Domain Model-Based Data Stream Validation for Internet of Things Applications

Simon Pizonka
Humboldt-Universität zu Berlin
Berlin, Germany
simon.pizonka@hu-berlin.de

Timo Kehrer
Humboldt-Universität zu Berlin
Berlin, Germany
timo.kehrer@informatik.hu-berlin.de

Matthias Weidlich
Humboldt-Universität zu Berlin
Berlin, Germany
matthias.weidlich@hu-berlin.de

ABSTRACT

The Internet of Things (IoT) has become ubiquitous, connecting an ever increasing amount of devices, many of which are online 24/7 and send data continuously. The quality of these data plays a pivotal role for many IoT applications, which demands for continuous monitoring and validation of streaming data in order to spot and react to potential errors. Yet, implementing such validation facilities requires a deep understanding of the processed data. IoT developers are often bothered with technical details such as the data structure and format, which is not only tedious but also prone to errors.

In this paper, we advocate a model-based approach to this problem, deriving validation facilities from models written in the Vorto modeling language, an emerging domain-specific modeling language for declaratively describing basic characteristics of IoT devices. We evaluate our approach and prototypical implementation using the so-called Intel Lab Data as experimental subject. While the experiment showcases the feasibility of our approach, we also identify limitations to be addressed in future work to fully realize our vision of domain model-based data validation for IoT applications.

KEYWORDS

Model-driven Engineering, Internet of Things, sensor devices, stream processing, data validation

1 INTRODUCTION

The Internet of Things (IoT) has become reality and is constantly growing. There are several forecasts of how many IoT devices we will have in the future. One frequently cited source is the analyst company Gartner Inc., which expects around 20.4 billion IoT devices by 2020 [1]. All these devices will be connected to the internet, many of them will be online 24/7 and will send continuous streams of data which need to be processed and stored. The quality of these data plays a pivotal role for many IoT applications, which demands for continuous monitoring and validation of streaming data in order to spot and react to potential errors.

To date, data validation facilities are often implemented ad-hoc and in a manual fashion. This is a tedious task, prone to errors, which not only requires expert knowledge in the respective application domain, but also a deep understanding of various technical details, such as the format and structure of the processed data, how to plug-in the validation routines into a suitable stream processing framework, etc. Approaches towards more automated data validation solutions have started to be developed, yet, are still in their infancy. One common idea is to detect anomalies in data streams by using a learning approach where a statistical reference model is created based on historical data representing normal behavior, e.g., as presented in [21].

In this paper, we propose a complementary approach to data stream validation for IoT applications, in which validation rules are derived from pre-defined domain models which are being interpreted in a stream processing framework at run-time. Following basic principles of Model-Driven Engineering (MDE) [2], our goal is to specify device properties in a high-level and platform-independent fashion, while the validation itself is achieved in a fully automatic way without requiring the need for manual development efforts on the technical level. We present a reference implementation of our approach, referred to as VortoFlow, in which IoT device information is modeled using the Vorto DSL[2], an emerging domain-specific modeling language for declaratively describing basic characteristics of IoT devices, and these device models are interpreted within Apache Beam[3], serving as an abstraction layer over a set of widely used stream processing frameworks. We evaluate our approach and prototypical implementation using the so-called Intel Lab Data [13] as experimental subject.

While the experiment showcases the feasibility of our approach, we also identify limitations which need to be addressed in future work in order to fully realize our vision of domain model-based data validation for IoT applications. However, we believe that the automated derivation of data validation facilities from domain models is another consequent step in leveraging MDE principles for the development of IoT applications [3, 4, 14].

The remainder of the paper is structured as follows. Section 2 introduces a running example which motivates our approach, an overview of which is presented in Section 3 and whose applicability is evaluated in Section 4. Related work will be studied in Section 5, before we conclude and outline future work in Section 6.

2 MOTIVATING EXAMPLE

With the IoT, many objects of our daily life get connected and controllable via the internet. We would like to pick-up here a kitchen blender serving as running example. Traditionally, a kitchen blender has a physical interface, comprising a rotary knob to turn on and off the device and to control its speed, and additional buttons to enable advanced features, e.g., for crushing ice cubes or preparing smoothies. Now, imagine there is a mobile application to monitor the kitchen blender. This application can show, e.g., whether the blender is active, the rotation speed, and which of the advanced features are enabled.

The kitchen blender periodically sends messages comprising various meta-data (e.g., a timestamp) as well as information about its

[1] http://www.gartner.com/newsroom/id/3598917

[2] https://www.eclipse.org/vorto/

[3] https://beam.apache.org/

```
1  @ProcessElement
2  public void processElement(ProcessContext c) {
3    String entry = "";
4    try {
5      entry = c.element();
6      String [] elms = entry.split(";");
7      // parse values
8      int rotations = Integer.parseInt(elms[0]);
9      long runTime = Long.parseLong(elms[1]);
10     Date dateTime = dateTimeFormat.parse(elms[2]);
11     // new data structure
12     ObjectNode root = mapper.createObjectNode();
13     root.put("rotations", rotations);
14     root.put("runtime", runTime);
15     root.put("datetime", dateTimeFmt.format(dateTime));
16     // output
17     c.output(dataOK, mapper.writeValueAsString(root));
18
19   } catch (Exception e) {
20     LOG.error("Processing␣failed␣for:␣" + entry, e);
21     c.output(dataError, entry);
22   }
23 }
```

Listing 1: Apache Beam user-defined function implementing a message data conversion.

current state (activity, rotation speed etc.). Messages are transmitted in a device-specific message format. In our example, in a comma-separated string encoding in which single values are separated by a semicolon. Each value represents a dedicated part of the message, depending on its position. A recurring problem is to convert such a native message format into some other data structure, e.g., a structured JSON object with is better suited for further processing in the cloud. Here, we use Apache Beam as a software abstraction layer over a concrete stream processing engine. Our exemplary data transformation of incoming messages may be plugged-in into Apache Beam by providing a so-called user-defined function, a Java implementation of which is shown in Listing 1. As we can see in lines 8 to 10, dedicated data values may be accessed via their fixed position within the comma-separated message string, while the type-specific parsing of values is delegated to built-in Java functions. If parsing of an input value fails, the parsing exception is caught and the message is marked as invalid (lines 19 to 21). Otherwise, a simple JSON object representing the message is constructed in lines 12 to 15.

In addition to the pure syntactic validation of the message string, we would now like to progress towards a more semantic data validation by incorporating domain knowledge. For example, from the data sheet of the kitchen blender, we know that it has has a maximum rotation speed of 12.000 rounds per minute, which means that the value range for rotation speed is between 0 and 12.000. Listing 2 shows the code we need to add to validate the value range of the rotation property. The code snippet is to be inserted after parsing the input values and before creating the JSON object. This additional code is required for each property to validate the value range. Typically this code is handwritten. Similar checks may be added for other properties of the blender.

As we can see, even for our small example, developers of IoT applications are typically confronted with multiple technical details such as message protocols, data formats, etc. Moreover, a lot of

```
1  if(rotations < 0) {
2      throw new
3          MinConstraintViolation("Rotations␣<␣0");
4  }
5  else if(rotations > 12000) {
6      throw new
7          MaxConstraintViolation("Rotations␣>␣12000");
8  }
```

Listing 2: Implementation of additional check routines validating the value range of the rotation property.

repetitive yet very schematic code needs to be produced in order to implement data validation facilities such as the rather simple checks used in our running example. Finally, the hand-crafted validation routines are highly technology-specific and cannot be easily transferred to other platforms and frameworks.

3 APPROACH AND PROTOTYPICAL IMPLEMENTATION

In this section, we present our approach and prototypical implementation to combining a domain model with a stream processing system in order to validate data streams in IoT applications. Specifically, as illustrated in Section 3.1, we use the Vorto DSL to declaratively describe the capabilities of IoT devices, which includes the platform-independent specification of message structures and further data integrity constraints such as the measurement range of a sensor device. Such a model can be used in a stream processing system to validate incoming data streams. To easily adapt to multiple stream processing systems, we use Apache Beam as an abstraction layer over several standard stream processing engines for that purpose. An overview of our integration with Vorto, referred to as *VortoFlow*, is presented in Section 3.2.

3.1 Device Information Modeling in Vorto

The Vorto project, which serves as a basis for our approach and prototypical implementation, aims at achieving interoperability among IoT device manufacturers, platform providers and application developers through the generation of platform adapters (aka. stubs) from domain models. Therefore, Vorto provides a high-level domain-specific modeling language, the Vorto DSL, to describe the functionality and characteristics of IoT devices in terms of so-called *Information Models*. An information model contains one or multiple *Function Blocks*. These function blocks are structured into five *Sections*. The *Configuration* section defines read- and writable properties to configure a device, while the *Status*, *Fault* and *Events* sections define readable properties that define the device's current status, fault states, and publishable messages, respectively. Properties are typed, and a type may be a primitive type or a complex type. The latter can contain further complex types, primitive types and enumerations. Finally, the *Operations* section defines operations that can be invoked on the device from, e.g., external applications.

Listing 3 shows a function block describing the kitchen blender of our running example. In the event section (lines 14 to 20), we declaratively describe the structure of a message called `speed` which is periodically published by the device. In contains the same properties (`rotations`, `runTime` and `dateTime`) as used in

```
namespace de.hu_berlin.blender
version 1.0.0
displayname "Blender Function Block"
functionblock Blender {
    configuration {
        mandatory firmwareVersion as int
    }
    status {
        mandatory speed as int <MIN 0, MAX 100>
        mandatory powerOn as boolean
        optional iceCrushActive as boolean
        optional smoothieActive as boolean
    }
    events {
        speed {
            mandatory rotations as int <MIN 0, MAX 12000>
            mandatory runTime as long <MIN 0>
            optional dateTime as dateTime
        }
    }
    operations {
        mandatory updateFirmware()
    }
}
```

Listing 3: Vorto information model describing the characteristics of a kitchen blender.

our manual implementation in Section 2. However, note that we can now use the MIN and MAX constraints of the Vorto DSL to define the value range of the rotations property.

3.2 Data Stream Validation through Model Interpretation in Apache Beam

Figure 1 illustrates how a Vorto model can be used in an IoT scenario. Specific code generators, collectively referred to as Vorto Generator in Figure 1, enable the generation of platform adapters supporting communication and message exchange between components on different platforms. Receiving the measurements and readings from sensor devices, the platform adapter is capable of transforming the incoming data to a format which the IoT platform can process. In our prototypical implementation, device-specific messages are converted into a structured JSON object, which is passed to the IoT platform running in the cloud. The platform receives the incoming data stream and forwards it for data validation which, in our case, takes place in Apache Beam on some concrete stream processing engine. The validation itself is performed in a fully automated way by the *Data Validation* component contributed by VortoFlow. The validation rules which are to be executed on the data stream are obtained from the domain model. If the validation fails, the message is marked and equipped with details about the validation error.

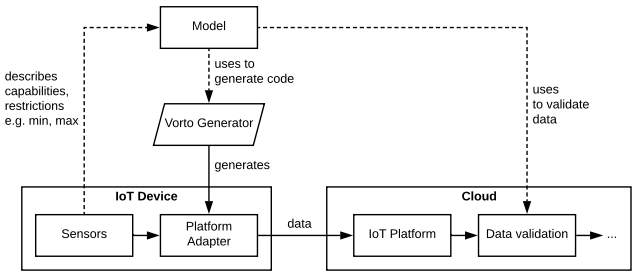

Figure 1: Using VortoFlow in an IoT pipeline.

Technically, the realization of VortoFlow is based on the following design decisions. First, instead of generating data validation components from domain models, we choose an interpretative approach in which a generic data validation component interprets the domain model at run-time. This enables a flexible deployment process when the domain model changes. Second, this generic data validation component is implemented as a Java library which can be included in an Apache Beam project. The idea is that, besides model validation, further processing steps can be included in the final project. This is resource-efficient because the messages are already loaded. Finally, the current processing function in VortoFlow is stateless, and thus can be included without much effort.

The implementation of the generic data validation component is rather straightforward. To date, VortoFlow supports syntactical conformance checking w.r.t. the message structure defined by the domain model, and to check value ranges constrained by lower and upper bounds as the one used in our running example. Furthermore, due to the stateless functioning of VortoFlow, only a single message is processed at the same time. Please note that, as positive side-effect of this simplicity, VortoFlow can be operated in stream and batch mode. While the classical use case is to process a stream of incoming real-time data and to give instant feedback, VortoFlow also supports the validation of existing data. This can be helpful for multiple reasons. First of all, data that already exists can be validated and a Vorto model can be created afterwards. Secondly, it allows the user to re-evaluate data if the model has changed.

4 EVALUATION

We evaluate the applicability of our approach and prototypical implementation with respect to two research questions:

- **RQ.1 (Error Detection):** Is it possible in principle to find errors in real-world IoT streaming data using our model-based validation approach?
- **RQ.2 (Scalability):** Does the validation by model interpretation scale up to realistic IoT applications, which process streaming data of high volume and veracity?

4.1 Experimental Subject and Setup

Intel Lab Data. In the Intel Berkeley Research Lab, 54 Mica2Dot Mote[4] boards equipped with weather boards were deployed and operated from February 28 to April 5 2004, measuring the temperature, humidity and light through environment sensors [13]. The collected dataset contains several obvious errors which makes

[4] https://www.eol.ucar.edu/isf/facilities/isa/internal/CrossBow/DataSheets/mica2dot.pdf

```
functionblock MICA2DOTWeatherSensor {
  status {
    // yyyy-mm-dd
    mandatory date as string
        <REGEX "[0-9]{4}-[0-9]{2}-[0-9]{2}">
    // hh:mm:ss.xxxxxx
    mandatory time as string
        <REGEX "[0-9]{2}:[0-9]{2}:[0-9]{2}.[0-9]{1,6}">
    mandatory epoch as int <MIN 0>
    mandatory moteid as int <MIN 1, MAX 54>
    mandatory temperature as float
        <MIN -40, MAX 123.8, NULLABLE true>
    mandatory humidity as float
        <MIN 0, MAX 100, NULLABLE true>
    mandatory light as float
        <MIN 0, MAX 1847.36, NULLABLE true>
    mandatory voltage as float
        <MIN 0, MAX 3.2, NULLABLE true>
    }
}
```

Listing 4: Information model: Mica2Dot weather board.

it an ideal experimental subject for our study. To validate the Intel dataset with VortoFlow, we developed a domain model of the weather board using the Vorto DSL, and a test program processing the dataset in Apache Beam.

Domain Model. The domain model of the weather board is shown in Listing 4, its properties are described in the Function Block's status section. Here, we used domain knowledge such as the provided sensor data sheets [15] to derive the respective boundaries. For example, the temperature and humidity sensor have a measurement range from -40°C to 123.8°C and 0% to 100%, respectively.

Test Program. The test program comprises the processing pipeline shown in Figure 2. First, the Intel Lab dataset is loaded as a ZIP file from a Google Cloud Storage Bucket[5]. The file is extracted into a CSV file being processed line by line, each line represents a message which is to be validated. Therefore, each line of the CSV input is transformed to a JSON object which is compatible with our Vorto domain model of the weather board. The JSON object is passed to the generic validation function of VortoFlow and validated w.r.t. the constraints defined by the domain model. All messages which contain an error are written to a text file on a Google Cloud Storage Bucket. The experiments are run on Google Cloud Dataflow[6] and using the latest version of the Apache Beam SDK (2.4.0) for Java.

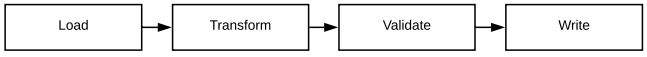

Figure 2: Apache Beam pipeline for processing the Intel dataset used as experimental subject.

4.2 Results

RQ.1 (Error Detection). On the one hand, when validating the dataset, multiple violations of the humidity constraints were detected. Figure 3 shows an example of such a violation. Here, starting at 26th March 2004 00:30:05, the humidity dropped below zero, the respective values are marked by the dotted line. These are values which violate the `MIN` constraint of the `humidity` property defined by our domain model.

On the other hand, some errors passed the validation undetected. For instance, when considering the graph in Figure 4 depicting temperature values recorded by one of the temperature sensors, it is obvious that there is something wrong with the data. However, the exceptional increase in the temperature was not spotted as an error by VortoFlow since all values are still in the valid range of $[-40, 123.8]$ as defined by the weather board model.

Nonetheless, the first example shows that the general approach works and that errors can be detected in principle, which lets us formulate a positive answer for RQ.1.

RQ.2 (Scalability). Table 1 lists the execution times of three independent runs of the pipeline shown in Figure 2 for processing the Intel dataset. For each step, the wall-clock time is given along with the average over all runs. The wall time represents the approximate time taken from initialization to termination. There are multiple reasons why the results are varying from run to run. The read and write tasks require the system to access the network. Here, the

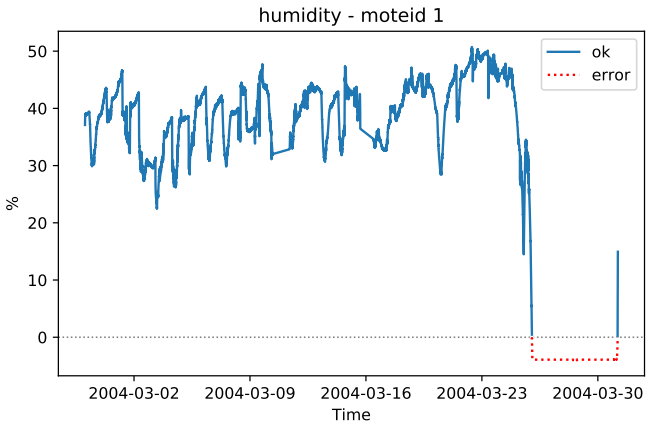

Figure 3: Errors in humidity readings (Mote with id 1).

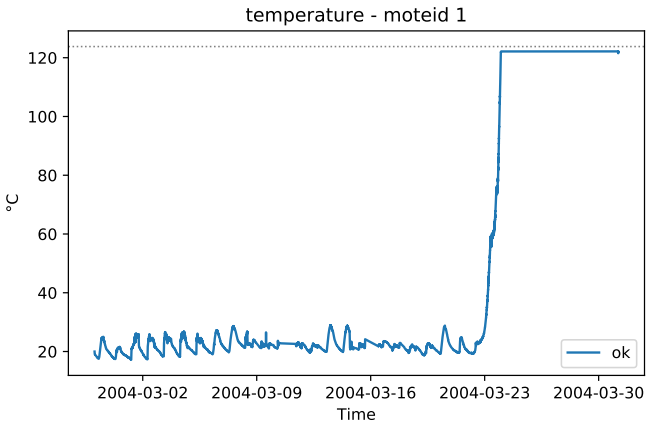

Figure 4: Temperature readings (Mote with id 1).

[5]https://cloud.google.com/storage/docs/json_api/v1/buckets
[6]https://cloud.google.com/dataflow/

	Read	Transform	Validate	Write
Run 1	19 sec.	26 sec.	1 min. 31 sec.	12 sec.
Run 2	17 sec.	24 sec.	1 min. 14 sec.	10 sec.
Run 3	14 sec.	26 sec.	1 min. 22 sec.	10 sec.
Avg.:	~17 sec.	~25 sec.	~1 min. 22 sec.	~11 sec.

Table 1: Execution times of three independent runs of processing the Intel Lab dataset with VortoFlow running on Google Cloud Dataflow.

available bandwidth may vary. Furthermore, the processing of the data requires memory and CPU time, which may be affected by the fact that the hardware is potentially shared with other users.

Although VortoFlow is not optimized for performance at the moment, the experiment with the Intel dataset shows that the validation can be done in a reasonable time. As expected, the validation step needs most of the time with around 1 min 22 sec, about three times as long as reading and writing the dataset, which we consider to be acceptable. The dataset contains 2,313,682 elements which means, per run, around 28,216 messages were validated per second. Thus, RQ.2 can be answered positively as well.

4.3 Discussion

Using the Vorto DSL, it was possible to create a simple yet concise domain model for the considered domain of our experimental subject. This model, in turn, could be used in VortoFlow to detect elements that violate the constraints defined by the domain model. Using a model-driven approach saved us from writing plenty of repetitive code compared to a manual implementation of the same data validation facilities.

However, as indicated by the second example, checking the range of values can be only seen as a first indication for errors. Not very surprisingly, not all the errors comprised by the Intel Lab dataset could be detected using VortoFlow. Therefore, the expressiveness of the Vorto DSL needs to be extended by further kinds of constraints which then need to be checked by the generic validation component. A starting point for inspiration are classical data description languages. JSON-Schema, for instance, has many more features to validate a JSON document compared to the Vorto DSL [20]. Moreover, to address the detection of data errors, outliers and anomalies over time, like the exceptional increase of the temperature value shown in Figure 4, the current stateless processing of single messages is no longer appropriate.

From a technical point of view, there is much room for improvement w.r.t. optimizing the performance of our prototypical implementation. The internal structure is not optimized for a quick access of all property values. For example, each time a validation of a *REGEX* constraint is executed, the regular expression is recompiled. A better approach would be to cache the compiled expressions.

5 RELATED WORK

In this section, we review related work from two different perspectives. First, in Section 5.1, we will have a look at approaches leveraging MDE for the development of IoT applications, before Section 5.2 gives an overview of the state-of-the-art in the field of data stream validation.

5.1 Model-Driven Engineering for the IoT

Both industry and academia have recognized the need for research on a consolidated set of best practices that will guide developers through the manifold challenges of software engineering for the IoT [11]. Model-driven Engineering has been mentioned as one of the key paradigms that bear the potential to tackle these challenges.

One of the predominant challenges addressed by adopting MDE principles are distribution and heterogeneity in the IoT. An example for this is the ThingML (Internet of Things Modeling Language) approach [8, 14]. It supports the modeling of IoT applications from different viewpoints (from the architectural level to the behavior of individual devices) through a modeling language which combines well-established visual modeling constructs (such as state charts and component diagrams) and an imperative yet platform-independent action language. The generation of platform-specific code and adapters is supported through a set of readily available yet customizable code generators for popular programming languages and open IoT platforms (e.g., Arduino, Raspberry Pi, Intel Edison). As mentioned, the Vorto project follows a similar motivation and goal. The Vorto DSL has been used, e.g., to specify manufacturer-independent abstraction layers describing the functions and properties of vehicles on different levels of granularity [12, 19]. We selected Vorto as a technological basis for our work since it is actively developed, maintained and continuously evolved (cf. commit logs on GitHub[7]. Moreover, Vorto is supported as an integral part of the Bosch IoT Suite[8] and based on the widely used Eclipse Modeling[9] technology stack.

Besides heterogeneity and distribution, other values supported by MDE principles such as separation of concerns for collaborative development, automation for enabling self-adaptation at run-time, or reusability of development artifacts have been addressed, e.g., in [4]. More recently, the same group of authors has put a specific focus on the engineering of mission-critical IoT systems [3]. These systems expose further challenges w.r.t. dependability requirements such as reliability, safety and security which may be tackled by exploiting models for the sake of verification.

A domain-specific MDE framework that targets IoT-based manufacturing systems in an Industry 4.0 context has been presented in [17]. Following other approaches to MDE in this domain (see, e.g., the research roadmap presented in [18]), the methodology exploits the UML profiling mechanism [9] to tailor a set of popular UML diagrams towards the specific needs of manufacturing engineers.

However, none of the existing approaches to leveraging MDE for the development of IoT applications exploits domain models for the automated derivation of data stream validation facilities.

5.2 Data Stream Validation

Aiming at scalability of stream validation, it has been suggested to rely on concepts of data stream processing [5]. In that case, languages for data stream processing enable the formalization of

[7] https://github.com/eclipse/vorto
[8] https://www.bosch-iot-suite.com
[9] https://www.eclipse.org/modeling

validation requirements using a well-defined set of streaming operators, including stateless ones such as filters and transformations, as well as stateful operators, e.g., to detect sequential patterns. Data stream management systems then enable the distributed execution of these operators in a compute cluster [6].

The application of these concepts has been illustrated in SVALI (Stream VALIdator) [21], a system that supports two data stream validation modes: In a *model-and-validate* mode, users directly formalize validation requirements as a function over streaming data, which is then continuously evaluated. In a *learn-and-validate* mode, a statistical reference model is learned from samples of normal behavior, which is then used as basis for validation. Either way, validation requirements are defined on the technical level, not connected to conceptual models of the application domain.

In a broader context, a plethora of techniques for the detection of anomalies in data streams has been presented in recent years. They have in common that they assess the characteristics of a stream to detect data that deviate significantly from expected values and, hence, can be thought of as a continuous variant of traditional outlier detection. Common techniques for anomaly detection in data streams are distance-based [1, 7, 10]. Here, a stream element is considered abnormal, if it is far from a pre-defined number of neighboring streaming elements according to some distance function. Moreover, anomaly detection may also exploit the ideas of density-based clustering to flag abnormal stream elements [16] or be based on the angles of data elements in a high-dimensional value space [22]. However, all such techniques characterize anomalies by means of a mathematical model over streaming data and are, therefore, completely disconnected from domain models that describe data sources and the context of a specific IoT application.

6 CONCLUSION

In this paper, we demonstrated how MDE principles can be employed in the development of IoT applications. Specifically, we focused on the question of how to validate data streams emitted by IoT sources through a model-driven approach. We proposed VortoFlow, which builds upon the Vorto DSL for the specification of IoT devices. It enables users to capture validity requirements in terms of value ranges as part of an information model. This models then serves as the basis for online validation of data streams: A generic data validation component, prototypically realized in Apache Beam, interprets the model at run-time and flags invalid data accordingly. We demonstrated the general feasibility and applicability of VortoFlow using the case of a weather board.

In order to fully exploit the potential of model-driven validation of data streams, we intend to extend VortoFlow to support the specification of more expressive validity requirements, along several dimensions. First, the temporal context of data stream elements may be worth to consider, e.g., by validating a sliding average of data stream values over a 1 minute window. Second, information models are specified per device, whereas the Vorto DSL currently does not support the specification of relations between the models of different devices. Enabling the definition of such relations, however, would be useful to capture validity requirements in terms of causal relations of data produced by different devices (e.g., activation of an electric device should be correlated with load measurements at a smart meter). Third, constraint languages commonly adopted in MDE, such as OCL, provide another angle to increase expressiveness of information models w.r.t. to validity requirements.

REFERENCES

[1] Fabrizio Angiulli and Fabio Fassetti. 2010. Distance-based outlier queries in data streams: the novel task and algorithms. *Data Min. Knowl. Discov.* 20, 2 (2010), 290–324.

[2] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2012. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering* 1, 1 (2012), 1–182.

[3] Federico Ciccozzi, Ivica Crnkovic, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Romina Spalazzese. 2017. Model-driven engineering for mission-critical iot systems. *IEEE Software* 34, 1 (2017), 46–53.

[4] Federico Ciccozzi and Romina Spalazzese. 2016. MDE4IoT: supporting the internet of things with model-driven engineering. In *International Symposium on Intelligent and Distributed Computing*. Springer, 67–76.

[5] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.* 44, 3 (2012), 15:1–15:62.

[6] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2016. *Data Stream Management: Processing High-Speed Data Streams*. Springer.

[7] Dimitrios Georgiadis, Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2013. Continuous outlier detection in data streams: an extensible framework and state-of-the-art algorithms. In *Proceedings of the Intl. Conference on Management of Data*. 1061–1064.

[8] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 2016. Thingml: a language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. ACM, 125–135.

[9] Timo Kehrer, Michaela Rindt, Pit Pietsch, and Udo Kelter. 2013. Generating Edit Operations for Profiled UML Models. In *ME@MoDELS (CEUR Workshop Proceedings)*, Vol. 1090. CEUR-WS.org, 30–39.

[10] Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2011. Continuous monitoring of distance-based outliers over data streams. In *Proceedings of the 27th International Conference on Data Engineering*. 135–146.

[11] Xabier Larrucea, Annie Combelles, John Favaro, and Kunal Taneja. 2017. Software engineering for the internet of things. *IEEE Software* 34, 1 (2017), 24–28.

[12] Jeroen Laverman, Dennis Grewe, Olaf Weinmann, Marco Wagner, and Sebastian Schildt. 2016. Integrating Vehicular Data into Smart Home IoT Systems Using Eclipse Vorto. In *IEEE 84th Vehicular Technology Conference*. 1–5.

[13] Samuel Madden et al. 2004. Intel Lab Data. http://db.csail.mit.edu/labdata/labdata.html

[14] Brice Morin, Nicolas Harrand, and Franck Fleurey. 2017. Model-based software engineering to tame the iot jungle. *IEEE Software* 34, 1 (2017), 30–36.

[15] Sensirion Inc. 2011. Datasheet SHT1x (SHT10, SHT11, SHT15) Humidity and Temperature Sensor IC. https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/0_Datasheets/Humidity/Sensirion_Humidity_Sensors_SHT1x_Datasheet.pdf

[16] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. 2006. Online Outlier Detection in Sensor Data Using Non-Parametric Models. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 187–198.

[17] Kleanthis Thramboulidis and Foivos Christoulakis. 2016. UML4IoT: A UML-based approach to exploit IoT in cyber-physical manufacturing systems. *Computers in Industry* 82 (2016), 259–272.

[18] Birgit Vogel-Heuser, Stefan Feldmann, Jens Folmer, Jan Ladiges, Alexander Fay, Sascha Lity, Matthias Tichy, Matthias Kowal, Ina Schaefer, Christopher Haubeck, et al. 2015. Selected challenges of software evolution for automated production systems. In *13th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 314–321.

[19] Marco Wagner, Jeroen Laverman, Dennis Grewe, and Sebastian Schildt. 2016. Introducing a harmonized and generic cross-platform interface between a Vehicle and the Cloud. In *17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks*. 1–6.

[20] Austin Wright, Henry Andrews, and Geraint Luff. 2018. *JSON Schema Validation: A Vocabulary for Structural Validation of JSON*. Working Draft. IETF Secretariat. https://tools.ietf.org/html/draft-handrews-json-schema-validation-01

[21] Cheng Xu, Daniel Wedlund, Martin Helgoson, and Tore Risch. 2013. Model-based validation of streaming data: (industry article). In *The 7th ACM International Conference on Distributed Event-Based Systems*. 107–114.

[22] Hao Ye, Hiroyuki Kitagawa, and Jun Xiao. 2015. Continuous Angle-based Outlier Detection on High-dimensional Data Streams. In *Proceedings of the 19th International Database Engineering & Applications Symposium*. 162–167.