALLI; MACCHI, 2017)

Things that previously worked independently from their context and didn't share data with anything else now can be life-savers. Although it originates from simple time-measurement devices, today's smartwatches play a major role in IoT. Wearable devices such as those can improve clinical diagnostic by providing physicians with real-time monitoring information of some vital aspects of their patients. This large amount of accurate data could previously not be collected through sporadic doctor's visits. In some e-health applications, algorithms of data mining, time-series pattern recognition, and machine learning also have an important role in generating insights about users' health. (FARAHANI; FIROUZI; CHAKRABARTY, 2020)

2.2 Smart Homes

Smart homes are homes equipped with technologies that make daily life more convenient. Smart Home Technology, also referred to as Domotics, can have different purposes, such as making the home more secure by using object recognition to detect the presence of burglars and making the home smarter with autonomous cooling and heating systems (BROMLEY; PERRY; WEBB, 2003).

Although the limitless prospects that smart homes bring, there is still resistance to adopting these technologies. The reasons include their price (GILL et al., 2009)

(HARPER, 2003), and that's why the Internet of Things plays a significant role in home automation by providing affordable and reliable sensors.

An example of application that combines the concepts of IoT and Smart Homes for making a home more secure is presented by Kaet et al. (KEAT; CHUAH, 2018). Their work is about a home monitoring system that, when any motion is detected, it captures an image, uploads it to a cloud storage, and finally sends a notification to the user by email and SMS. In this case, motion sensors and internet connectivity allowed an immediate and remote notification to the users when there is a risk of danger in their house.

Another use case is a lighting system that automatically controls a dimmable lamp considering the user's preferences and the current illumination of the environment. Motlagh et al. (MOTLAGH et al., 2018) could accomplish an energetic gain by installing light sensors external and internally to a room and asking the user to set the most pleasant light tone for them.

2.3 Digital Twin

This section defines the concept of the Digital Twin. Later, we describe its value and purpose; after that, its applications, exemplified with related work in the subject. In the end, we expose how to evaluate the system to give a notion of the ideal DT.

2.3.1 Concept

Who first introduced the concept of the DT was Grieves (GRIEVES, 2016), who defines a Digital Twin as a digital replica of a physical system that can completely describe its physical counterpart and communicates constantly and bi-directionally. The information from the sensors installed in the physical system transits to the virtual one in real-time. Moreover, the DT may request data from external API and receive manual inputs from its stakeholders, such as a Bill of Materials that lists current and past components and specifications of the pieces of equipment. The digital copy will use this aggregated information to produce simulations and analyses, predict failure, and more, depending on the application. The result of this computation may be sent back to the physical world, triggering actuators, or may help the users draw their conclusions.

2.3.2 Value and Purpose

The possibilities of the DT are numerous. Accordingly to Grieves (GRIEVES; VICKERS, 2017), the core premise for the Digital Twin is that information replaces wasted physical resources, i.e., time, energy, and material. Since virtual costs are sinking every year while the physical ones are rising, a DT can dramatically reduce the costs of creating, producing, and operating a system. It enables us to, for example, perform destructive tests inexpensively or test if two parts of the system fit together before constructing them, reducing the waste of materials.

Another possibility is to from-run a simulation in time. For example, it could generate an alert to an airplane's pilots to change some behaviors before they result in a fatal tragedy.

Grieves also lists two main purposes of the DT:

- **Predictive:** DTP could predict the behavior of the physical system before it even exists. A DTI, in turn, can produce data-driven analyses of one physical product or a combination of more to predict some aspects of the physical world, such as faults.
- **Interrogative:** This functionality applies only to DTI's and allows the users to inspect the current state of the physical system. The DTI's could show, for example, the current temperature measured by a sensor or the average of the past values.

The term is also considered promising by NASA (GLAESSGEN; STARGEL, 2012). As future vehicles will have new characteristics, they will require more reliable techniques of simulation and monitoring. Traditional heuristic-based safety measures may become ineffective for future complex systems, and extensive physical testing may become too expensive. In this context, the DT will revolutionize aspects, such as certification and fleet management, by providing ultra-high fidelity models that mirror a product's lifecycle. Once a space vehicle launches, its DT will signalize anomalies, produce continuously refined predictions, and help mission managers to simulate the impact of in-flight changes.

Regarding buildings, Khajavi et al. (KHAJAVI et al., 2019) list some of the considerable benefits of creating a digital twin in this sector:

- gathering, generating, and visualizing the environment of the building
- analyzing data irregularities
- optimizing building services

Another possibility of a DT is the optimization of repairs. Some defects on, for example, a building's elevator can be easily detected, and the mechanicals immediately notified, reducing the inconvenience to the residents. Real state agencies also can profit from the data from all their products to make better future decisions. [1]

Because of the growth of Virtual Reality and the introduction of Facebook's Metaverse[2], there was a significant enhancement of 3D modeling technologies. In this scenario, DT from buildings can also have advantages. Virtual visits to a facility could replace costly physical ones.

2.3.3 Applications

In this section, we list some applications of the DT found in the literature. Although there are some misconceptions of the definition of a DT, the projects use this technology for simulating changes, monitoring, making predictions, and detecting anomalies and faults. We divide them into two sections: the first covers works concerning buildings and houses, and the latter other contexts.

2.3.3.1 Buildings and Houses

Liu et al. (LIU; ZHANG; WANG, 2020) propose a DT for a building as an indoor safety management system (ISMS). The virtual replica constantly receives information from sensors installed on the building, such as cameras, thermometers, and magnet sensors placed on the doors and windows. Having this data, AI models can detect the risk of fire, overcrowding, or intruders. If it is the case, some alerts containing the level and type of risk and some counteractions suggestions appear on the digital 3D representation of the building, which is accessible via a website and also works as a means of monitoring. Moreover, the insights are sent to the physical world and fire an alarm. The authors tested the approach in a Bobsleigh building that will be part of the 2022 Winter Olympics in China and had positive results.

Khajavi et al. (KHAJAVI et al., 2019) placed a Wireless Sensor Network (WSN) in a building's facade to measure its light, temperature, and relative humidity. The DT

[1] What are digital twins in smart buildings? - <https://inbuildingtech.com/bms/digital-twin-commercial-office-building/> Visited on: 6 Fev. 2022.

[2] The Metaverse and How We'll Build It Together – Connect 2021 - <https://youtu.be/Uvufun6xer8?t=2695>. Visited on: 6 Fev. 2022.

receives the real-time data captured by the construct and produces a graphical visualization of the light values on different parts of the facade of the building. In this way, the DT contains a 3D model of the object studies and overlays it with a color-coded gradient based on the light values. This additional component is segmented in rectangles so that each of them represents the region covered by one of the nodes of the WSN. Each partition is colored based on the light values sensed by its respective node. The higher the frequency of the color shown in the virtual system, the higher is the light intensity measured by the respective sensor. The work also suggests the same technique for visualizing other sensors and further use of the WSN in applications involving smart curtains and air conditioning control.

Kaewunruen et al. (KAEWUNRUEN; RUNGSKUNROCH; WELSH, 2019) make a DT of an existing building in order to calculate the necessary means to transform it on a Net Zero Energy Building (NZEB). The authors designed two different Building Information Models (BIMs), one for the existing building characteristics three-dimensionally; the other is a copy with modifications on some elements, such as windows and walls, to improve the thermal efficiency. With the help of an assistant software, they calculated that the modifications resulted in an energy consumption decrease of 6,67%. After doing more simulations, they prove that it is possible to install some solar panels and wind turbines on the building to make it energetically self-sufficient and guarantee a 23 years payback. The term DT in their work is a synonym of a BIM model, which has a slightly different definition from what we consider since the latter structure does not work with real-time data (KHAJAVI et al., 2019). However, their proposal and results are relevant to our discussion.

2.3.3.2 Further Applications

Bairampalli et al. (BAIRAMPALLI et al., 2021) make a digital twin of his Italian coffee machine, or Moka, to detect anomalies in the coffee-making process. They place nine temperature sensors in different parts of the object studied and a pressure sensor. Afterward, they build a data set by preparing coffee under different conditions while collecting data from the sensors. Having the data-set, an AI algorithm for fault detection could reach some conclusions, such as the relationship between the temperature during the process and the amount of water used.

Another sector that has its attention to the DT is manufacturing. One of the main reasons this technology can provide the manufacturers with real-time monitoring of their

machinery and production line and, thus, the ability to predict issues sooner. Zheng et al. (ZHENG; YANG; CHENG, 2019) propose a DT of a welding production line that enables real-time virtual monitoring of the physical system. The virtual replica contains models of the geometrical, physical, and kinematic properties of the object. The first describes the shape and size of the machinery; the second, its mechanical and thermodynamic state; the last, the connections between its components. Both virtual and physical systems are synchronously connected so that a remote, real-time, visual monitoring of the physical production process is possible. The 3D representation of the system shows the current state of the equipment and alerts of defects detection.

The final field of application of the DT that we investigate is healthcare. Liu et al. (LIU et al., 2019) define the physical world composed of a patient, wearable devices, medical equipment, and environmental circumstances which can affect a person's health, such as weather and political decisions. The virtual space should receive in real-time all the healthcare data produced by the physical, such as real-time monitoring data collected by wearables, simulation data, historical medical records, blood pressure measurements, and blood test results. The authors state that DTH has two distinct roles:

- **Hospital management and design:** This system aggregates data from different patients' health status and weather factors to predict seasons of high demand for medical assistance. Having these simulation insights, it can notify hospital personnel and help in scheduling the appointments.
- **Patient healthcare:** This system is divided into real-time supervision and crisis early warning. The first receives real-time data from a patient, such as the heart rate, and enables supervision by healthcare givers. Moreover, the procedure will recommend dosage and frequency of medication and provide medication reminders. The crisis early warning system, in turn, will run models to predict dangerous events such as falling and, if it is the case, will alert caregivers and guide the person on what to do.

2.3.4 Evaluation

Grieves et al. (GRIEVES; VICKERS, 2017) also propose a method very inspired by the Turing Test for evaluating a DT. In the test, a human poses in front of two screens. One shows a video stream from an actual room where a physical system is present, and

the other the DT. The first part of the evaluation is called the Sensory Visual Test, in which the human can ask for spatial manipulations on both digital and physical worlds. If the observer cannot tell which screen shows which system, the DT is approved on this test. The second part is the Performance Test, in which the observer can ask to have any action performed on both systems, such as applying a force to an element. Again, if the observer cannot distinguish the behavior between the systems, the DT passes the Performance Test. The third and last test is the Reflectivity Test. Now the observer can ask to know the status of some components, e.g., they may ask the temperature read by a thermometer placed on a room. If the virtual model provides precise information about the physical object and the observer can't conclude which screen shows which system, the DT passes the test.

Although we do not evaluate our work in this project and none of our references do it, we think it is worth mentioning the tests to show what the ideal DT looks like and provide a better understanding of the Digital Twin concept.

2.4 Comparison between terms

In this section, we try to differentiate the concepts of IoT, DT, Smart Home, and BIM. Since there is often an overlap between them and there are different definitions in the literature, we describe below the ones adopted in this work.

Some confusions may arise by distinguishing DT's from general computing models and simulations (GRIFFOR et al., 2017). IBM states on its website[3] that the difference stays on the scale. While simulations usually apply for one single process, a DT studies the combination between various of them. Moreover, simulations commonly do not benefit from a bi-directional data integration with the studied system.

A DT from a building might also be confused with a Building Information Model (BIM), which is the resulting file from Building Information Modeling (also BIM). The latter is the process of keeping an accurate and interoperable record of building information to enhance planning, construction, and maintenance over the life cycle of a facility. The former, in turn, combines a 3D model of a building and additional information such as the materials of its components to produce some estimations related to cost, energy consumption, and maintenance management. With these insights, designers, architects, and engineers can prevent mistakes in the design and production phase. The first key difference is that the BIM model does not count on real-time and autonomous interac-

[3]What is a DT: <https://www.ibm.com/topics/what-is-a-digital-twin>. Visited on: 6 Fev. 2022.

tion with the physical system during its operation phase. Another difference is the users. While a BIM is for the responsible personnel for the construction of the building, a DT of a home can help its residents to monitor it and make it more intelligent and comfortable for them.

To conclude this section, we differentiate traditional Smart Home technologies from a home's Digital Twin. The first aims to bring house smartness mainly with automation. We see the latter also as a means of turning an ordinary home into a Smart Home; however, it has a wider range of purposes. A DT for a home can provide the users with, besides automation to their residences, means of simulation and analysis with a higher degree of fidelity.

2.5 Technologies

This section introduces two technologies that enabled the implementation of our use cases, MQTT and Kafka. Thus, it provides the necessary background information for understanding our development described in Cap. 4.

2.5.1 MQTT

The Message Queue Telemetry Transport (MQTT) is a connectivity protocol standardized by OASIS. The MQTT protocol is the most widely used for collecting the measurements of IoT sensors. Because of its push protocol aspect, and bandwidth efficiency, it is more suitable for this context that comprises low-power machines and a poor bandwidth network (SONI; MAKWANA, 2017). Thus, we use this technology in our use cases to collect data from different sensors installed in our physical object of study, i. e., a house.

Its architecture relies on a many-to-many publish-subscribe model that comprises two main entities: the clients and the broker. The clients can be publishers, subscribers, or both. The first category, which is mainly composed of sensors, firstly connects to the broker and then publishes messages to topics on the broker. Subscribers subscribe to one or more topics after being connected to the broker. The broker's role is to receive data on different topics and send the information to the respective subscribers of each topic. Fig. 2.1 illustrates this functioning representing topics as colors and the data path as the

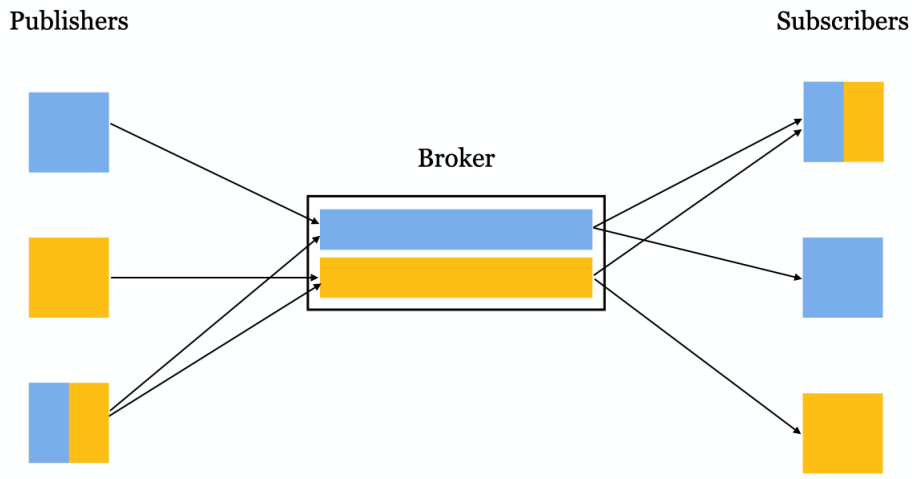


Figure 2.1 – Publish-subscribe model

arrows. While the publishers write to one or more topics, the subscribers read from many of these structures. Although the image doesn't show, it is possible to have clients that play both roles.

2.5.2 Apache Kafka

Kafka is an event streaming platform created and open-sourced by LinkedIn in 2011. The original developers later founded Confluent, the leading distributor and maintainer of the software. According to the latter company's website, the software is expressively gaining popularity among big corporations, including Netflix, Airbnb, Microsoft, and, of course, LinkedIn.

Kafka's core data structure is a topic, which is an ordered and infinite event sequence. Each of these events contains a timestamp and some data to describe the current state of a system. For example, we could have a topic for monitoring the location of a car. We then could every second send an event to the topic. Each of the messages could contain the latest latitude and the longitude of the vehicle. The messages' metadata field named ROWTIME is responsible for storing a timestamp, which can be set by the producer or by the broker. In the first case, ROWTIME represents the event-time, i. e., the current wall-clock time of the producer environment when the event happened. In the second, the field stores the ingestion-time, i. e., the current wall-clock time of the broker environment when the broker received the event. Having both moments in the messages

is also possible. The producer can add a field in each message's content, not metadata, containing time information. The broker, lacking a setting that states it should consider that field as the ROWTIME, sets the latter as it receives messages. That is how we implemented both of the use cases in this work. The scripts that build and publish the events add time information on their content, and the Kafka broker adds the ROWTIME to each of them. However, the latter data was unneeded for our applications.

All topics are stored durably and persistently, meaning that past data is accessible. Furthermore, they are fault-tolerant by being replicated in many disks, and there are also no restrictions on the size of the topics. It is possible to make small ones, to represent phenomena during a short period, or others that last years and handle trillions of events a day. Because of its trustworthy and efficient data storage and manipulation, we used this technology in our two use cases. It allowed us to keep different data sources in separate streams and query their information to execute the analysis.

We next describe the four main components of Kafka: Producer, Consumer, Connect, and Streams. Narkhede et al. provide more information in their book (NARKHEDE; SHAPIRA; PALINO, 2017).

Producers write data on Kafka topics. They are clients of Kafka's services and may also be known as publishers or writers.

Consumers read data from Kafka topics. They are also clients of Kafka's services and may be known as subscribers or readers.

Connect is a tool provided by Kafka that allows the integration of Kafka with external data sources. The Kafka Source Connector is a Producer that reads from an external service and writes on Kafka topics. The Kafka Sink Connector is a Consumer that reads from Kafka topics and moves data to an external system. For example, we could have a database that stores bank accounts information. We could configure a Source Connector to publish a message in a Kafka topic each time a new bank account is created or deleted. We then would have our database reflecting the current state of data and a Kafka topic to represent a historical log of the changes in this database. There are different connector types depending on the external system we want to integrate. There are connectors for many technologies, such as *MongoDB Atlas* and *PostgresSQL*, available for downloading on Confluent Hub.

Kafka Streams is a Java library developed by Apache Kafka for stream processing. Its most important data abstraction is the stream, which is a wrapper for a topic. The API allows us to execute operations on these structures, such as filtering, aggregating, and

joining. Following the previous example, we could use Kafka Streams to get the number of accounts created per minute during the last hour. As a result of our operation, we would have another topic with 60 events, one for each minute of the last hour and each containing the number of accounts created in that minute. Kafka's **ksqlDB** is another API built upon Kafka Streams that allows stream processing with an SQL-like query language. KsqlDB allows us to make three types of queries: Persistent, Push, and Pull. Them we describe next.

Persistent Queries are server-side queries made every time we create a new stream or table, which may be manipulations of others of these structures. For example, suppose we have a stream named "user_locations" that monitors users' locations. Each of its events contains a user's id plus their latitude and longitude on a moment. If we wanted a new stream that filters the previous to monitor a specific user, let's say one of id "user_1", we would write the following persistent query:

```
CREATE STREAM user_location_filtered AS \
SELECT * \
FROM users_location \
WHERE userid='user_1';
```

Push Queries are client-side queries that emit changes over time. We use one of these types to real-time monitor something since it does not end unless we manually close it. They characterize themselves by the presence of the statement "EMIT CHANGES" by their end. Following the previous example, if we wanted to receive *user_1*'s locations in real-time, we must write the following push query:

```
SELECT * FROM user_location_filtered EMIT CHANGES;
```

Pull Queries are also client-side queries, but, unlike Push Queries, they terminate and don't contain the "EMIT CHANGES" statement. They retrieve the current state of the system similarly to traditional RDBMS queries. If we wanted to count how many locations each user emitted *until now*, we would make a pull query. These queries require more steps to take place. First of all, we must configure Kafka to enable Pull Queries without WHERE clauses, which are not necessary in our case. Secondly, we must create a new structure named table, which is very similar to traditional RDBMS ones. It stores the current state from the system, containing a single record for each different value of its primary key. We create a table from a stream by using one aggregate function, such

as COUNT, AVG, and SUM, plus a GROUP BY <key> clause. The key in the latter construction determines the primary key of the table. As they reflect the current value of its primary keys, we can apply pull queries only on them, not on streams. However, we can also make push queries on tables, which work as a change-log. Returning to our example, we must take the following steps to count the locations of each user computed until now. Firstly, we set the configuration mentioned, and then, we create a persistent query to create a table. Finally, we make our pull query.

```
SET 'ksql.query.pull.table.scan.enabled' = 'true';

CREATE TABLE my_table AS \
SELECT userid, COUNT(*) \
FROM users_location \
GROUP BY userid;

SELECT * FROM my_table;
```

3 PROPOSAL

This section presents an architecture model of a residence's digital twin, which instantiates the conceptual model proposed by Behrendt et al. (DAMJANOVIC-BEHRENDT; BEHRENDT, 2019). We illustrate our model in Fig. 3.1 and detail its building blocks. More information regarding the abstract model can be found in the mentioned reference work.

Physical World: The physical world comprises real objects. It includes primarily our object of study, the house, and other factors that can interact with the system by changing its state.

Besides everyday items such as household appliances, furniture, and lights, the house encompasses sensors that perceive some of its characteristics later received by the virtual system. Moreover, it comprises actuators, which perform decisions from the digital counterpart. Among the sensors that we can have are, for example, infrared sensors to detect the presence of people, surveillance cameras, and thermometers at different locations in the house. An example of an actuator, in turn, is an alarm that fires every time the virtual system detects a dangerous situation.

Under the elements external to the house are people. Among others, residents and visitors interact with the object of study and change its state, for example, by turning on and off the appliances. The residents are the users of the whole system, whose goal is to reach data-based conclusions about the residence and somehow benefit them. External factors such as the weather can also change some aspects of the house, e. g., its indoor temperature.

Virtual World: The Virtual System's core component is the Digital Twin, which receives data from its physical counterpart and may also collect information from other sources, such as external APIs and simulators. Manual inputs containing files, for example, the Bill of Materials of the house, are also possible. The software stores all these data for different purposes, including analysis, prediction-making, anomaly detection, and simulations. It can then show the results of these computations to the users through different data visualization techniques or use them to make decisions and send them back to the physical house. Grieves (GRIEVES, 2016) describes that a DT should contain *"any information that could be obtained from inspecting a physical manufactured product."* We are more skeptical regarding this concept, and, for the scope of this work, a DT replicates only the information it needs for its goals. For example, a DT may not use the current in-

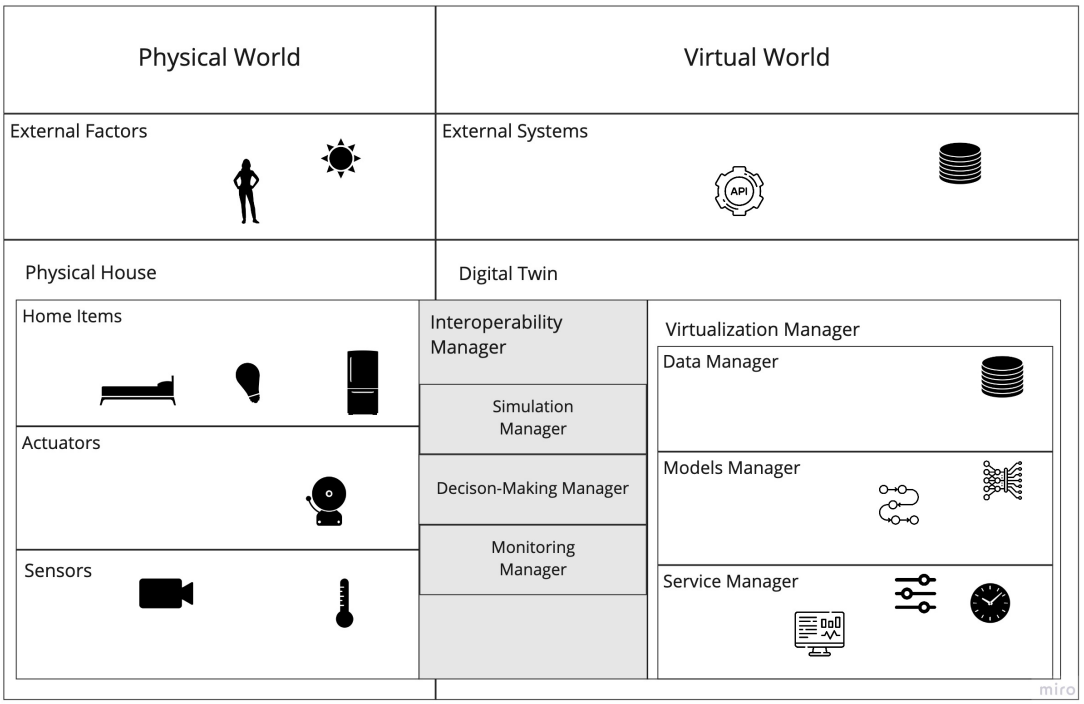


Figure 3.1 – A Home's DT architecture

terior position of a house's appliances and furniture to detect burglars. However, if it uses the information of a camera to do so, it must receive a real-time stream of its captures.

Behrendt et al. (DAMJANOVIC-BEHRENDT; BEHRENDT, 2019) divide the roles of a DT into two main building blocks: the Interoperability Manager and the Virtualization Manager. The latter comprises the following three sub-components:

- **Data Manager:** This has two sub-components: the **Data Acquisition** and the **Data Analytics** managers. The former is responsible for acquiring different forms of data. To this concern, this module may collect data from a real-time stream from the sensors placed on the house or make queries to external services such as a weather API. Non-volatile (unchangeable) data, such as the specifications of the house appliances or its BIM model, may also be collected. In turn, the Data Analytics component stores the data and prepares it for further operations and queries. The decision of how to keep each type of data must consider factors such as scalability and fault-tolerance, which are also part of a DT's duties. Examples of free tools that may compose this module are Kafka Streams and MongoDB, a NoSQL database.
- **Models Manager:** This includes data computation models, which perform the processing, in batch or real-time, and analysis of the data stored by the data manager. This last module can later receive and store the results to aid future computations.

An example of a computational model is an object recognition model that detects people in images. Examples of open-source tools highly used for machine learning and analytics are Tensorflow, Keras, and Pytorch. In turn, examples of software for batch and real-time processing are Apache Spark and Kafka Streams, respectively. The Model Manager also includes data representation models, which allow data storage, exchange, and searching.

- **Services Manager:** This is responsible for orchestrating the functionalities of the DT. As the DT may have different users with different interests, this module also supports a custom operation of the system. It may comprise authentication and authorization mechanisms that protect a home's information from non-allowed people. Moreover, they may allow only determined residents to use a particular feature. This module also comprises visualization dashboards, which provide a graphical display of different functionalities, such as the real-time monitoring of the house's sensors.

Besides the virtualization manager, the DT also comprises the **Interoperability Manager**, which interfaces the real and the virtual domains. Its **Monitoring Manager** connects the physical sensors to the data manager component. In turn, the **Decision-Making Manager** sends the findings of the data processing to the home's actuators. For example, it sends a signal to turn on an alarm in the presence of a risk of fire. The last sub-component of the **Interoperability Manager** is the **Simulation Manager**. It provides the simulation, which is structured based on the visualization dashboards of the **Virtualisation Manager**.

4 USE CASES

We divide this chapter into two sections, one for each use case we present. The first one focuses on the interrogative role of the Digital Twin. It allows the real-time monitoring of the house's sensors, analyses, and user interactions. The second, in turn, focuses on the predictive role, using simulation data to make predictions about the heating of a house and decisions regarding that. The two sections detail the demonstrations' purposes, design, and implementation.

4.1 Power consumption visualization

The first use case that we studied is a Digital Twin that analyzes a home's power consumption to help the residents rationalize it. Electricity in our lives has become ubiquitous: citizens of well-off countries are dependent on it for many of our chores and sometimes do not even notice. That is why producing energy is a significant contributor to climate change, representing 27% of the emission of greenhouse gases (GATES, 2021). Thus, this use case aims to give people more information about the subject and raise environmental awareness.

The idea is to constantly measure the power usage of some home appliances and make a virtual color-coded gradient map based on these data. The more intense the color of the virtual counterpart of a device, the more energy it is currently demanding. The users can choose the gradient color and execute some queries on the past data. For example, they may want to know the consumption average of each instrument in the last month. The virtual system then pauses the live procedure and assigns new colors to the appliances based on their average consumption for the period stated by the user.

Fig. 4.1 shows the system's architecture, which relies on the model we presented in the previous section. Its building blocks we detail below.

4.1.1 Physical System

The physical system ideally consists of a house with power sensors attached to as many electrical household appliances as possible. However, since this scenario is difficult

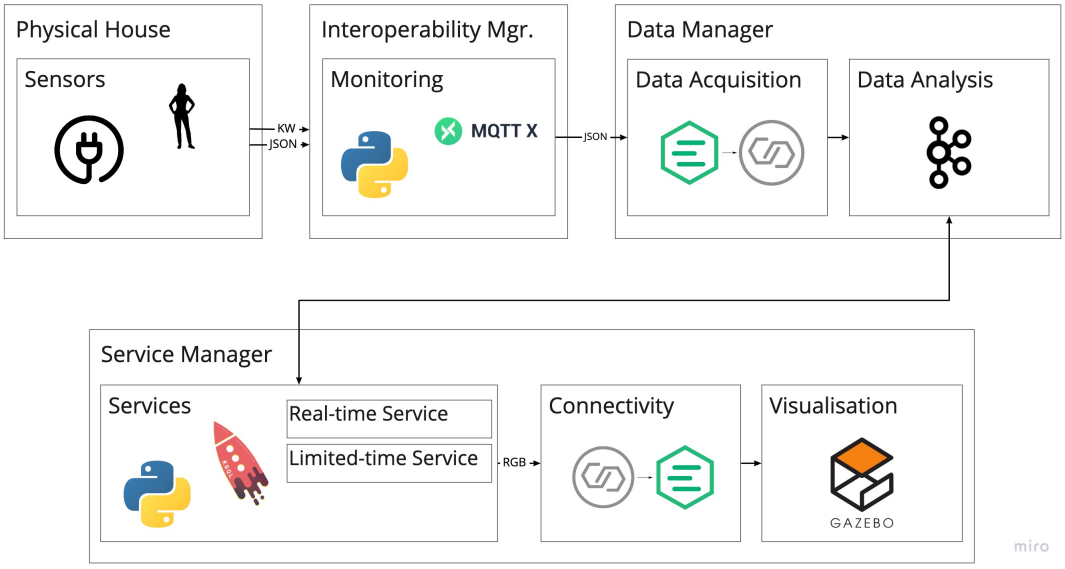


Figure 4.1 – A DT's flow diagram

to obtain, we simulated it using an open-source data set[1] in a CSV format that contains the readings of house appliances in kW from a smart meter. It also includes the timestamps of the measurements, which space themselves one second apart. The user's commands in this system are treated just like sensors, following the same data flow as the sensor's data. Table 4.1 shows a part of the data set, with some of its column names modified for simplification.

Time	Dishwasher	Oven	Fridge	Microwave
1451624400	3.33E-05	0.0207	0.12415	0.004066667
1451624401	0	0.020716667	0.12353333327.00	0.004066667
1451624402	1.67E-05	0.0207	0.123533333	0.004066667

Table 4.1 – Sample of "Smart Home Dataset with weather information"

4.1.2 Monitoring

A Python script parses the CSV line by line. In each iteration, the program builds a message in the JSON format for each machine and then publishes them in an MQTT topic named *equipment*. Before proceeding to the following line, the program waits one second, which is the same time that spaced the data in the CSV document. Next, we describe the schema of the messages:

[1]<https://www.kaggle.com/taranvee/smart%2Dhome%2Ddataset%2Dwith%2Dweather%2Dinformation>Visited on 15. Mar. 2022.

- **equipment:** (required). A string that contains the name of a piece of equipment;
- **power:** (required). A double value containing a power consumption measurement, in kW;
- **measured_time:** (required). A long integer that stores the timestamp when the measurement was taken;

Thus, an example of a message the script sends to the broker is:

```
{
    "equipment": "Fridge",
    "power": 0.1,
    "measured_time": 1647279554
}
```

Another MQTT topic named *users* expects users' commands to change the color of the visualization or the mode of operation. These messages also must be JSON formatted following the schema below.

- **color:** (optional). An integer defining the color to display the gradient map; The current options are red, green, and blue, and their respective codes are 1, 2, and 3.
- **command:** (optional). A string with the value of "limited" to indicate the analysis should consider a closed and fixed period, or "realtime" to tell the DT to display the current values captured by the sensors; If the user chooses the first option, the following two fields are required; otherwise, they are disregarded.
- **from_date:** (optional). A string in the format "yyyy-MM-dd" to indicate the initial date of the closed period analysis;
- **to_date:** (optional). A string in the format "yyyy-MM-dd" to indicate the final date of the closed period analysis;

This module ideally comprises a user-friendly UI that would translate the user needs to the JSON format and publish in the user's topic. This interface could be, for example, a mobile application, which could increase the User Experience (UX) and accessibility of the system mainly for people non-familiar with the data structure. However, this component's implementation did not enter the scope of this work. We then "simulate" it with the MQTT Client MQTT X, another EMQ product that offers a desktop UI that eases publishing in topics or subscribing to them. We chose this platform because it is easy to use and monitor the execution of the whole system, even though it may not be so

simple for non-developers users. Also, it is available to download for free on their official website[2].

So, in our implementation, if the user wants, for example, to change the color of the visualization to green, they must first download, install, and open the MQTT X platform. Then, they must start an MQTT client and write the following message, already JSON formatted.

```
{"color": 2}
```

Moreover, they must write in a field the topic's name in which they want the command to be published, i. e., *users*, and hit *publish* button.

We placed the MQTT X platform under the monitoring component because it is here where an ideal mobile application would be. Also, although it does not perform any transformation to the user's input, it does act as an interface between the physical and the digital domains.

4.1.3 Data Manager

For the data acquisition, an EMQX MQTT broker receives the messages sent by the previous component. We chose this broker because it is open source and has a massive scale, connecting millions of IoT devices efficiently[3]. Since we use Apache Kafka in the next component, the Data Analysis, it was necessary to bridge the data from the MQTT broker to it. Therefore, we used Kafka Connect, which subscribes to the two MQTT topics mentioned and publishes in correspondents Kafka topics. Since this module *brings* data to Kafka, it is called Kafka Connector *Source*.

Confluent makes an MQTT Connector Source available on its Confluent Hub[4]. We must only install and configure it with JSON files whose properties Confluent defines on its website. The number of files one must write is the number of MQTT topics in the system. Since we mapped each of the two MQTT topics to a different Kafka topic, two files were necessary.

For processing our application's data, we used Apache Kafka's software. As explained, we count in two topics, *users* and *equipment*, that keep all data received.

[2]<https://mqttx.app/> Visited on 15. Mar. 2022
[3]<https://www.emqx.io/> Visited on 16. Mar. 2022
[4]<https://www.confluent.io/hub/confluentinc/kafka%2Dconnect%2Dmqtt> Visited on 15. Mar. 2022

4.1.4 Services

The Services Manager has two services, the *real-time* and the *limited-time*, and controls them with Python script. We illustrate both operation modes in Fig. 4.2 and detail them below.

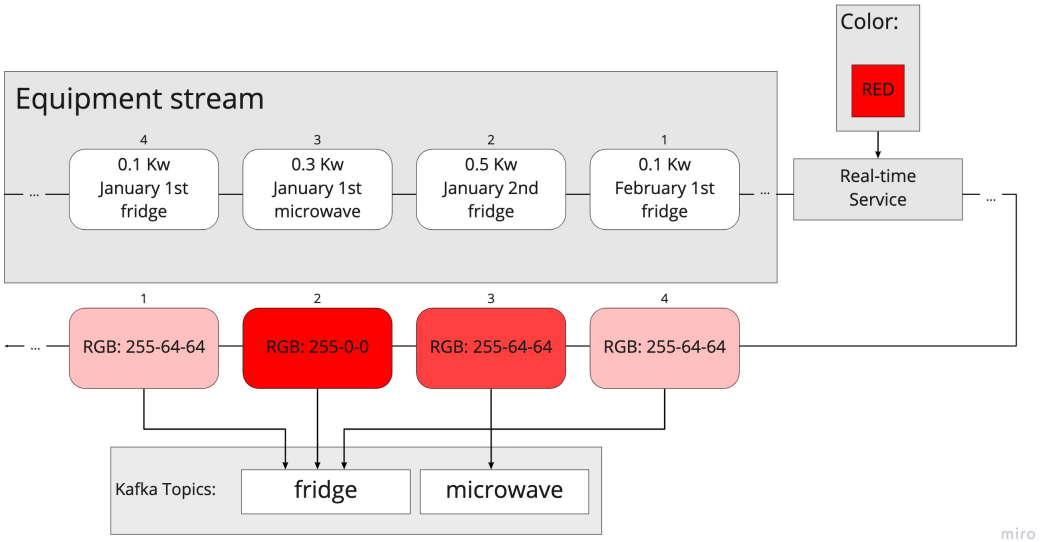
The system initially starts in real-time mode, listening to the *equipment* topic. For each message it receives, the procedure converts its power value to a color, an operation that rests on a gradient composed of 5 different shades of the last color chosen by the user, red by default. Each of the five shades corresponds to a range of energy consumption. We show the mapping from power ranges to a red color gradient in Fig. 4.3. The program represents the color obtained as a hyphen-separated string containing its RGB and publishes this information in a Kafka topic whose name is in the *equipment* key of the same message.

If the system receives a message from the *users* topic with the *command* field equal *limited*, it blocks the thread responsible for the real-time processing of the events and starts the *limited-time* mode. This procedure makes an average, per device, of the power values measured during the period determined by the *from_date* and *to_date* fields of the said message. The algorithm then converts the obtained results to colors, following the same principle above, and publishes them on all the output topics. If a new message reaches the *users* topic with the *command* field equal to *realtime*, the program unblocks the live-processing thread and returns to coloring the machines based on their current state. If the program receives a message containing a *color* field, independently of the operation mode, it saves its value and considers it for the following transformations.

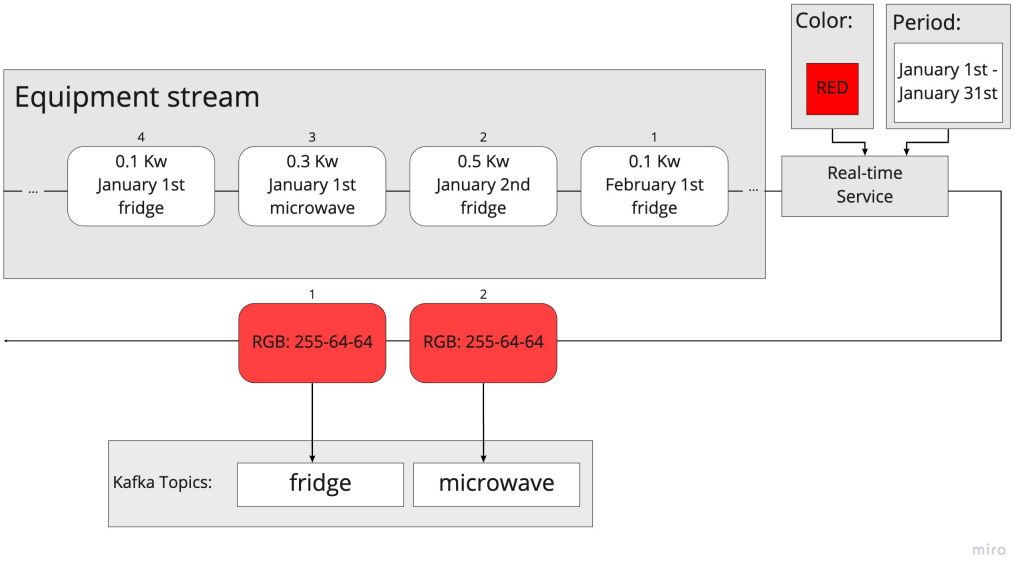
For processing the streams, we used ksql API written for Python. It queries the data stored on the Kafka topics, part of the Data Analysis component. So, there is a constant exchange of queries and their results between the Service and Data managers. We explain below how we achieved the feature of getting the average power consumption in a certain period. We first describe the steps required for the ksql Command-Line Interface (CLI) and later list some adaptations for dealing with Python's environment.

Let us suppose we wanted the average of the machinery in February of 2022. For achieving that, we would execute the following steps on kSQL terminal assistant.

1. Set configurations. The first is to make a pull query without a WHERE clause, i. e., the last step, and the second is to handle past events.



(a) *Real-time* service operation



(b) *Limited-time* service operation

Figure 4.2 – The two operation modes of the system

Color:					
Power:	< 0.1	>= 0.1 && < 0.2	>= 0.2 && < 0.3	>= 0.3 && < 0.4	>= 0.4

Figure 4.3 – Association between power, in kW, and color

```
SET 'ksql.query.pull.table.scan.enabled'='true';
SET 'auto.offset.reset'='earliest';
```

2. Create a stream from the equipment topic;

```
CREATE STREAM EQUIPMENT_STREAM \
(equipment varchar, \
power double, \
measured_time BIGINT)
WITH (KAFKA_TOPIC='equipment', VALUE_FORMAT='json');
```

3. Create another stream transforming the *measured_time* BIGINT field to a DATE type. As this field contains a value in seconds and kSQL's function TIMESTAMPTOSTRING expects a value in milliseconds, we made this conversion.

```
CREATE STREAM CONVERTED_STREAM \
AS SELECT equipment, power, \
CAST( TIMESTAMPTOSTRING( \
    measured_time * 1000, 'yyyy-MM-dd' \
) AS DATE) AS MY_DATE
FROM EQUIPMENT_STREAM;
```

4. Make a query on the stream created in the previous step to obtain the average of the events in the given period. This operation selects the events based on their *MY_DATE* field, which originates from *measured_time*, assigned by the Monitoring component 4.1.2. Therefore, this operation selects the events based on when they happened, not when the broker received them. If we wanted the opposite, we should consider the metadata field named ROWTIME, assigned by the broker. This query is a *pull query*, which, unlike the *push query*, terminates after returning a single result for each machine. Each of the results is JSON-formatted and contains an appliance's name and power consumption average in the period. The latter determines the color of its 3D model, the former the topic to publish this information.

```
SELECT equipment, AVG(POWER) AS AVG_POWER
FROM CONVERTED_STREAM
WHERE MY_DATE >= '2022-02-01' AND \
MY_DATE <= '2022-02-31'
GROUP BY EQUIPMENT;
```

Implementing the steps above using Python's kSQL library required some minimal changes. The first one regards to step 1. We found out that the CLI cached the configurations listed and sent them with every subsequent request. Since Python's library doesn't do the same, we sent the following configuration in items 3, and 4.

```
{"ksql.streams.auto.offset.reset": "earliest"}
```

The second setting, which allows pulling queries, was not necessary. We substituted the pull query of item 4 for a push query because the previous one was improperly terminating after the first event it emitted, which we believe is a bug in the software. Our solution was to make a push query and hard-code the number of events we wanted to handle, equal to the number of appliances. After we parsed these events, we manually ended the query.

The program executes steps 1 (modified) to 3 at its initialization and step 4 every time the user sends a command with the *command* key equal to *limited* and *to_date* and *from_date* not empty. After the initialization, while the program does not receive any command of this type, it operates in real-time mode. In this case, it executes the following push query constantly.

```
SELECT * FROM EQUIPMENT_STREAM EMIT CHANGES;
```

We add the following configuration to the query, similar to the one we explained above but stating that we want only the events that follow the query execution in the result.

```
{"ksql.streams.auto.offset.reset": "latest"}
```

Similarly to item 4, the results consist of JSON messages containing the current power measurement of an appliance and its name. When it starts the *limited-time* operation, the system ends this query and redoes it when it re-enters the *real-time* mode.

4.1.5 Connectivity

This module is responsible for connecting the services, which work with Kafka, to the visualization component, which works with MQTT. Thus, the connectivity block does a similar process to the one of Data Aquisition but in reverse order. Since this module transfers data from Kafka to the MQTT broker, it is called Kafka Connector Sink.

```
{
    "name" : "mqtt-sink",
    "config" : {
    "connector.class" : "io.confluent.connect.mqtt.MqttSinkConnector",
    "tasks.max" : "1",
    "mqtt.server.uri" : "tcp://broker.emqx.io:1883",
    "topics" : "Fridge, Microwave, Dishwasher, Furnace",
    "confluent.topic.bootstrap.servers": "localhost:9092",
    "value.converter":"org.apache.kafka.connect.storage.StringConverter",
    "value.converter.schemas.enable": "false",
    "confluent.topic.replication.factor": "1",
    "confluent.license":"",
    "mqtt.retained.enabled":"false"
    }
}
```

Figure 4.4 – Connector Sink's configuration file

We set this component up with a JSON configuration file, shown in Fig. 4.4, containing all the Kafka topics it must subscribe to, whose names correspond to the machinery. The connector then forwards the events to MQTT topics with the same name. We again used the Connector for MQTT available on Confluent Hub and EMQX as our broker.

Since one of the roles of the Service Manager is to orchestrate data and provide visualization dashboards, we place here the services, the visualization system, and the connection between them.

4.1.6 Visualization

To provide the users with a visualization of the color-coded gradient map, we chose the software Gazebo. It is an open-source robotics simulator but can also fit other applications that involve 3D modeling, which is our case. The platform provides us with some ready-made 3D models, which comprise mesh files formatted as STL, Collada, or OBJ. Moreover, it must contain an SDF file containing the path to the mesh's file. The SDF format has an XML-like syntax and can represent a single model or a union of many, called a world.

Therefore, the platform allowed us to design a 3D copy of our physical house and also write scripts in C++ to control the household appliances. These programs are called plugins and can be one of three types: model, sensor, or visual. In our case, we needed the latter one, which allows us to control the rendering process of the 3D object, and thus change its color.

The script we wrote is then a subclass of *VisualPlugin* and has an MQTT client.

```
<model name='fridge'>
    <static>1</static>
    <link name='fridgelink'>
      <collision name='collision'>
        <geometry>
          <mesh>
            <uri>model://fridge/meshes/fridge.obj</uri>
            <scale>0.012 0.012 0.012</scale>
          </mesh>
        </geometry>
      </collision>
      <visual name='visual'>
        <plugin name='color_change_plugin' filename='libcolor_change_plugin.so'>
          <topic>Fridge</topic>
        </plugin>
        <geometry>
          <mesh>
            <uri>model://fridge/meshes/fridge.obj</uri>
            <scale>0.012 0.012 0.012</scale>
          </mesh>
        </geometry>
      </visual>
    </link>
    <pose>0.674415 3.95475 0 0 -0 0</pose>
</model>
```

Figure 4.5 – Extract from World's SDF file

The class expects an initialization parameter indicating the name of the MQTT topic to which its client must subscribe. Each of the models then receives messages containing the color to render itself. These programs subscribe to MQTT topics instead of Kafka topics because of the lightweight characteristic of the protocol, which allows more efficiency in the visualization. Also, it was a design decision to make the MQTT broker our system's input and output gateway.

The names of topics are the same as the machinery names, and we pass them as parameters to the instances of our control plugin in the SDF describing the model. For example, Fig. 4.5 shows the part of the world's SDF that represents the fridge's model. It references the *color_change_plugin*, which is the name of the program described above. Moreover, it references the file *fridge.obj*, which is inside a folder that comprises additional information to display the object, e.g. texture files, and some documentation. For simplifying the picture, we cut off some tags under the model's one. As they specify the physics and surface of the appliance, we thought they would not be relevant.

To conclude, Fig. 4.6 shows the real-time functioning of the algorithm. The moment we took the screenshot, the oven was the most energy-consuming object; the fridge was the second; the rest had an insignificant usage.

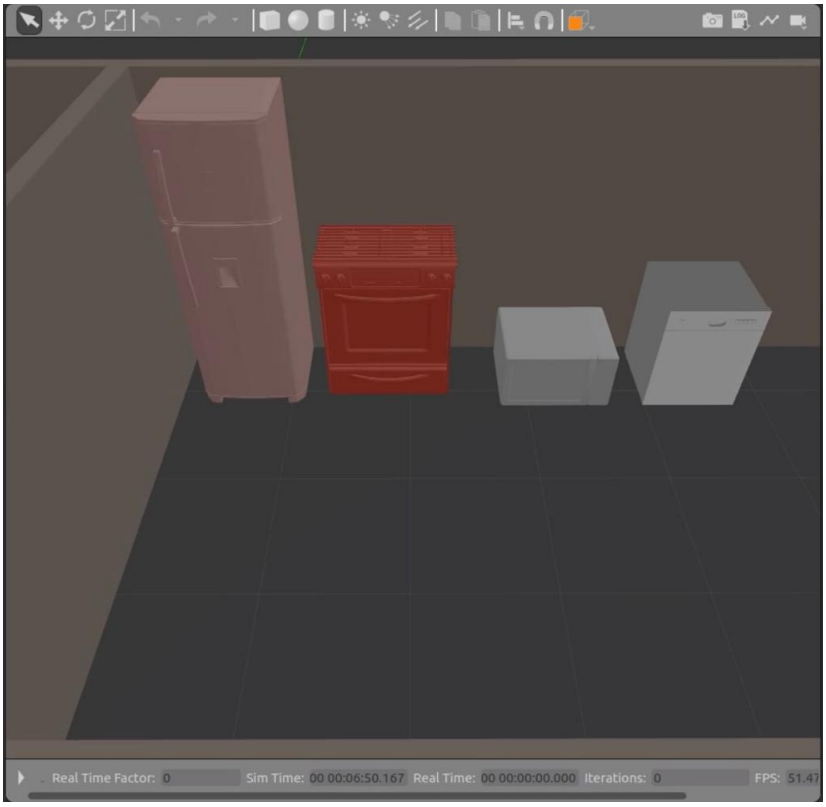


Figure 4.6 – First use case result

4.2 Indoor temperature control

This use case aims to daily obtain the right time to turn on power sources in different rooms of a house so that its indoor temperature is equal to a target value when the user arrives home. With the help of a simulator, we generated a data set that describes the heating of different rooms. Our system consults this information every time it receives a temperature measurement from the physical world to conclude whether a heater should be turned on or not. We consider the target temperature to be 20ºC and that every day the user leaves home at 8 am and returns at 6 pm. This section first explains the implementation of the scenario described and, in sequence, its evaluation.

4.2.1 Implementation

We designed the second use case similar to how we did the first, taking Behrendt et al.'s (DAMJANOVIC-BEHRENDT; BEHRENDT, 2019) work as a reference. The final architecture we show in Fig. 4.7 and its building blocks we describe below. As we will further describe in the next section 4.2.2, we evaluate the system's operation in the first

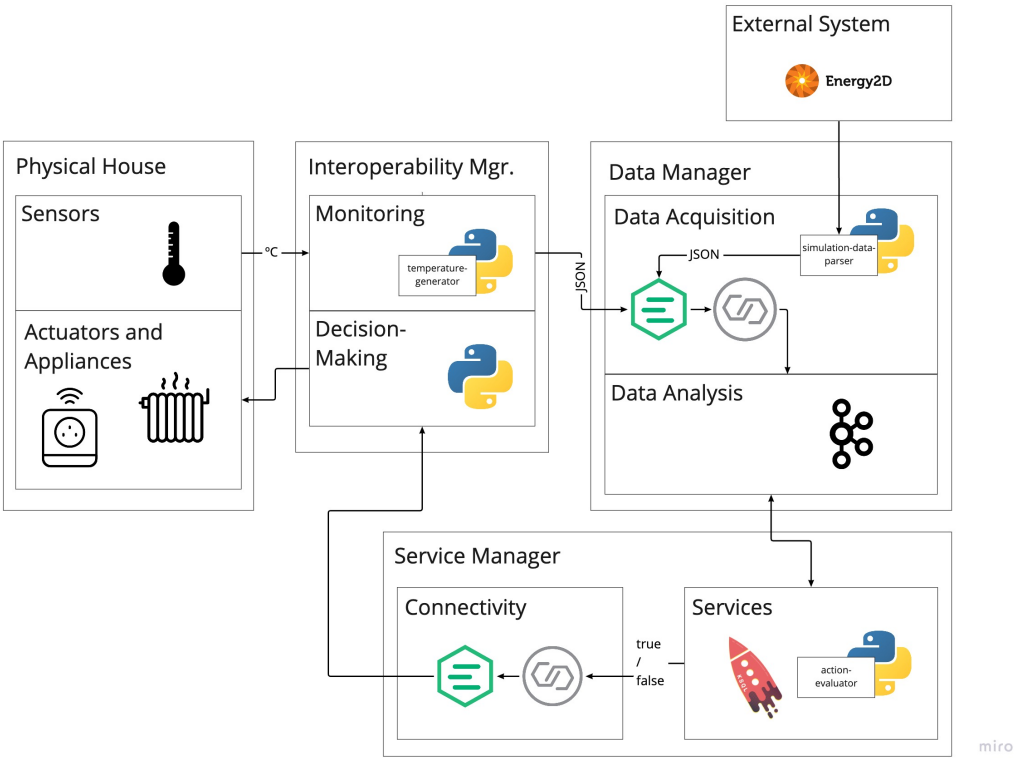


Figure 4.7 – Second use case's architecture

four months of the year and three different rooms. Each iteration of the system considers a room and a month, information which the scripts that compose the architecture receive as an execution argument.

4.2.1.1 Physical System

The physical system ideally comprises several rooms of a house. Each space would contain temperature sensors and a power source, such as an electrical heater, that our algorithm would switch on and off. One way to achieve this autonomy of the thermal appliances would be to connect them to smart power outlets, which would receive the final statements from our system and control the heaters accordingly.

Since this scenario was unfeasible for us, we emulated the the readings of the thermometers with a data set we obtained online. It consists of a CSV file containing hourly temperature measurements of the German city Stuttgart. Table 4.2 shows a piece from this document, which we further refer to as *stuttgart-weather*.

month	day	hour	temperature
1	1	1	-1.9
1	1	2	-3.6
1	1	[...]	
1	1	24	-6.7
1	2	1	-6.3
1	2	2	-5.9
1	[...]		
1	31	24	1.7
2	1	1	1.5

Table 4.2 – Sample of *stuttgart-weather* file

4.2.1.2 Interoperability Manager

The Interoperability Manager has two sub-components: the Monitoring and the Decision-Making managers. The first ideally comprises a program that receives thermometers' readings and sends them to our system by publishing them in an MQTT topic. The latter, in turn, would include a program that controls the heaters based on the information our system published in an MQTT topic. Therefore, as in the first use case, the MQTT broker is our system's input and output gateway.

As we do not have an actual physical system, we wrote one Python script called *temperature-generator* that takes a month as input and publishes in an MQTT topic the information respective to this month from the *stuttgart-weather* data set. Moreover, the script subscribes to an MQTT topic called *heater-actions*, from which it knows when it is time to turn on the heater. Instead of sending this information to smart power outlets, the program uses this information to retrieve a CSV file containing, for each day, the respective time at which the system decided to turn on the heater and the temperature at that moment. We refer to these output files as *heater-routine* files, and Table 4.3 shows a segment of one generated during our test executions.

date	actiontime	temperature
1/4	17:15	17.73
2/4	16:00	14.7
3/4	17:15	17.75

Table 4.3 – Sample of a *heater-routine* file

The mentioned *temperature-generator* script does some preprocessing on the *stuttgart-weather* file before forwarding its information to the MQTT topic, called *temperature*. The preprocessing selects the rows whose *month* column equals the one given

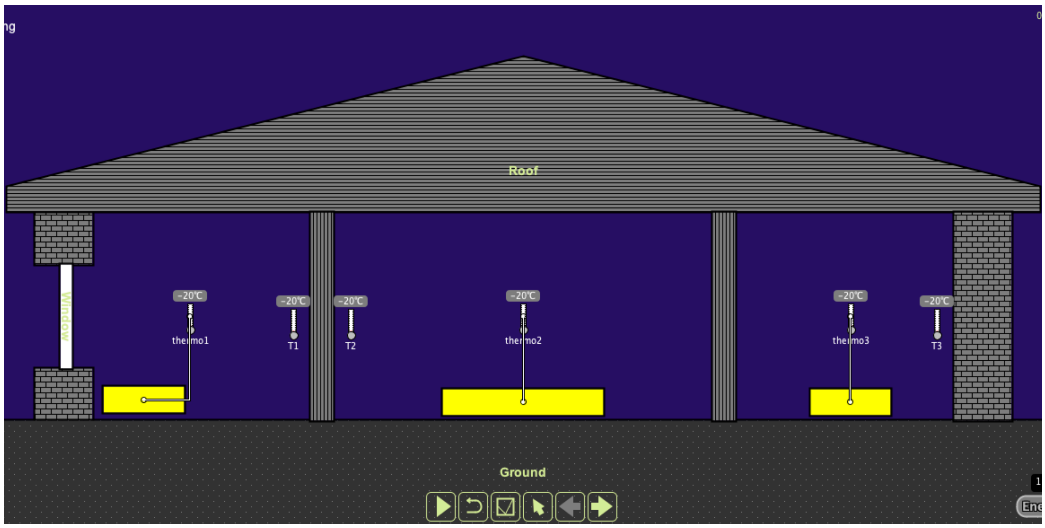
as a parameter and whose *hour* column is between 8 am and 6 pm, which we consider the time that the user is not home. Moreover, since our data set contains hourly measurements, we decided to provide a better precision to the whole system by generating three intermediate values between its rows. The newly created values follow a linear progression. For example, supposing the temperatures measured at 10 and 11 am are 10 and 20 degrees, respectively, the algorithm assumes that the temperatures at 10:15, 10:30, and 10:45 were 12.5, 15, and 17.5, respectively.

The script then reads the resulting file line by line, now spaced by 15 minutes. When the parser reaches the end of a day, i. e., the line with the time equal to 6 pm, or receives a command from the *heater-actions* topic to switch on the heater, it adds a new row to the output file and starts reading the information of the next day. The final output file, *heater-routine*, lists all the days of the given month and their respective *action-time*, i. e., the time the system determined to turn on the heater on that day. If the heater was unnecessary on a day because the indoor temperature naturally reached 20ºC before 6 pm, its *action-time* column is *"none"*.

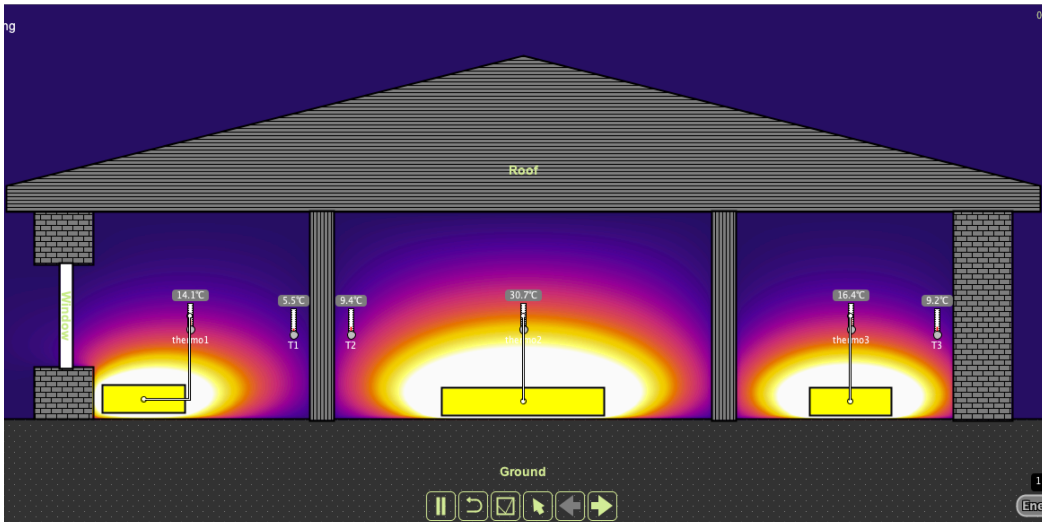
4.2.1.3 Data Acquisition

The data acquisition module obtains data from the monitoring component and also from an external system that provides data describing a house's heating. This simulation software, called Energy 2D, allowed us to create three rooms, which we named *smallroom*, *smallroomwindow*, and *bigroom*. The first two have equal dimensions, but the second counts on a window, so it loses more heat to the outside world. The third room is larger than the others and does not contain any windows. The house includes three power sources, one in each space, with dimensions and wattage proportional to the respective room's size. The heaters pause their operation when their attached thermostats measure 20 degrees Celsius or higher and continue otherwise. Each room also comprises one more thermometer more distant from the heater, where it takes longer to warm. Fig. 4.8a and Fig. 4.8b show the complete system in the simulation's initial and intermediary states, respectively.

We simulated the house's heating for the equivalent of circa 13 hours. After that, the assistant software provided us with a discrete sequence of events in a table format. Each row represents one of these exogenous events and contains a timestamp and six temperature readings, one for each thermometer in the designed house. The inter-row interval equals 100 seconds, totaling approximately 470 rows. Table 4.4 shows the initial



(a) Simulation's initial state



(b) Simulation's intermediary state

Figure 4.8 – Simulation states on Energy 2D's interface

lines of the provided data-set with the values rounded to 2 decimal places for simplification. The first column indicates the simulation time in seconds. The following three, labeled TX, denote the readings from the thermometers placed in room X, and the final three, thermoX, are the thermostats in room X.

time	T1	T2	T3	thermo1	thermo2	thermo3
100	-6.96	-7.01	-6.97	-16.93	-19.63	-15.94
200	-4.61	-5.05	-4.64	-11.68	-17.31	-10.56
300	-3.24	-4.05	-3.29	-8.05	-14.28	-7.12

Table 4.4 – Sample of Energy 2D's resulting table

Having the resulting data from the simulator, we ran a Python script called *simulation-data-parser* that transfers the data from the columns T1, T2, and T3 to their room's respective MQTT topics. For example, the program maps each row of T1 to a JSON message containing a key to store T1's value and another to store the row's time. Then, it publishes the message on the respective topic, *small-room-window-stream*. When it finishes parsing all lines, it repeats the process for the other two mentioned columns.

Moreover, the Data Acquisition component transfers data from the three MQTT topics to respective Kafka topics with the same names. Same as we did in the first use case, the MQTT broker we used was EMQX, and the MQTT-Kafka mapping we did with the MQTT Connector Source available on Confluent Hub.

One might ask why the *simulation-data-parser* script does not publish the messages on the Kafka topics, avoiding this hop. The goal is to maintain consistency with other components and make all data leaving and entering our system pass through the MQTT broker.

4.2.1.4 Data Analysis

As explained, our system stores all its operational data in Kafka streams. It counts on three topics that store the time series that represent the heating of the three rooms, which we simulated in Energy 2D. The names of the three streams concern their rooms: *small-room-window-stream*, *small-room-stream*, and *big-room-stream*. An additional stream, the *temperature-stream*, receives real-time information from the physical world. All four streams share the same structure. Their messages contain two keys, *measures_time* and *temperature*, whose value's type and origin vary with respect to the stream.

The second field has a float value representing a temperature measurement in Celsius. This temperature comes from the *stuttgart-weather* file in the case of the *temperature-stream*'s messages and from the simulation software measurements in the case of the other three streams' messages.

The *measured_time* key contains a string with the format "HH:MM" in the *temperature-stream* messages. This time information also originates from the *stuttgart-weather* file. In the real world, the field would contain the event-time, i. e., the current time in the wall-clock of the *temperature-generator* script's environment. However, in our case, it has nothing to do with when the program operates or when the Kafka broker receives the events. In the messages present in the other three streams, the same key contains a float value representing the time measurement in seconds during the simulation on Energy 2D. Again, these values have nothing to do with their event-time or ingestion-time. The latter is stored by the ROWTIME metadata field but is not used in the system.

4.2.1.5 Services

In parallel with the *temperature-generator* script we mentioned in 4.2.1.2, we run another one that subscribes to the *temperature-stream* and receives messages containing temperature measurements and time information. For each received message, the algorithm calculates whether it is necessary to turn on the heater or not, publishing this Boolean decision in a Kafka topic. Fig. 4.9 shows its operation. Before starting, the program obtains as input a room name, which determines which simulation stream it should consult during its operation. Whenever the algorithm receives a message, it queries the mentioned stream to know when the simulation had a temperature close to the one received. Since the query is a kSQL query, some SQL operations, such as ORDER BY, are not present. Thus, the query statement fetches the lowest temperature among those higher than the one received. The resulting temperature might not be the closest to the requested but is close enough since the spans between the simulated temperatures are not big. Given the timestamp resulting from the query, the algorithm firstly calculates how long the simulation took to reach the target temperature, i. e., 20ºC. Then, considering the time information on the message from the physical world, it calculates how much time is left until the user returns home at 18 o'clock. The algorithm decides the heater should be turned on whenever this latter period is less than or equal to the previous. It finishes the procedure by publishing the Boolean decision in a Kafka topic named *heater-actions* and waits for more messages on the *temperature* topic.

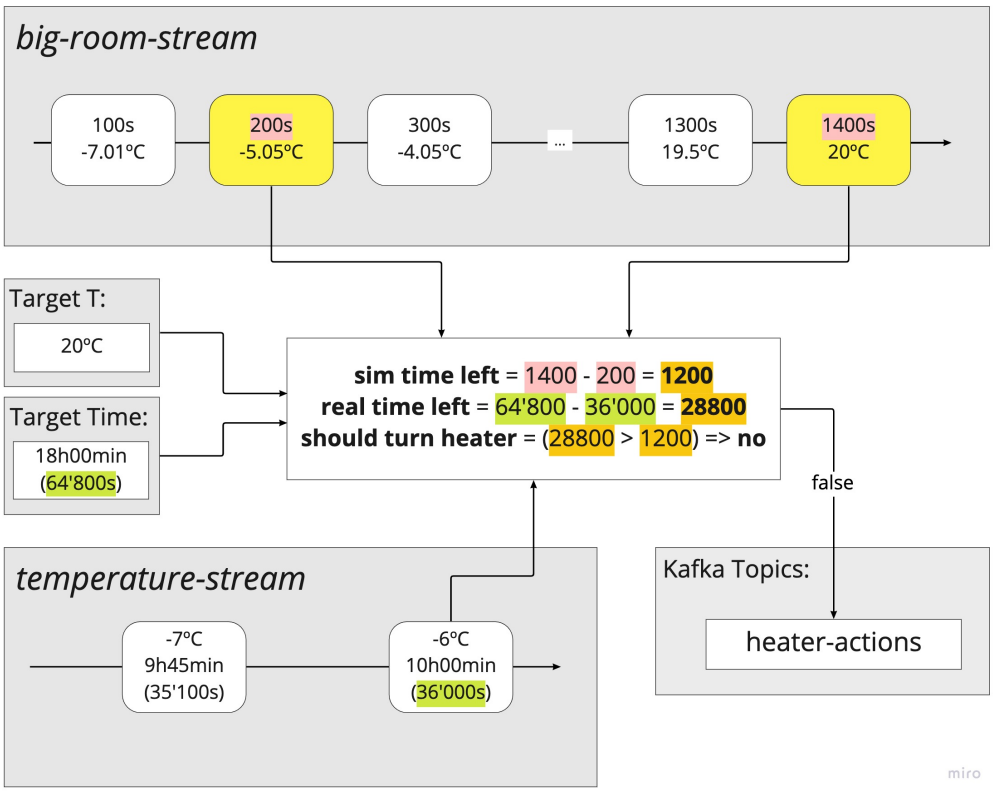


Figure 4.9 – Data Analysis operation

4.2.1.6 Connectivity

This module forwards each message on Kafka's *heater-actions* topic to an MQTT topic with the same name. For doing so, we again used Kafka's MQTT Connector Sink available on Confluent Hub. Its configuration file is very similar to the one of the first use case, shown in figure 4.4. It differs only in the *name* and *topics* keys.

4.2.2 Evaluation

To evaluate the second use case of the system, we compare it to one that turns on all the heaters every day at the same fixed time, which we further refer to as *fixed-time* system. We analyzed the behavior of the two approaches during the first four months or the year and compared them regarding the thermal comfort they provide and the power demanded. Fig. 4.10 shows the steps we made for evaluating the system, which we explain below.

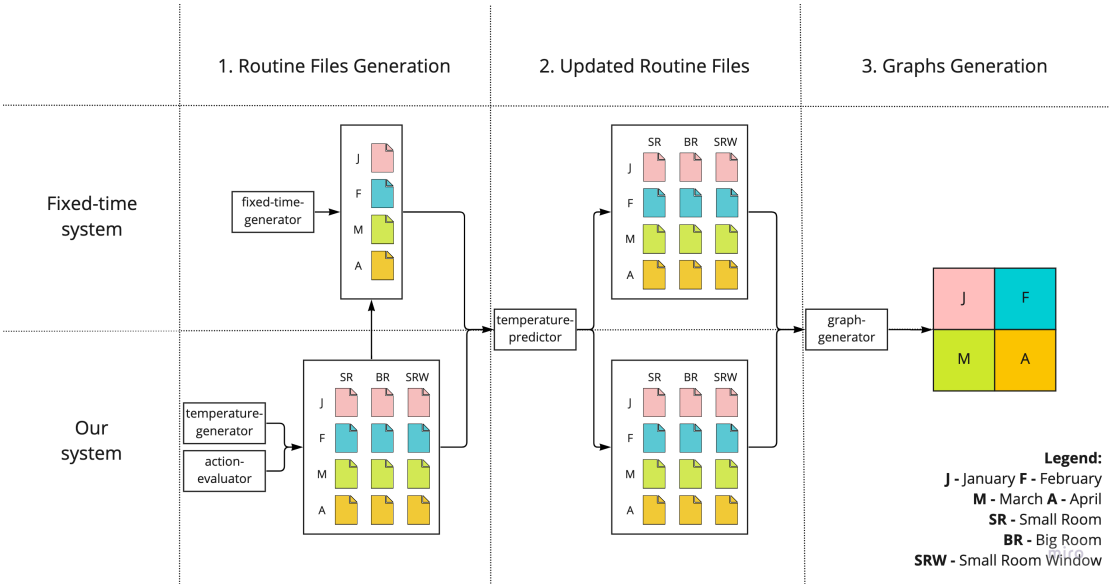


Figure 4.10 – Evaluation process

4.2.2.1 Generation of heater-routine Files

This step generates the files named *heater-routine* and previously described in Sec. 4.2.1.2. To generate the data regarding our system, we run, in parallel, the scripts *temperature-generator*, also described in Sec. 4.2.1.2, and *action-evaluator*, described in Sec. 4.2.1.5. The first one receives a month as a parameter, the second a room name. Therefore, we executed the two programs together twelve times and obtained twelve files. This number is the number of combinations between the three rooms and the four months.

For generating the files that simulate the *fixed-time* system, we made four files, one for each considered month. They have the same structure as the files cited above, but each file has the same value in all rows of column *actiontime*. We determined this fixed time for each month from the average time among the files generated for our system for the same month. So, for example, for March, we parse the three files respective to this month generated previously in this step. Then, we sum all the action times and divide by the number of days, which in this case is 93, i. e., 31 times 3. We then rounded the values with the frequency level of 15 minutes. Having calculated the *actiontime* for the month, we fetch the temperature in which our city of interest was at that moment. If it is a whole hour, we straightforwardly consult the respective row of the *stuttgart-weather* data set described in 4.2.1.1. Otherwise, we calculate the temperature at that moment similarly to how we generated the inter-row values in Sec. 4.2.1.2, i. e., assuming the temperature grows linearly between two full hours.

4.2.2.2 Estimation of temperature at the target time

This step is about estimating what would be the temperature of each room if we switched on their heaters at the calculated action times. So, we wrote a script that parses a file generated in the previous step and returns another one with two additional columns: *time-heater-on* and *simulated-temperature*. Besides the file to parse, the program expects a room name as a parameter, which determines the simulated stream to consult. For our system, we produced 12 files in the previous step, and each generates a single updated one. For the *fixed-time* system, three new files originate from each fixed-time system's file since the final temperature estimation varies by room. Thus, the total number of executions of this module is 24.

For estimating the temperature at the target time, the program executes the following steps for each line of the given file:

1. Gets the moment the simulation had the given temperature For this step, the script executed the following kSQL query, in which the parameter temperature is the rows' temperature column, and the stream_name corresponds to the room name's input.

   ```
   select measured_time from {stream_name} \
   where temperature >= {temperature} \
   emit changes limit 1;
   ```

   Again, we don't fetch the closest temperature to the searched one. Instead, we look for the lowest temperature that is higher than it.

2. Calculate how much time remains from the line's *action-time* field until the target For this step, we calculate the difference between the target time, 18 o'clock, and the one the system decided to turn on the heater that day.

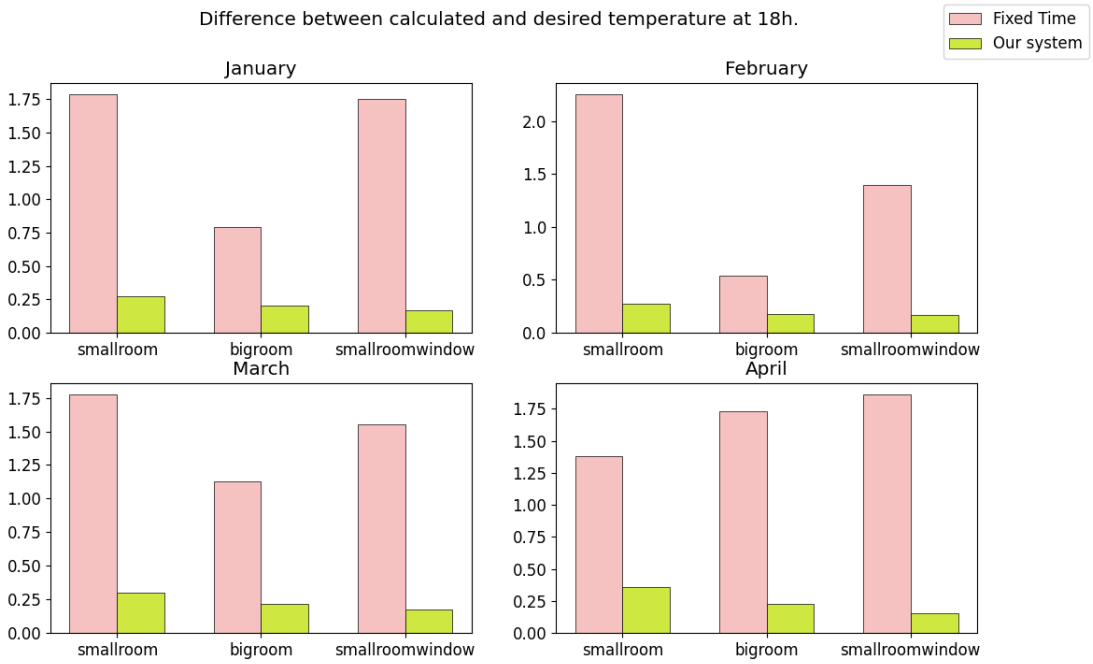
3. Sums the two last periods
   Gets the temperature the simulation had at the latter moment
   For this step, we used a query similar to the one in Step 1:

   ```
   select temperature from {stream_name} \
   where measured_time >= {measured_time} \
   emit changes limit 1;
   ```

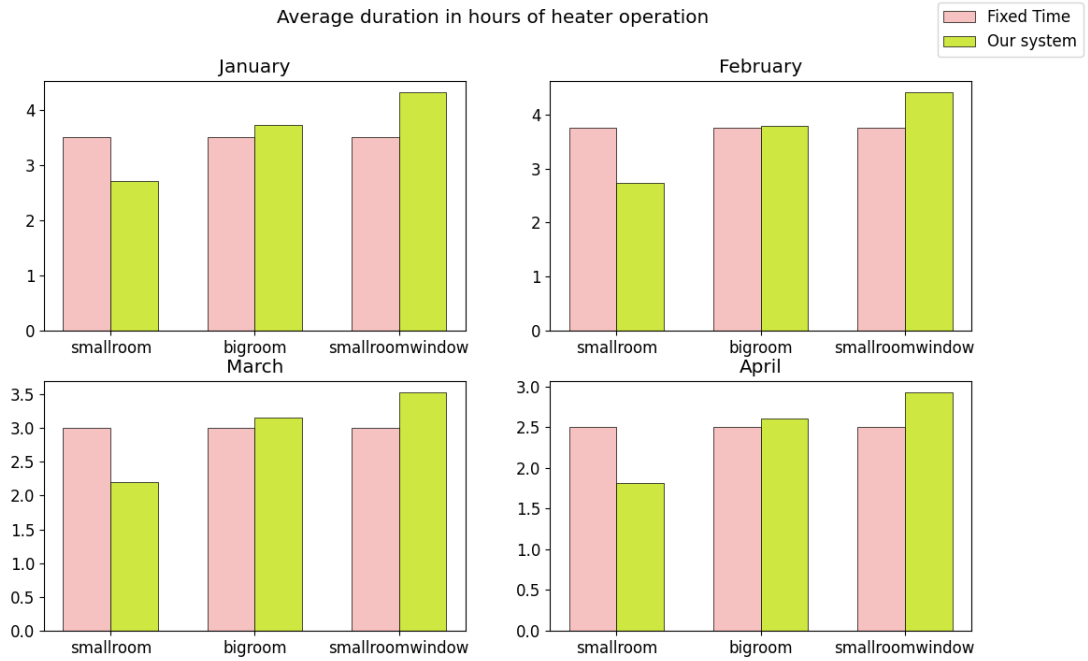
4.2.2.3 Graph Generation

Having the temperature estimation at the target time and the heater's operation duration by day, this step analyzes our system. Fig. 4.11a shows the comparison between the thermal comfort provided by our system and the fixed-time one. We measured the absolute difference between the estimated final temperatures and the target one, 20ºC. Then, we made the monthly average of these values for each room and each considered month. Results show that our system provided better thermal comfort in all scenarios. On average, when using our system, the house's indoor temperature differed from the target by 0.38 degrees Celsius at 18 o'clock. In turn, by using the fixed-time system, this average difference equals 2.56. Thus, we affirm that our approach delivers a thermal comfort approximately 85% lower than the one used for comparison.

Fig. 4.11b compares both systems regarding the operation duration of the heater. The duration the graphs show is the monthly average per room and considers, for each day, only the period from the moment the heater started working to 18 o'clock. Since the action times of the *fixed-time* system originate from the averages of the ones of our system, we can see that our gain or loss of energy is strictly related to the rooms. Since the smallest is the one that heats faster, its heater usually turns on later. Thus, our energy consumption is less than the one provided by a system that turns on the monthly average time between the three rooms. However, the situation is the opposite when considering the small room with windows. Since it needs an action time earlier than this average, its heater operates longer.



(a) Comparisons regarding thermal comfort



(b) Comparisons regarding heater's operation duration

Figure 4.11 – Second use case results

5 CONCLUSION

This work showed that building a Digital Twin for a home can contribute to it with analysis and automation. We first proposed an architecture for the system, which comprises a physical house, a digital replica, and their connection. We then implemented two use cases that share architectural components and technologies. Both rely on the MQTT protocol to collect data from and return data to the physical part. Additionally, we used Kafka Streams for data storage and kSQL for data manipulation in the two use cases. Although their similar structure, they have different purposes, contemplating the two core roles of the Digital Twins.

The first use case focused on the interrogative role of a Digital Twin. It allows the users to inspect the current state of home appliances regarding their power consumption. Moreover, it lets them analyze the average of the same property during a custom period. We presented both past and present data with 3D replicas of the pieces of equipment colored by a color-coded gradient. The higher their power usage, the more intense their color was.

The second use case focused on the predictive role of a Digital Twin. Based on a house's heating simulations, it could determine the best time to turn on its power sources in different rooms so that their indoor temperature equals a target value when the user arrives home at the end of the day. We compared our approach to one that turns all the heaters on every day simultaneously and could improve the houses thermal comfort by 85%.

This project has some limitations that we plan to improve in future work. The main one is the absence of an actual physical system for our use cases. We simulated both power usage and temperature readings from data collected previously from different environments. Implementing our project in a real house with thermometers, smart power outlets, and power sensors is a considerable step our project can take. Moreover, an scalability test was not presented in this work and is a promising future work possibility. It could analyze the feasibility of expanding our systems to larger residences with more appliances and rooms or numerous homes, for example, a building with many apartments. Also, it could list and implement the necessary changes in our implementations to allow this expansion.

Another limitation was the heating simulation in the second use case, which we did in the assistant software Energy 2D. Due to its two-dimensionality, we could not know

the exact volume of the rooms we designed. Moreover, the hypothetical three spaces are too simple. As they were empty rooms, they did not consider interactions with the humans and the impact of a home's furniture and appliances on the temperature change. Another negative point of our simulation is that we examined the readings of only one thermometer per room, not the average temperature of its whole space, which is different at different places in the same area. A significant improvement of the system is analyzing its integration with other applications. We may be able to provide more realism to our system by using another heating simulation software, for example, EnergyPlus[1], developed by the U.S. Department of Energy (DOE). It simulates a building's energy consumption considering the characteristics of both the building and the climate in its region. The software can, among other features, calculate thermal comfort and convection processes (CRAWLEY et al., 2001). We believe this service, among others we may find, can provide a higher-fidelity simulation of a house's heating.

In conclusion, this study presents a proof of concept for a home's Digital Twin. We proved this with two use cases, contemplating two different purposes, and will soon available their source code on GitHub for possible further expansions.

[1] <https://energyplus.net/Visitedon28.Mar.2022>

REFERENCES

ASHTON, K. et al. That ‘internet of things’ thing. *RFID journal*, v. 22, n. 7, p. 97–114, 2009.

BAIRAMPALLI, S. N. et al. Digital moka: Small-scale condition monitoring in process engineering. *IEEE Sensors Letters*, v. 5, n. 3, p. 1–4, 2021.

BROMLEY, K.; PERRY, M.; WEBB, G. *Trends in smart home systems, connectivity and services*. 2003. Disponível em: <https://www.nextwave.org.uk/>.

CRAWLEY, D. B. et al. Energyplus: creating a new-generation building energy simulation program. *Energy and Buildings*, v. 33, n. 4, p. 319–331, 2001. ISSN 0378-7788. Special Issue: BUILDING SIMULATION’99. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0378778800001146>.

DAMJANOVIC-BEHRENDT, V.; BEHRENDT, W. An open source approach to the design and implementation of digital twins for smart manufacturing. *International Journal of Computer Integrated Manufacturing*, v. 32, p. 1–19, 04 2019.

FARAHANI, B.; FIROUZI, F.; CHAKRABARTY, K. Healthcare iot. In: ____. *Intelligent Internet of Things: From Device to Fog and Cloud*. Cham: Springer International Publishing, 2020. p. 515–545. ISBN 978-3-030-30367-9. Disponível em: <https://doi.org/10.1007/978-3-030-30367-9_11>.

FLEISCH, E. What is the internet of things? an economic perspective. *http://www.alexandria.unisg.ch/Publikationen/68983*, Auto-ID White Paper, 01 2007.

GATES, B. *How to avoid a climate disaster*. [S.l.]: Random House Large Print, 2021. ISBN 9780385546133.

GILL, K. et al. A zigbee-based home automation system. *IEEE Transactions on Consumer Electronics*, v. 55, n. 2, p. 422–430, 2009.

GLAESSGEN, E.; STARGEL, D. The digital twin paradigm for future nasa and u.s. air force vehicles. In: . [S.l.: s.n.], 2012. ISBN 978-1-60086-937-2.

GRIEVES, M. *Origins of the Digital Twin Concept*. [S.l.], 2016.

GRIEVES, M.; VICKERS, J. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In: ____. [S.l.: s.n.], 2017. p. 85–113. ISBN 978-3-319-38754-3.

GRIFFOR, E. et al. *Framework for Cyber-Physical Systems: Volume 1, Overview*. [S.l.]: Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2017.

HARPER, R. Inside the smart home: Ideas, possibilities and methods. In: ____. *Inside the Smart Home*. London: Springer London, 2003. p. 1–13. ISBN 978-1-85233-854-1. Disponível em: <https://doi.org/10.1007/1-85233-854-7_1>.

KAEWUNRUEN, S.; RUNGSKUNROCH, P.; WELSH, J. A digital-twin evaluation of net zero energy building for existing buildings. *Sustainability*, v. 11, n. 1, 2019. ISSN 2071-1050. Disponível em: <https://www.mdpi.com/2071-1050/11/1/159>.

KEAT, L.; CHUAH, C. W. Smart indoor home surveillance monitoring system using raspberry pi. *JOIV : International Journal on Informatics Visualization*, v. 2, 09 2018.

KHAJAVI, S. H. et al. Digital twin: Vision, benefits, boundaries, and creation for buildings. *IEEE Access*, v. 7, p. 147406–147419, 2019.

LIU, Y. et al. A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access*, v. 7, p. 49088–49101, 2019.

LIU, Z.; ZHANG, A.; WANG, W. A framework for an indoor safety management system based on digital twin. *Sensors*, v. 20, n. 20, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/20/5771>.

MIORI, V.; RUSSO, D. Domotic evolution towards the iot. In: *2014 28th International Conference on Advanced Information Networking and Applications Workshops*. [S.l.: s.n.], 2014. p. 809–814.

MOTLAGH, N. H. et al. An iot-based automation system for older homes: a use case for lighting system. In: *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*. [S.l.: s.n.], 2018. p. 1–6.

NARKHEDE, N.; SHAPIRA, G.; PALINO, T. *Kafka: The Definitive Guide*. [S.l.]: O'Reilly Media, Inc., 2017. ISBN 9781491936160.

NEGRI, E.; FUMAGALLI, L.; MACCHI, M. A review of the roles of digital twin in cps-based production systems. *Procedia Manufacturing*, v. 11, p. 939–948, 2017. ISSN 2351-9789. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2351978917304067>.

SONI, D.; MAKWANA, A. A survey on mqtt: a protocol of internet of things (iot). In: *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*. [S.l.: s.n.], 2017. v. 20, p. 173–177.

ZHENG, Y.; YANG, S.; CHENG, H. An application framework of digital twin and its case study. *Journal of Ambient Intelligence and Humanized Computing*, v. 10, p. 1868–5145, 2019.