

Requirements Engineering for Continuous Queries on IoRT Data: A Case Study in Agricultural Autonomous Robots Monitoring

Leandro Antonelli¹^a, Hassan Badir²^b, Houssam Bazza²^c, Sandro Bimonte³^d
and Stefano Rizzi⁴^e

¹Lifia, Fac. de Informática, UNLP, Buenos Aires, Argentina

²Abdelmalek Essaadi University, Tangier, Morocco

³TSCF - INRAE, University Clermont, Aubiere, France

⁴DISI, University of Bologna, Bologna, Italy

fi

Keywords: Requirements Engineering, Continuous Queries, IoRT, Prototyping, Agriculture.


Abstract: The Internet of Robotic Things (IoRT) is an extension of the Internet of Things, where intelligent mobile devices acquire sensor data and physically act in the environment. IoRT devices produce huge data streams, typically analyzed using continuous queries. We propose an approach to engineer requirements about continuous queries over IoRT data. Our proposal is specifically devised for end-users not skilled in IT and relies, for requirements elicitation, on spreadsheet-like templates called *stream tables*. Requirements analysis uses a novel UML profile, while requirements specification and validation rely on a fast prototyping tool so as to allow end-users to define continuous queries by themselves and validate them via web-based prototyping. Non-functional requirements are taken into account as well, in the form of available technological resources and data sources, and used for requirements validation. The results of some preliminary tests made with some real users suggest that stream tables are a valuable instrument for the engineering of continuous queries, and that fast prototyping is an effective support to the specification and validation steps.


1 INTRODUCTION


Recently, the advent of sophisticated sensors and robots, together with the development of communication technologies, enabled intelligent robots to connect to the Internet, so that they can efficiently exchange data among themselves and with the cloud. The Internet of Robotic Things (IoRT) is an extension of the Internet of Things, where robots monitor events and merge sensor data to determine a better course of action, then take real actions in the physical world (Simoens et al., 2018). Robots usually embed sensors to collect data from the environment, but they also produce data that describe their mechatronic behaviour (position, odometry, etc.). Differently from IoT devices, which typically have limited


computation resources, robots can produce very large data streams since they embed computers powered by the robot battery. Moreover, the embedding of significant computation and storage resources enables the use of more complex technologies and software to manage the produced data stream. Therefore, from a non-functional point of view, IoRT can have more demanding requirements than IoT, which strongly impacts the hardware and software architecture adopted for stream data processing.

IoRT applications usually come with real-time analysis tools based on complex data streaming computations. In particular, they often rely on *continuous queries* that constantly process streaming data to produce a stream of results. A continuous query is characterized by a *frequency* and a *temporal window*, the latter defining the set of streaming data over which the query is computed. An example is: “every 2 seconds (frequency) compute the average speed over the last 2 seconds (temporal window)”. Different kinds of continuous queries have been defined in the literature,

^a <https://orcid.org/0000-0003-1388-0337>

^b <https://orcid.org/0000-0002-6754-7807>

^c <https://orcid.org/0009-0009-4853-1699>

^d <https://orcid.org/0000-0003-1727-6954>

^e <https://orcid.org/0000-0002-4617-217X>

mainly depending on the type of window used (Golab and Özsu, 2003): *fixed* (where the size of the window is the same of the frequency), *sliding* (where the frequency is different from the window size), and *landmark* (where the window is periodically reset). Continuous queries are processed by streaming engines, such as Apache Flink, Esper, etc.; their formulation is based on ad-hoc query languages and requires advanced IT skills, also in consideration of the different possible window types.

Requirements have a critical role in the development of software of all types. This is also true for projects involving IoRT-based applications, where effective and error-proof engineering of functional and non-functional requirements is a key to success. Unfortunately, “conventional elicitation techniques are often time-consuming and not sufficiently scalable for processing such fast-growing data” (Lim et al., 2021), so new techniques must be investigated for the IoRT field. Due to the complexity of continuous queries, managing the related requirements can be particularly difficult. Additionally, IoRT application domains typically involve stakeholders with little or no Information Technology (IT) skills (Bimonte et al., 2021b), which further hinders the effectiveness of requirement engineering. End-users not skilled in IT may ignore the existence of specific sensors and robots; thus, they have no idea of the data made available and of the analyses that IoRT can offer, and it is difficult for them to suggest using some sensed data when they describe their analysis needs. Finally, advanced sensors and robots come with technological constraints (e.g., the type of communication network, the network latency, and the computation overload) that may undermine the feasibility of IoRT applications, and end-users are often unaware of them.

Although IoRT systems have reached a good level of maturity, only a few efforts have been made towards methods to engineer the related requirements (see (Aguilar-Calderón et al., 2022) for a survey); while some of these methods offer good coverage of the different steps of requirements engineering, none of them offers specific techniques to cope with continuous queries—especially in presence of end-users not skilled in IT. To overcome this limitation, in this paper we propose an end-to-end approach in which (non-IT) end-users and IT users cooperate to engineer requirements about continuous queries over IoRT data. Our approach can be framed within any methodology featuring the classical four steps of requirements engineering, namely, elicitation, analysis, specification, and validation (e.g., the one proposed by Kaleem et al. (2020)). Our main contributions for the four steps are as follows:

- We propose the use of spreadsheet-like templates to support requirements elicitation of continuous queries by non-IT end-users.
- We introduce an UML profile to support requirements analysis of continuous queries by IT users via class diagrams.
- We describe a prototyping tool to support requirements specification and validation of continuous queries.
- We consider non-functional requirements as well, in the form of available technological resources and data sources, and use them for requirements validation (Pohl and Rupp, 2015).

The rest of the paper is organized as follows. Section 2 discusses the related works, while Section 3 describes a case study concerning the monitoring of agricultural autonomous robots. In Section 4 we overview the methodological framework and explain our approach in detail, while in Section 5 we present an experiment made to assess its validity. Section 6 draws the conclusions and outlines our future work.

2 RELATED WORK

The seminal paper by Golab and Özsu (2003) surveys the different types of continuous queries in terms of data models, semantics, and query languages. In particular, the basic operators on streams (group by, aggregate, etc.), the supported windows (fixed, landmark, and sliding), and their execution frequency (streaming or periodic) are discussed. Later on, other works investigated continuous queries from different points of view (e.g., (Sumalatha and Ananthi, 2019; Gomes et al., 2019)). Visual languages allow users to define complex queries on data without any knowledge of the query language syntax, and therefore they could be considered as queries elicitation tools. Although some visual languages were proposed in this direction (e.g., (Silva et al., 2023)), to the best of our knowledge no work proposes an approach to generate a continuous query from a simple representation created by a user with poor IT skills.

Several proposals deal with requirements engineering in the IoT or IoRT domains. An experience of Use Case modeling in the healthcare domain is described by Laplante et al. (2016), who emphasize the stakeholder identification phase using actors of Use Case diagrams. A similar approach using UML and SysUML modeling is proposed by Meacham and Phalp (2016). Mezghani et al. (2017) propose a model-driven development approach based on UML

and patterns; although they do not describe new activities or tools, they introduce the use of patterns and ontologies. Costa et al. (2017) propose a model-based approach relying on a modeling language named IoT-RML. An agent-based approach consisting of three steps: (i) analysis, (ii) design, and (iii) implementation is proposed by Fortino et al. (2018). Although the authors consider the complexity of the cyber-physical elements, they only focus on one phase of requirements engineering, namely, analysis.

Rafique et al. (2020) introduce a service-based framework to develop IoT applications. They do not include stakeholder identification in their proposal, neither they consider the importance of continuous queries in IoT scenarios. Silva et al. (2019) present an approach that deals with many of the stages of the requirements engineering process; however, they deal with stakeholders only in a shallow way and do not mention continuous queries. A five-step approach is described by Kaleem et al. (2020). They do not propose specific artifacts to deal with continuous queries, and do not single out specific stakeholder roles. Silva et al. (2021) propose a comprehensive approach to build requirements specification documents. Their proposal addresses the classic stages in the requirements engineering process, but continuous queries are not considered. Zambonelli (2016) propose a general approach for software engineering to implement IoT applications. The activities of their approach are: (i) actors analysis and identification, (ii) functionality and requirements, and (iii) infrastructural analysis. Their phases agree in general terms with our proposal; however, our proposal is specifically focused on requirements engineering for continuous queries.

Overall, the main differences between our approach with the previous ones are as follows. Firstly, it introduces continuous queries as first-class citizens in requirements engineering. Secondly, it distinguishes two profiles of stakeholders, namely, end-users and IT users, who bring very different contributions to the engineering process; this is very relevant since IoRT applications have an important technological component and IT users should contribute with their knowledge and experience in the requirements engineering process. Finally, it covers most of the phases in requirements engineering identified by Aguilar-Calderón et al. (2022).

3 CASE STUDY: MONITORING OF AGRICULTURAL AUTONOMOUS ROBOTS

An agricultural robot is an unmanned ground vehicle equipped with sensors and actuators and able to safely and autonomously perform one or several tasks on a farm field. Such a robot is composed of a locomotion part connected to a navigation system, plus an agricultural part with either mounted, semi-mounted, or towed tools.

In the context of *agro-ecology*, which aims at developing new cultural practices leading to an environment-friendly farming production, farmers and agriculture stakeholders need a supervising system allowing them to monitor the advancement of robots in fields, as well as the data sent from sensors (André et al., 2023). Thanks to this system, the physical presence of people in the field will no longer be necessary, so they will save time for other tasks. This supervising system must enable easy monitoring of all the data usually involved in robotized agricultural tasks: sensors data (e.g., air temperature, wind speed, etc.); odometry data (i.e., data from motion sensors to estimate the changes in the robot's position over time); alert data (related for instance to mechanical faults of robots or to delays in the planned task); spatial and alphanumeric data representing contextual information (such as the geometries of the plots, the name of the farmer, etc.) Among these, sensors, odometry, and alert data come in streams: they are produced in a continuous way, and they are also queried via continuous queries. For example, it is important to monitor the movement of the robot in the field (speed, GPS, etc.) and the meteorological conditions (temperature, humidity, etc.) to interrupt the robot task when either its delay becomes too long, or the meteorological conditions are not suited for that task.

4 REQUIREMENTS ENGINEERING FOR CONTINUOUS QUERIES

Figure 1 portrays our approach and how it can be framed within a standard requirements engineering methodology. On the left-hand side (in grey), the four steps typically encompassed by requirements engineering; for each step, the specific techniques we propose to cope with continuous queries and the stakeholders involved in each of them. Two different kinds of stakeholders have been identified:

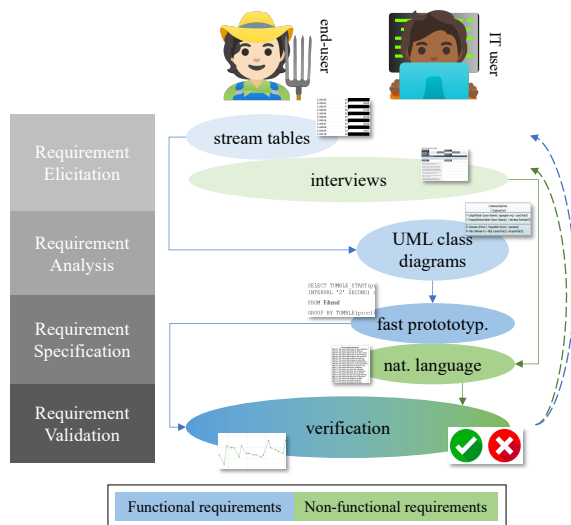


Figure 1: The techniques involved, at each step, to manage functional (in shades of blue) and non-functional (in shades of green) requirements for continuous queries on IoRT data.

- *End-users* (middle column of Figure 1), who are experts in the applications domain. They are normally not skilled in IT, and their digital literacy is limited to the basic usage of spreadsheet tools. We assume they can manipulate columns and rows and define basic formulas in a spreadsheet.
- *IT users* (leftmost column), who are experts in IoT and robots and in charge of the system implementation.

Notably, in Figure 1 we distinguish between techniques related to *functional* (in blue) and *non-functional* (in green) requirements. Below some additional explanations (see the work by Aguilar-Calderón et al. (2022) for a further description of the four steps):

1. **Requirements Elicitation:** aims at identifying the goals the system has to achieve.
 - *Functional requirements:* to specifically elicit the requirements related to continuous queries, we introduce *stream tables*, i.e., spreadsheet-like templates that end-users will fill to state what data they need and how they should be computed.
 - *Non-functional requirements:* End-users are interviewed by IT users and asked to describe the technological constraints entailed by the application domain (e.g., low communication bandwidth) and to survey the data sources available (e.g., robot GPS data sent every 10 seconds). Should end-users be unable to provide the needed information, IT users can actively help them.

2. **Requirements Analysis:** refers to creating models from the raw information obtained from end-user and make them consistent with each other (Laplante and Kassab, 2022).

- *Functional requirements:* To cope with continuous queries, IT users start from the stream tables created at the previous step to draw a UML class diagram relying on an ad-hoc profile.

3. **Requirements Specification:** produces a description of the system behavior.

- *Functional requirements:* IT users create a prototype that implements continuous queries, thus providing their formal specification via a query language.
- *Non-functional requirements:* a list of available resources is specified by IT users in natural language.

4. **Requirements Validation:** establishes whether or not the software requirements correctly capture the stakeholders' requirements.

- *Functional requirements:* end-users check the prototype to verify that the queries specified meet their analytical needs.
- *Non-functional requirements:* IT users verify that these queries are consistent with the technological resources and data sources available, i.e., that they actually can be implemented.

Should one or both of these checks fail, an iteration to the previous steps is necessary to refine the requirements.

The techniques involved in the four steps are described in detail in the following subsections.

4.1 Requirements Elicitation

At this step we propose to perform two distinct activities, namely *stream table design* and *identification of technological and data resources*, related to functional and non-functional requirements, respectively.

Stream table design allows end-users to tentatively express the continuous queries they need on their own, i.e., without relying on the support of IT users. The main idea, inspired by Bimonte et al. (2021a), is to let them use tools they are familiar with, such as spreadsheets. The template we have defined to this end, called *stream table*, can be used by end-users non skilled in IT to state (i) what data they need and (ii) how this data is computed. A stream table has two components: a declaration area, and a data area.

- The declaration area specifies (i) the temporal frequency for data collection (*Acquisition frequency*), (ii) the *IoRT identifier*, i.e., the sensor or

robot that sends the data, (iii) the data to be collected (*Source data*), and (iv) a formula applied to filter the source data (*Filtered data*). Moreover, the declaration area specifies how the data must be aggregated (*Group by*, e.g., by robot), and the computation to be made with its frequency and time window (*Continuous query*).

- In the data area, the end-user creates a small case of computation starting from an example provided in the template.

The stream table is filled by end-users in two stages:

1. First, they specify the *Acquisition frequency*, the *IoRT identifier*, the *Source data*, and the formula for the *Filtered data*.
2. Then, they specify the *Group by* and the *Continuous query* formula.

The identification of technological and data resources available in the application domain involves both end-users and IT users, and its outcome is necessary at the requirements validation step to confirm that the continuous queries specified are feasible. We propose to execute this activity via interviews focused on *sensing devices*, *computing resources*, and *communication network*. These interviews are conducted by IT users, and answered by end-users. When end-users cannot provide technical details about technology resources and data sources, IT users are in charge of retrieve them.

Example 1. *A farmer needs to monitor the behavior of an ADAP2E robot in the field to ensure that it respects the planned time scheduling and to check the air temperature. Specifically, she wants to issue two continuous queries:*

Que.1 Acquire the robot speed every second, filter erroneous data (marked with the '-1' value) and compute its maximum every 2 seconds on a window of 2 seconds (fixed-window query), grouped by farm name. The stream table created by the farmer is shown in Figure 2; black cells are used to mark the times when the query is not computed.

Que.2 Sense the temperature every second to compute its average over a window with maximum size 4 seconds (landmark-window query).

As to the identification of technological and data resources, examples of questions asked to end-users are: "What is the duration of the robots' batteries?", "What kind of devices do you want to use for visualizing data?", and "What is the speed of the data connection covering the fields?" □

4.2 Requirements Analysis

This step, performed by the IT user, aims to build a model of the continuous queries defined by end-users at the previous step; it has a crucial importance since in at this stage the information gathered during elicitation is organized and its consistency is checked. Then, the following specification will be produced according to this model.

To support this step we have defined an ad-hoc UML profile by extending the profile for IoT data proposed by Bimonte et al. (2023). The representation of continuous queries given in that work is simplified and does not enable the query details (e.g., the temporal window type and the group by attributes) to be expressed. Therefore, we extend that profile by providing a detailed conceptual representation of continuous queries based on either fixed, sliding, or landmark temporal windows. IT users can use our extended profile to translate smart tables into UML class diagrams. Importantly, these diagrams also show how queries are related to the data elements used by the application, thus providing a global picture of the project. This enables IT users to check that the continuous queries are consistent with the other data used by the application.

The UML profile we propose extends the one by Bimonte et al. (2023) with the following stereotyped attributes:

- «OutputAttribute» represents the values returned by the query and has two tagged values: Source (which class the value comes from) and Aggregation (the aggregation operator applied).
- «GroupingAttribute» represents the attribute(s) used to group the data. The grouping attribute(s) can be imported from other classes via the Source tagged value, then they can be used to establish the compatibility of the continuous query with the other data used by the application.
- «TemporalWindowAttribute» represents the timestamp used in the temporal window. It can be either explicit (the data source timestamp) or implicit (the timestamp of data reception).

A continuous query is also characterized by two operations:

- «Generation», which models the query computation frequency with the two tagged values Period and TemporalUnit.
- «Filter», which can model a selection condition over the input data.

We propose three stereotypes for the different kinds of temporal windows:

	A	B	C	D	E	F
1	external data or IoT identifier					
2	Group by	Acquisition frequency	Continuous query	IoT identifier	Source data	Filtered data
3	farm name	each second	each 2 seconds, MAX speed over last 2 acquisitions		robot speed in Km/h	robot speed >= 0
4	Montoldre	11:00:00		1	25	25
5		11:00:01	27,00	1	27	27
6		11:00:02			-1	
7		11:00:03	28,00		28	28
8	Aubiere			2	26	26
9		11:00:00		3	27	27
10				2	25	25
11		11:00:01	27,00		3	-1
12				2	20	20
13		11:00:02		3	15	15
14			2	14	14	
15		11:00:03	20,00	3	16	16
16						

Figure 2: Stream table based on a fixed window.

- `<<ContinuousQueryFixed>>` for fixed windows. In this case, the size of the window is the one of the Generation operation.
- `<<ContinuousQuerySliding>>` for sliding windows. Here, the tagged values `WindowSize` and `TemporalUnit` define the window size, while the query frequency is the one of the Generation operation.
- `<<ContinuousQueryLandmark>>` for landmark windows. The tagged values `WindowMax` and `TemporalUnit` model the maximum window size that can be reached before the window is reset.

4.3 Requirements Specification

To provide a formal specification for continuous queries, we have implemented a prototyping tool that simulates the real implementation. Apache Kafka (<https://kafka.apache.org/>) is used as a *message broker*; in fact, all IoT applications commonly use message brokers to manage the exchange of data between the devices and the fog/cloud server, and Kafka is today one of the most used systems in all kind of applications. Kafka relies on two main components: the *data producer*, which is responsible of data generation and sending, and the *data consumer*, which processes the data. In our tool, the input comes from a JSON file that includes historical data owned by end-users. The data producer is implemented in Java; it reads the JSON file and sends the data according to the frequency defined with the stream table. The consumer is implemented via the Apache Flink streaming engine (<https://flink.apache.org/>), a widely adopted system that implements the continuous queries over data received by Kafka. The results of these queries are then sent in input to Kafka. As previously described, continuous queries may also use additional, non-streaming data, therefore a DBMS (PostgreSQL in our implementation) is also present for their storage and retrieval. Finally, query results are visualized by means of Grafana (<https://grafana.com/>), which can

read and visualize data from Kafka in a simple and intuitive way so that the end-user can check, during the following step of requirements validation, that the query formulated actually matches her/his requirement. Note that, although some simulation solutions for sensors and robots (such as Matlab SimuLink) exist, they do not rely on the streaming engines and message broker systems normally used to handle big data streams. Therefore, an additional advantage of our prototyping architecture is that of proving that the continuous queries specified are actually feasible using commonly-adopted real stream technologies.

On the non-functional side, the available resources that emerged during the interviews made to IT users are specified using natural language. In our example, they are specified as follows:

- Res.1 An ADAP2E robot can autonomously move in the field, while sensing and sending data of different types. The robot speed is sensed and transmitted every second.
- Res.2 The sensors deployed in the field are battery-based and are able to acquire data and transmit them via Wi-Fi connection. Batteries are expected to be replaced approximately every month; however, their duration heavily depends on the frequency of data transmissions.

4.4 Requirements Validation

The first activity performed at each step is related to non-functional requirements and aims at letting end-users validate the continuous queries specified. Indeed, stream data is continuous, and the static visualization provided by stream tables does not really show end-users what query results will look like. In other words, stream tables alone cannot be considered as an effective tool for requirements validation. On the other hand, the visualization provided by Grafana as part of the prototyping process described in Section 4.3 is well-suited to this end, since it gives a dynamic

view based on historical data. As an example, the Grafana visualization of the results of query Que.1 is shown in Figure 3.

The second activity to be carried out aims at checking the feasibility of the continuous queries against the non-functional requirements specified. In our example, query Que.1 applied to the robot data turns out to be feasible, since the amount of data required can be easily processed either by a simple PC deployed at the farm or in the Cloud. The same is not true for Que.2. In fact, Que.2 is not feasible since temperature data is collected by battery sensors, and the acquisition and sending frequencies required are too high, which would imply that batteries are changed too often —while IT users suggested that batteries should be changed every month. Therefore, Que.2 must be redefined taking advantage of the sensor capability of computing simple functions such as the average. Thus, farmers create a new stream table where the acquired data is the 6-hours average temperature computed by the sensor. The new query, Que.2bis, can be stated as “every 6 hours, compute the maximum of the 6-hours average temperature with a maximum window size of 24 hours”. Sending data each 6 hours drastically reduces the number of sending operations of the sensor, thus significantly increasing the battery duration.

5 EXPERIMENT

In this section, we present an experiment we have conducted to evaluate the effectiveness of stream tables in the requirements engineering process. Ten robotics engineers with no previous knowledge of continuous queries have been enrolled. After a 10-minutes training about how to use stream tables, we have asked them to design the following sliding-window query: “Acquire temperature data every hour to compute, every 2 hours, the minimum temperature over the last 4 hours, grouped by plot”.

Most users made a single error: they did not use black cells to show that the computing frequency was half the acquisition frequency. However, after a brief explanation of the error, they all were able to correctly draw the stream table. Thus, all users completed the task in 2 iterations; the maximum time they employed was 5 minutes.

We have also asked our users to answer two questions:

- QST.1 Do you think that a stream table is an effective tool to represent your analytical needs?
 QST.2 How difficult is it for you to edit the stream table template to suit your analytical needs?

As to question QST.1, all users answered yes, giving a very positive feedback. As to question QST.2, all users judged the editing of stream tables very easy.

Finally, we have asked our users to evaluate the benefits of the Grafana implementation via the following questions:

- QG.1 Do you think the visualization with Grafana helps you better understand what the results of the continuous query will look like?
 QG.2 Do you think the visualization can help you realize that the implemented continuous query does not actually match your analytical needs?

All users answered positively to both QG.1 and QG.2. Among the feedback they provided, most users stated that having the results of their continuous queries on historical data quickly visualized significantly helps to keep them actively engaged in the project.

6 CONCLUSION

In this paper we have proposed an approach for requirements engineering of continuous queries in IoRT. To this end we have introduced (i) stream tables as a way to let non-IT end-users communicate their requirements about continuous queries, (ii) UML stereotypes to precisely model these requirements, and (iii) a prototyping tool to let users visualize the results of queries on historical data. Importantly, non-functional requirements in the form of available resources and data are taken into account at each step, and used to verify the technical feasibility of continuous queries.

The results of the preliminary tests made with some real users suggest that stream tables are a valuable instrument for the elicitation of continuous queries, and that the fast prototyping tool based on Flink and Grafana helps is an effective support to the specification and validation steps. Our future work on this topic will be mainly focused on investigating the use of goal modeling languages, such as i^* , to let non-functional requirements be analyzed more precisely and in close connection with functional requirements (Salcedo et al., 2021).

ACKNOWLEDGEMENT

This work was supported by the French National Research Agency project IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) and by the European Union Next - GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) - MISSIONE

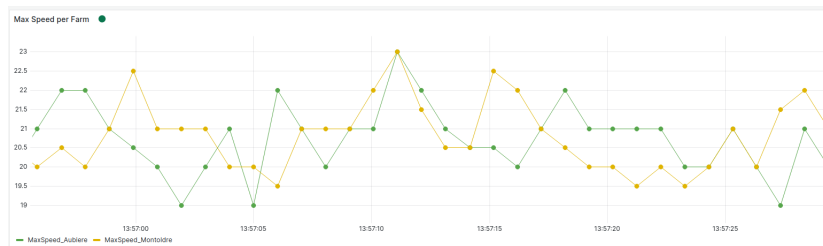


Figure 3: Visualizing the results of Que.1 with Grafana.

4 COMPONENTE 2, INVESTIMENTO 1.4 - D.D. 1032 17/06/2022, CN00000022). This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

REFERENCES

- Aguilar-Calderón, J.-A., Tripp-Barba, C., Zaldívar-Colado, A., and Aguilar-Calderón, P.-A. (2022). Requirements engineering for internet of things (IoT) software systems development: A systematic mapping study. *Applied Sciences*, 12(15):7582.
- André, G. et al. (2023). LambdaAgrIoT: a new architecture for agricultural autonomous robots' scheduling: from design to experiments. *Cluster Computing*, 26:2993–3015.
- Bimonte, S., Antonelli, L., and Rizzi, S. (2021a). Requirements-driven data warehouse design based on enhanced pivot tables. *Requir. Eng.*, 26(1):43–65.
- Bimonte, S. et al. (2021b). On designing and implementing agro-ecology IoT applications: Issues from applied research projects. In *Proc. EDOC*, pages 204–209, Gold Coast, Australia.
- Bimonte, S. et al. (2023). Data-centric UML profile for agroecology applications: Agricultural autonomous robots monitoring case study. *Comput. Sci. Inf. Syst.*, 20(1):459–489.
- Costa, B., Pires, P. F., and Delicato, F. C. (2017). Specifying functional requirements and QoS parameters for IoT systems. In *Proc. DASC/PiCom/DataCom/CyberSciTech*, pages 407–414, Orlando, USA.
- Fortino, G. et al. (2018). Agent-oriented cooperative smart objects: From IoT system design to implementation. *IEEE Trans. on Systems, Man, and Cybernetics: Systems*, 48(11):1939–1956.
- Golab, L. and Özsu, M. T. (2003). Issues in data stream management. *ACM Sigmod Record*, 32(2):5–14.
- Gomes, H. M., Read, J., Bifet, A., Barddal, J. P., and Gama, J. (2019). Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter*, 21(2):6–22.
- Kaleem, S., Ahmed, S., Ullah, F., Babar, M., Sheeraz, N., and Hadi, F. (2020). An improved RE framework for IoT-oriented smart applications using integrated approach. In *Proc. AECT*, pages 1–6.
- Laplante, N. L., Laplante, P. A., and Voas, J. M. (2016). Stakeholder identification and use case representation for internet-of-things applications in healthcare. *IEEE Systems Journal*, 12(2):1589–1597.
- Laplante, P. and Kassab, M. (2022). *Requirements Engineering for Software and Systems*. Auerbach Publications, New York, 2nd edition.
- Lim, S., Henriksson, A., and Zdravkovic, J. (2021). Data-driven requirements elicitation: A systematic literature review. *SN Computer Science*, 2:1–35.
- Meacham, S. and Phalp, K. (2016). Requirements engineering methods for an internet of things application: fall-detection for ambient assisted living. In *Proc. BCS SQM/Inspire Conference*, pages –.
- Mezghani, E., Exposito, E., and Drira, K. (2017). A model-driven methodology for the design of autonomic and cognitive IoT-based systems: Application to healthcare. *IEEE Trans. on Emerging Topics in Computational Intelligence*, 1(3):224–234.
- Pohl, K. and Rupp, C. (2015). *Requirements Engineering Fundamentals*. Rocky Nook, 2nd edition.
- Rafique, W., Zhao, X., Yu, S., Yaqoob, I., Imran, M., and Dou, W. (2020). An application development framework for internet-of-things service orchestration. *IEEE Internet of Things Jour.*, 7(5):4543–4556.
- Salcedo, J. A. et al. (2021). Modeling non-functional requirements of a reactive system. In *Proc. iStar Workshop*, pages 15–20, St. John's, Canada.
- Silva, D., Gonçalves, T. G., and da Rocha, A. R. C. (2019). A requirements engineering process for IoT systems. In *Proc. Brazilian Symp. on Software Quality*, pages 204–209, Fortaleza, Brazil.
- Silva, D., Gonçalves, T. G., and Travassos, G. H. (2021). A technology to support the building of requirements documents for IoT software systems. In *Proc. Brazilian Symp. on Software Quality*, page 4, São Luís, Brazil.
- Silva, E., Fidalgo, R., Ferro, M., and Franco, N. (2023). Visual query languages to design complex queries: a systematic literature review. *Software and Systems Modeling*, 22:1217–1249.
- Simoens, P., Dragone, M., and Saffiotti, A. (2018). The internet of robotic things: A review of concept, added value and applications. *International Journal of Advanced Robotic Systems*, 15(1).
- Sumalatha, M. and Ananthi, M. (2019). Efficient data retrieval using adaptive clustered indexing for continuous queries over streaming data. *Cluster Computing*, 22(Suppl 5):10503–10517.
- Zambonelli, F. (2016). Towards a general software engineering methodology for the internet of things. arXiv:1601.05569.