Outlier detection with ontology-driven fault contextualization in the Industry 4.0

Dionei Miodutzki[1], **Cesar Tacla**[1] **and Luiz Gomes-Jr**[1]

[1]Programa de Pós-Graduação em Computação Aplicada
Universidade Tecnólogica Federal do Paraná (UTFPR)
Curitiba, PR - Brasil

dionei@miodutzki.com, tacla@utfpr.edu.br, lcjunior@utfpr.edu.br

Abstract. *In the industrial sector, outlier detection makes it possible to quickly identify equipment failures. The evolution of Industry 4.0 is bringing challenges previously uncommon in the area. The large number of data constantly generated represents a processing challenge and can lead to the identification of a large number of outliers simultaneously. This scenario slows the troubleshooting process, delaying the identification of the source of the fault. This work presents a solution to support decision-making in a widespread failure scenario. Dependencies are represented using ontologies, to provide a clear and user-facilitated interpretation. An inference engine is used to identify the most probable causes of the failure. Performance tests demonstrate its scalability.*

1. Introduction

In general terms, an anomaly, also called an outlier, is discordant information, an exception [Chandola et al., 2009]. Outliers can create problems if not detected properly. Outlier detection techniques are used in several sectors of the economy: to detect failures in aeronautical sensors, detect credit card fraud and identify early signs of latent disease, among others. However, modern society imposes new challenges to the problem of detecting outliers. New technologies and devices are launched every day, and the industrial sector is at the forefront of this evolution. The advent of Industry 4.0 is equipping plants with sensors and automation devices to monitor and control production processes with high precision. All these sensors, systems and software generate a large amount of data that must be analyzed to have outliers properly detected. Such connectivity, however, ends up resulting in an also expressive number of detected outliers, up to the point of making it impractical for them to be analyzed manually.

Typical examples of widespread failures are natural events such as a lightning strike, where a single physical incident can prompt many outliers in several devices in a given geographic region. Another scenario, harder to interpret, is when an event causes the occurrence of cascading outliers. A failure causes an outlier observed by a sensor, and this failure is propagated to other devices that depends on the first. To make those outliers easier to interpret and use resources efficiently, it is crucial to identify these cases and present to the user a higher-level knowledge related to the origin of the failure.

The identification of the source of failure is a challenging task. Finding a flexible solution that allows representing the most diverse industrial setups and their relationships in a clear and objective way is difficult in itself. In our proposal, the relationships between the devices are described in an ontology, using the OWL language (Section 3). The

ontology simplifies knowledge management, especially in such complex scenarios of Industry 4.0. The ontology is stored in a graph database, which also receives and stores the detected outliers (Section 4). A graph query language is used to perform the inferences needed to determine the probable causes of the failure. The proposal was tested on large ontologies simulating complex industrial scenarios (Section 4.4). Test results demonstrate the scalability and feasibility of the proposal.

2. Fundamentals and Related Work

2.1. Sensors, IoT and Industry 4.0

Although there is no clear definition of Industry 4.0, it is characterized by a list of technologies and scenarios [Herrman et al., 2015]: Interoperability, with companies, systems and humans connected to each other; virtualization, where systems monitor physical processes through sensors and use this data in models and simulations; decentralization, as the large number of connected systems makes centralized control impractical; real-time capability, as data analysis and reaction must be immediate; service orientation, to allow easier use of all resources, they are encapsulated in large sets; and modularity, which allows quick adjustments, adaptations and scalability.

The concept of Industry 4.0 is deeply related to IoT. As Wang et al. [2022] have written, the Internet of Things (IoT) is an extended and expanded system network based on the Internet, built to achieve real-time interaction among things, machines and humans through various advanced technological means. IoT is bigger than Industry 4.0 and has applications in diverse fields, such as health, sports and home automation. The concepts and the technologies, either hardware, software and protocols used are complementary to those of Industry 4.0.

This increasing amount of sensors leads to an also increasing number of outliers generated in those new industrial environments. According to Gaddam et al. [2020], sensors used for IoT applications are often installed in harsh environments and are typically made of inexpensive electronic components, thus it is hard to guarantee a correct operation, free of malfunctions.

2.2. Outlier detection

Outlier detection seeks to identify patterns of behavior in the data and use this to classify which data does not belong to this pattern. As defined by [Hawkins, 1980], an outlier is an observation that deviates so much from others that it creates suspicion it was generated by a different mechanism. This type of technique is used in the most varied sectors of the industry: to detect failures in aeronautical sensors, detect credit card fraud, identify early signs of latent disease, etc.

In an IoT environment, data is constantly being generated, transmitted and stored. Thus, outlier detection techniques should be able to work with datasets that are not only big but also constantly receiving more data. Solutions to detect outliers in these architectures almost always involve the use of [Yu et al., 2014] data windows. In this approach, only a sample with a defined size is used for data analysis, and the window of samples used is constantly moved to accommodate newly received data.

Among the most easily implemented techniques to detect outliers are the distance-based algorithms [Orair et al., 2010]. On them, a radius is defined in which other data will

be considered as part of the neighborhood of the analyzed sample and data that do not have a minimum number of elements in their neighborhood are considered outliers. Despite being simple, there are quite sophisticated implementations of this type of algorithm that allow good precision and efficiency.

A different approach to obtain more details about an outlier is using outlying aspect mining (OAM). OAM complements traditional outlier detection because by identifying characteristics that make the observation different from other objects [Duan et al., 2015]. This technique is useful in understanding why a single data was detected as an outlier, it is not designed to detect the source of a group of outliers.

In this work, outlier detection is used as the first step of our architecture, identifying anomalies in the data generated by each device individually. This may lead to a large number of outliers generated as a consequence of the same real-world event. The outlier detection is run in a stream processing platform to allow a better performance with large volumes of data, in a similar way as described in [Toliopoulos et al., 2020]. The inference over the detected outliers is our main contribution.

2.3. Ontologies and OWL

Knowledge representation is a challenge in Industry 4.0. In the past, it was usual to represent entities, their properties and relationships directly on the code, using object-oriented programming. In Industry 4.0, however, this is not a viable method. New types of devices are added to the network constantly, and their relationship changes accordingly to what is needed in the production line. To deal with the problem of heterogeneity, ontologies emerge and play an important role to integrate seamlessly business processes with technical processes [Cheng et al., 2016].

According to Guarino [1998], in AI, an ontology usually refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. In this scenario, an ontology is a document that describes entities that are part of a system, listing their properties and how they are related to each other. In an analogy with object-oriented programming, the ontology would be the classes, with their hierarchies and properties.

The OWL is the Web Ontology Language, an ontology language for the Semantic Web with formally defined meanings [Group, 2012]. An OWL file contains an ontology with classes, properties, individuals and data values, allowing entities to share the entire ontology easily between different software and platforms.

In this work, an ontology was used to describe the dependencies (functional and geographic) of industrial equipment in large industrial settings, following Industry 4.0 practices. This makes the solution generic enough to be easily adapted to different industrial plants, or even to be used on other environments than industrial ones.

2.4. Related Work

Ontologies have been used in outlier detection scenarios. However, the focus is usually on detecting anomalies in the data relationships themselves. In other words, finding data that are related in ways they were not supposed to. A typical use for ontologies on outlier detection is to detect fraud in banking systems. Ramaki et al. [2012] used an ontology to

describe how each piece of information in a credit operation relates to each other, making it easier to create a model of an expected behavior. This model is then used to detect transactions that do not follow that expected behavior.

Fleischhacker et al. [2014] shows that ontologies can also be used to detect outliers in numeric datasets. Numeric values present in Wikipedia articles are extracted from the texts (countries' populations for example). An ontology describes how those values are related to each other (what is being counted, country or other geographic entity relevant, the year the data was collected, etc). Then, all numeric values present in the texts are compared with the inferred ontology, and those that are dissonant are quickly identified for verification.

In our work, ontologies are not the subject of outlier detection.. Instead, they are used to identify, among outliers that were previously detected with traditional methods, the most probable source of the real-world event.

3. Proposed Architecture

The proposed architecture consists of two main processing steps: (i) the outlier detection and (ii) the source inference. As shown in Figure 1, data are collected by telemetry devices and are sent to a server that performs traditional outlier detection, independently, in each of the sensors that feed the system. In this step, there is a detection model to analyze each sensor individually, detecting outliers in its regular operation. Given the high volume of data and processing demands, our solution uses a distributed stream processing architecture. However, the distributed outlier detection is out of the scope of this paper.

Once the outlier detection is finished, data and the detected outliers are moved to the second step, where data from all sensors are aggregated for each time window. Therefore, for a given time window (configurable), a set of detected outliers represents the current state of the facility. The detected outliers are then associated with ontologies representing functional and spatial dependencies among devices. This approach makes both steps completely independent, allowing any outlier detection implementation to work with the source inference. The inference engine determines the probable cause of the failure based on the ontologies and the outliers. Finally, the outliers identified as the sources of the anomalies are presented to the user in a report.

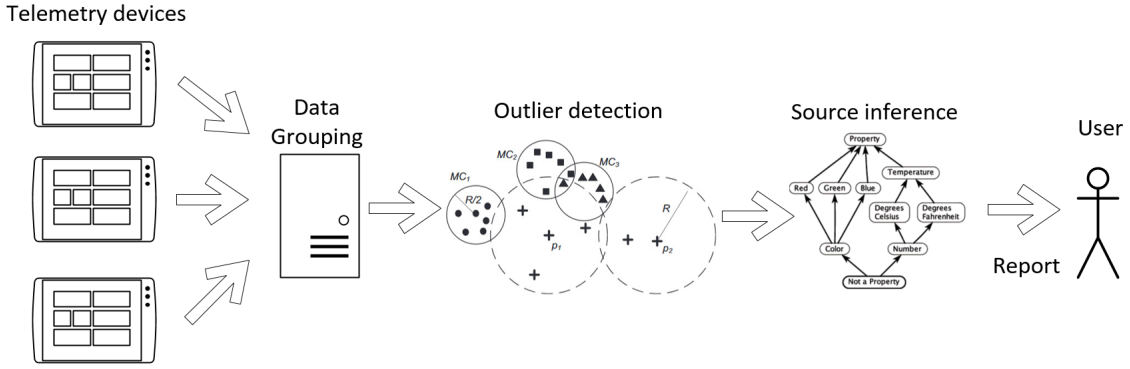

Figure 1. Proposed architecture for outlier detection and fault contextualization.

The main goal of using ontologies in this architecture is to streamline knowledge

representation, making it easier to maintain and modify the dependencies among devices. In addition, this allows the use of relationship names directly in queries, making them much more descriptive and easier to understand. Finally, as the relationships are only described in the ontology, the solution becomes more flexible and easily adaptable to other scenarios, without the need to modify the source code of the application.

3.1. Case Study

The case used to test and explain the system is based on a real scenario for water resources management in an industrial plant. The data are related to the production of water in deep wells, pipelines, reservoirs, distribution systems and various industrial equipment that consume and manage the water. Those facilities are operated by different companies in various Brazilian cities.

Each location of interest has devices that collect data continuously with a fixed sampling rate. The data are sent on a recurring basis to a server to be processed and stored. Each installation can have one or more variables, according to the device being monitored. Table 1 shows an example of data collected from two of those devices, with a column *Device* identifying the source. *Level* is the well water depth, *Flow* is how much water is being pumped out of it, *Volume* is how much water was pumped so far, and *Time* is how many minutes the water was pumped. The full scenario includes several types of devices with multiple monitored variables.

Table 1. Data sample

Timestamp	Device	Level (m)	Flow (m^3/h)	Volume (m^3)	Time (min)
2021-06-01 10:00	Well 1	250,6	23,1	25685,4	100
2021-06-01 10:00	Well 2	123,4	0	57483,2	5
2021-06-01 10:01	Well 1	252,1	23,5	25687,8	101
2021-06-01 10:01	Well 2	124,4	0	57483,2	5
2021-06-01 10:02	Well 1	252,4	23,7	25692,3	102
2021-06-01 10:03	Well 2	124,3	0	57483,2	5

Among the possible scenarios where multiple outliers caused by the same failure can occur in this environment, the most directly related to this particular industrial architecture is the propagation of anomalies through the pipelines. If there is a leak in the piping right before the location where a device is measuring the flow of the water, this device and possibly all the others that are supplied by it will generate outliers.

On the other hand, external factors, such as a power failure, for example, may render an entire industrial plant offline, causing all its equipment to generate outliers at the same time. The goal of our proposed system is to automatically identify, in a knowledge representation sense, the probable cause in such multi-outlier scenarios.

4. Implementation and tests

4.1. Data collection and processing

The devices collect data in regular intervals and transmit those data packages to the server in the order they were generated (first in, first out). Each package contains a full set of all

variables monitored by that device, and all devices collect data on the same timestamps. Data are then transmitted to the MQTT broker (*Message Queuing Telemetry Transport*, a very popular message protocol for telemetry and sensors communication) by using a mobile internet connection. Figure 2 shows an example of this data flow, in the order that it happens. In the figure, the first block (top left) represents the readings from a set of sensors (volume, flow and time) from a device at timestamp 2021-08-01 00:04:00).

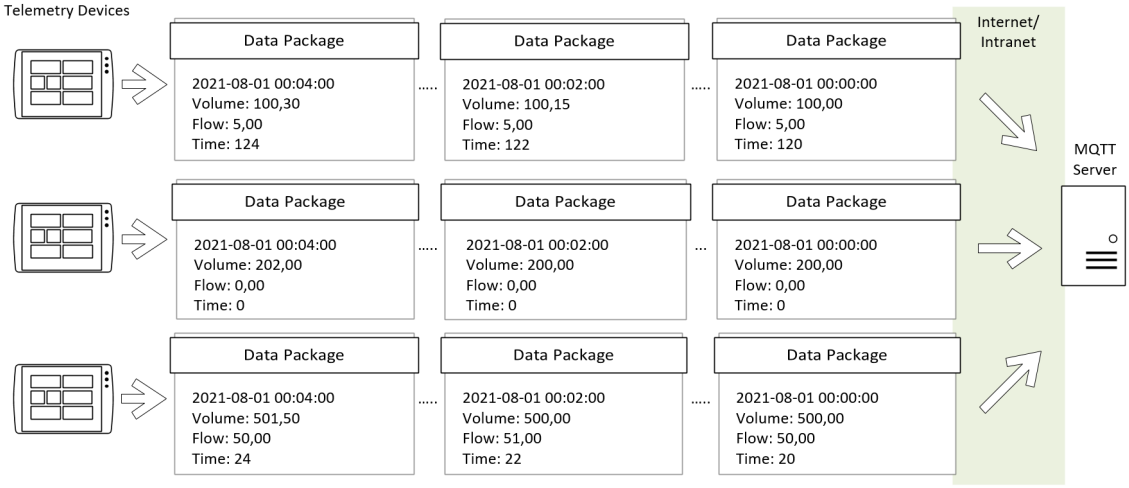

Figure 2. Data flow from the telemetry devices.

Once the data are received, the server waits for all devices to transmit their packages for each timestamp, group them and then send them to an outlier detection step where detected anomalies are combined with the original data. Figure 3 shows how the data is saved after the outliers are detected.

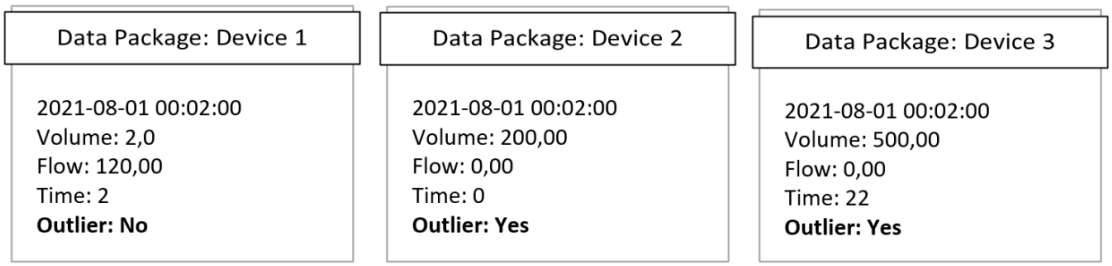

Figure 3. Data packages after the outlier detection.

4.2. Knowledge representation

An important aspect taken into account during the project was knowledge representation. The goal is to centralize the knowledge about the dependencies between devices, where they are and how they relate to each other.

This ontology contains two types of relationships. The first represents the geographic organization of the devices (in which company, sector, city and state they are contained), as is shown in Figure 4. The ontology contains information about where the

sensor is geographically located, grouping sensors at different levels of hierarchy, thus allowing generalized faults located in a specific region (power supply failure for example) to be classified as unique failures.

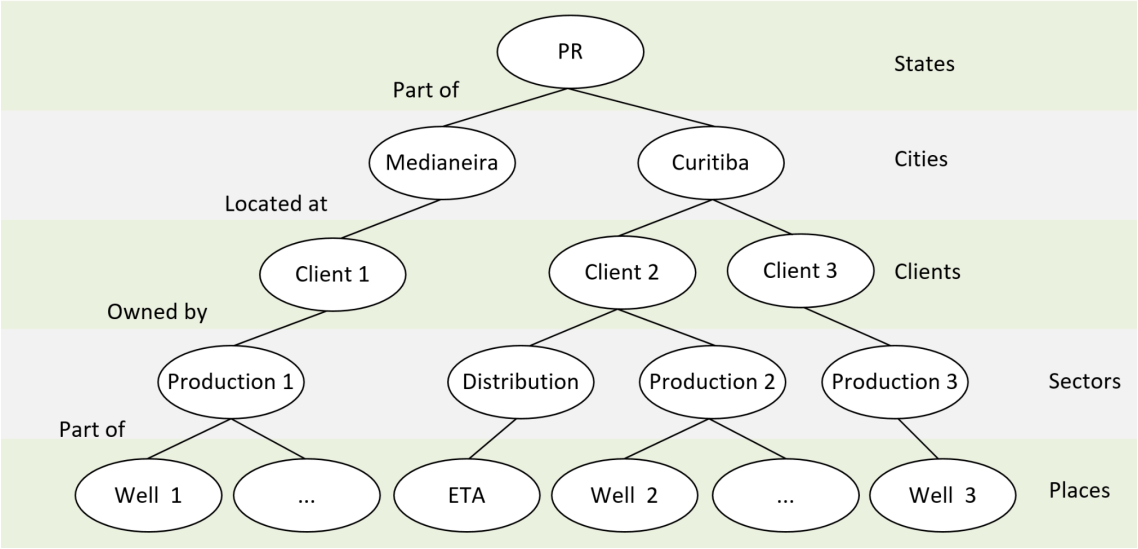

Figure 4. An ontology representing geographic distribution of devices.

The devices are distributed at various geographic levels. In the study case, each device is part of a sector, which is owned by a client. The client is a single industrial unit, located in a city, that is part of a state.

The ontology also contains a second type of relationship, that represents the dependency between the devices, that is, a malfunction in one device affects others down the line. Figure 5 shows an example of this dependency, where water is collected on *Well 01*, sent through a pipe on *Distribution 01*, then stored on *Tank 01*. From there, it is sent to either *Cleaner 01*, or to be stored on *Tank 01b* using *Pump 01*, from where it can be sent to *Fillers 01*. One failure on *Tank 01*, for example, may also cause issues on *Pump 01* and *Cleaner 01*, and as a consequence all devices that are supplied by them.

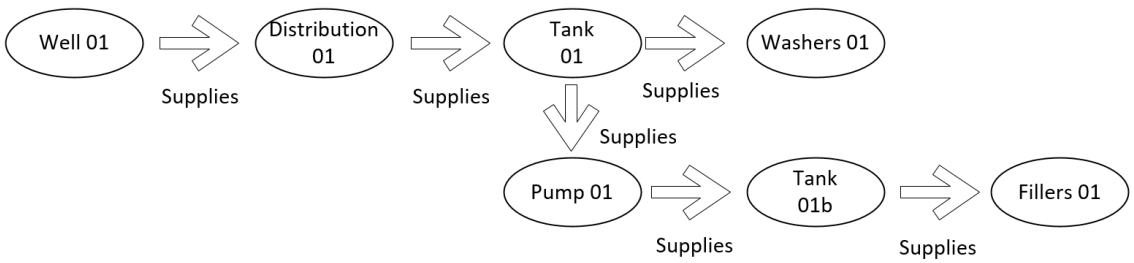

Figure 5. An ontology representing dependency between devices.

All this information, including the definition of what a sensor is, what information it contains, how it can relate to other sensors, and how it can be contained within a geographic region, is saved on an OWL file. The same file also contains details about the

spatial relationship between those elements. Finally, the file also contains all the elements instances of the project - the sensors and spatial entities.

4.3. Inference

A central step of our implementation, the inference is the moment in which the outlier that most probably originated the sequence of outliers is identified. Once the ontology is built with all devices, data, and its respective outliers, the information is loaded into a database to make inference easier and faster.

The solution was implemented using the graph database Neo4j with the Neosemantics ontology library. The detected outliers are mapped to the ontology in each analyzed timestamp, then two queries in Cypher language are used to detect the regional and dependency outliers sources, as discussed next.

4.3.1. Geographic Inference

In this implementation, a region is considered an outlier if all devices it contains are outliers in a given time window. This rule is applied recursively, so when all devices of all sets of a client are outliers, the client is considered an outlier, and so on. This permits to quickly identify anomalies from external agents, like issues with the power supply network, for example. Figure 6 shows this detection being activated.

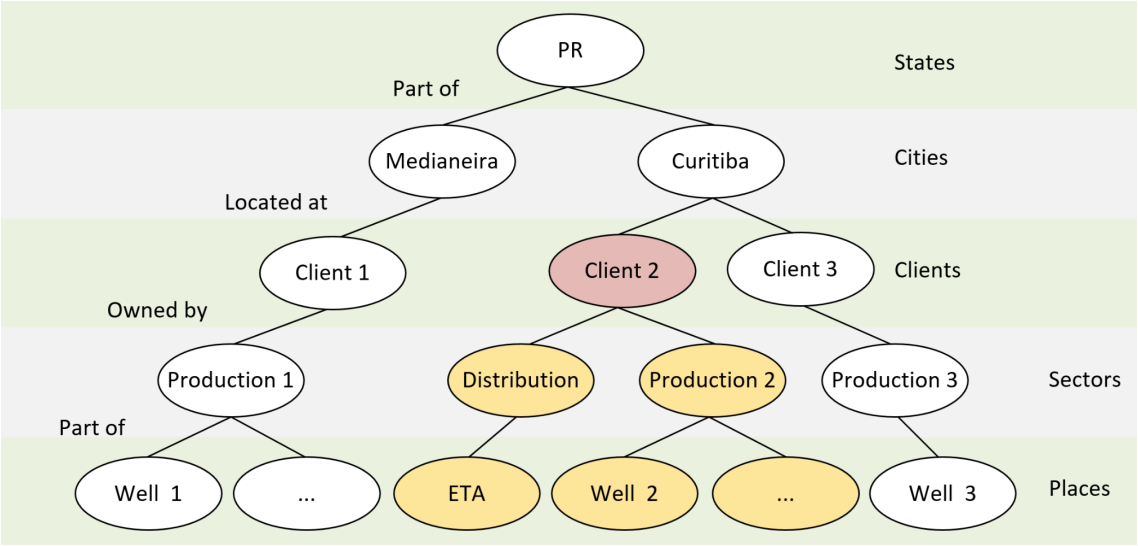

Figure 6. Example of data failure in an entire client.

Listing 1 shows the query that does the inference on regional dependency. First, it matches all region entities, then it selects only those that contain devices and that all devices have outliers.

Listing 1. Geographic inference query

```
match (n:region)
    where (n)-[:contains]->(:device {outlier: true} )
    and not (n)-[:contains]->(:device {outlier: false})
```

```
    return n
```

4.3.2. Dependency Inference

In our use case, the dependency between the devices is given according to the flow of water. The points of origin are the wells, where the water is captured, and then each device supplies one or more other devices, until reaching the final point of the distribution network. In the opposite direction, the device is supplied by each other.

In this scenario, a physical failure in *Distribution 01*, for example, can cause a variation in the data, and consequently new outliers, in devices that are supplied by it. Figure 7 illustrates this example, where the device in red is where the actual failure occurred, but the device in yellow also showed inconsistent data as a result.

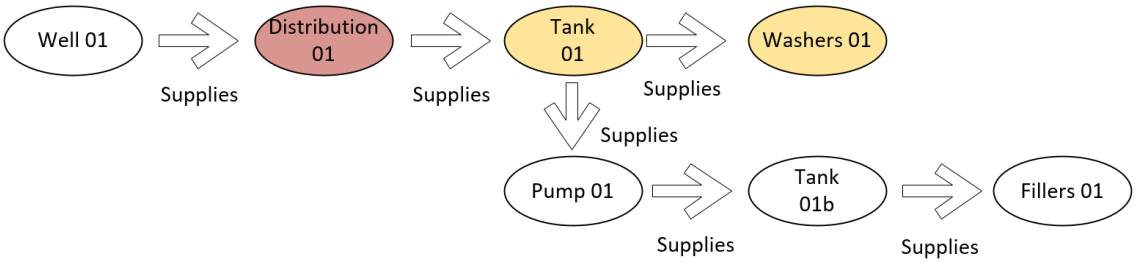

Figure 7. Example of data failure transmitted from one device to another.

Listing 2 shows the query that does the inference of the source of outliers based on the dependency between devices. First, it matches all device entities that are outliers, then it filters out the ones that are supplied by other devices that are outliers too. Therefore, only the devices that are the firsts in a sequence of outliers are listed.

Listing 2. Dependency inference query

```
match (n:device {outlier: true})
    where not (n)-[:supplied]->(:device {outlier: true})
    return n
```

4.4. Tests

Extensive tests were done to simulate the system operating on industrial plants of various sizes and setups. Testing was done in two steps: The first, qualitative, uses a handcrafted ontology,representing a series of real-world fault scenarios. The sample contains 18 devices, divided into two cities and three different clients, and simulates six fail events where the first outlier is known. This ontology and fail cases were validated by a domain expert. This ontology was used during development to assure the queries were detecting the expected location in each expected scenario.

The second test scenario was designed for performance evaluation. A Python script was developed that generates random ontologies using the same device and geographic entities, with different numbers of entities and outliers. The script loads the ontologies into the graph database and runs the queries, keeping track of the time it took

to process. The script run tests using ontologies with 5000 to 100000 entities, increasing in steps of 5000 entities, and varying the number of outliers between 10% and 70% of the entities. Each test was run five times, all of them as cold-runs (the envionment was restarted between each run), and the median value of all runs were used as a reference.

For this test, an Ubuntu Linux 21.04 virtual machine was used, running in a PC Intel I5-3470s with four cores at 2.90Ghz, 1TB SSD disk and 8GB of RAM. The computer was running the hypervisor operational system ESXI, and the virtual machine received four virtual cores with 1Ghz each, and 4GB of RAM. To avoid any interference between the tests, no other virtual machine was running on the server during the tests. Between each data sample size change, the virtual machine state was reverted to a snapshot taken at the beginning of the tests, thus assuring the results are not affected by any type of cache in the database.

4.5. Results

The qualitative test, using the manually written ontology, suggested that the solution could correctly classify the expected location on all simulated fail events. For both dependency and geographic inference, all simulations resulted in the expected most probable source identified.

The results of the performance tests were divided into two, showing the variation in the time for processing the ontologies when the number of entities is increased, and when the number of synthetic outliers is also increased.

4.5.1. Scalability

The graph in Figure 8 shows the median result of how much time (in milliseconds) it took to run the regional and dependency inferences. There is a visible but linear increase in the processing time of the regional inference while the size of the ontology is also increased. The time needed to process the dependency inference also increases linearly but at a slower rate. Tendency lines were added for an easier interpretation and suggest consistent linear growth, conclusive to good scalability.

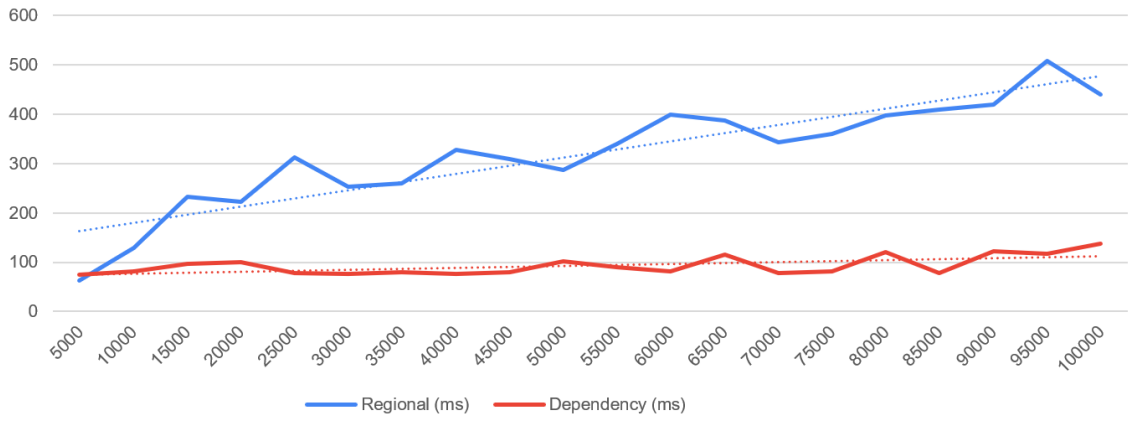

Figure 8. Time variation while ontology size is increased.

4.5.2. Increasing ontology size

Table 2. Results increasing ontology size

Outliers	Regional (ms)	Dependency (ms)
10%	297.9806662	80.47699928
30%	312.7410412	81.6538935
50%	365.3517962	77.27694511
70%	347.4471569	76.98905468

Table 2 shows how the inference time varies according to the number of synthetic outliers present. The number is the median result between all tested sizes of ontologies. Analyzing this data in the graph in Figure 9, there is again a tendency of increase in time for regional inference when more synthetic outliers are added. The dependency inference, in this case, stays stable between the different tests. Again, scalability trends suggest that the proposal is applicable in large industrial settings.

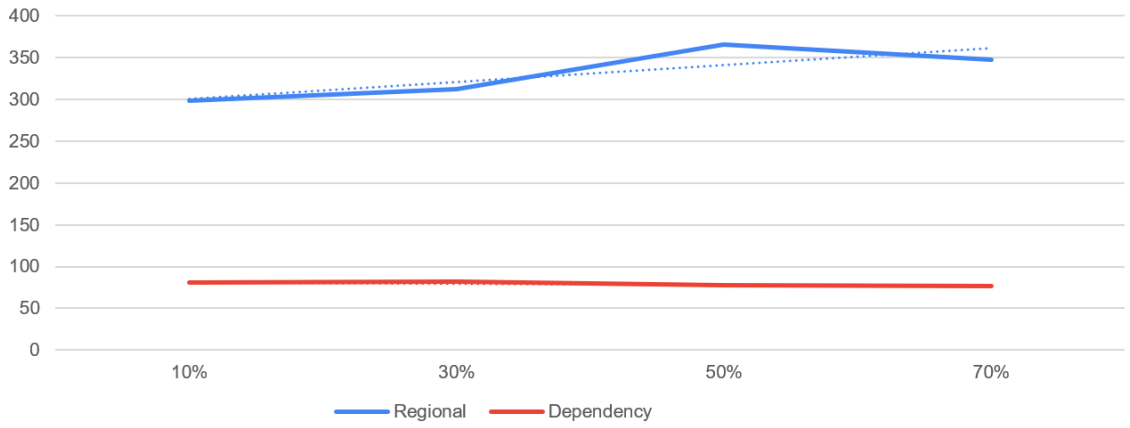

Figure 9. Time variation while ontology size is increased.

5. Conclusion

The proposed architecture tackles challenging aspects introduced by Industry 4.0, including the large number of sensors and the complexity of their dependencies. By combining ontology-based knowledge representation and graph-based inference, our proposal enables timely decision-making in a failure scenario with such challenging settings. As far as we know, this is the first time that ontologies and outlier detection were combined in such a manner.

Our tests showed the feasibility of the proposal, both in terms of semantics (using manually created ontologies and qualitative analysis of the results) and scalability. The performance tests evidence good performance even in the off-the-shelf hardware used. The fact that the processing time increases linearly as the number of entities in the ontology increases makes it more predictable in terms of required hardware as the industrial installations expand.

The implementation used off-the-shelf languages and software, showing that the solution is effective and possible in a real production environment. Also, all tools used were open source.

In a future work, we plan to add a higher-level language for inference specification, allowing more flexibility in the expansion of the inference capabilities. Also, we plan to add an aspect mining step to provide further information about the detected source of the failure.

References

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: a survey. *ACM Computing Surveys*, 2009.

Haibo Cheng, Peng Zeng, Lingling Xue, Zhao Shi, Peng Wang, and Haibin Yu. Manufacturing ontology development based on industry 4.0 demonstration production line. *Third International Conference on Trustworthy Systems and Their Applications*, 2016.

Lei Duan, Guanting Tang, Jian Pei, James Bailey, Akiko Campbell, and Changjie Tang. Mining outlying aspects on numeric data. *Data Mining and Knowledge Discovery*, 2015.

Daniel Fleischhacker, Heiko Paulheim, Volha Bryl, Johanna Völker, and Christian Bizer. Detecting errors in numerical linked data using cross-checked outlier detection. *The Semantic Web - ISWC 2014*, 2014.

Anuroop Gaddam, Tim Wilkin, Maia Angelova, and Jyotheesh Gaddam. Detecting sensor faults, anomalies and outliers in the internet of things: A survey on the challenges and solutions. *Electronics*, 2020.

W3C OWL Working Group. Owl 2 web ontology language document overview (second edition), 2012. URL `https://www.w3.org/TR/owl2-overview/`.

N. Guarino. *Formal Ontology in Information Systems*. IOS Press, 1998.

D. Hawkins. *Identification of Outliers*. Chapman and Hall, 1980.

Gustavo H. Orair, Carlos H. C. Teixeira, Wagner Meira, Ye Wang, and Srinivasan Parthasarathy. Distance-based outlier detection: consolidation and renewed bearing. *Proceedings of the VLDB Endowment*, 2010.

Ali Ahmadian Ramaki, Reza Asgari, and Reza Ebrahimi Atani. Credit card fraud detection based on ontology graph. *International Journal of Security, Privacy and Trust Management*, 2012.

Theodoros Toliopoulos, Christos Bellas, Anastasios Gounaris, and Apostolos Papadopoulos. Proud: Parallel outlier detection for streams. *SIGMOD Record*, 2020.

Jianxin Wang, Ming K. Lim, Chao Wang, and Ming-Lang Tseng. The evolution of the internet of things (iot) over the past 20 years. *Computers and Industrial Engineering*, 2022.

Yufeng Yu, Yuelong Zhu, Shijin Li, and Dingsheng Wan. Time series outlier detection based on sliding window prediction. *Mathematical problems in engineering*, 2014.