IoT-TEG 4.0: A New Approach 4.0 for Test Event Generation

Antonio Velez-Estevez, Lorena Gutiérrez-Madroñal, and Inmaculada Medina-Bulo, *Member, IEEE*

***Abstract*—The Industry 4.0 (I4.0) is a paradigm settled down by the introduction of the Internet of things (IoT) into the production and manufacturing environment. I4.0 promotes the connection of physical items such as sensors, devices, and enterprise assets, to each other and to the Internet. The information that flows through these items is vital because it serves to make relevant decisions. One of the main features of I4.0 is its adaptability to the human needs, this means that the items included in the I4.0 network are heterogeneous and they are large in number. The majority of I4.0 papers, which are focused on testing, describe a specific system or part of the I4.0 network. We have not found any paper that undertakes the testing of multiple connected IoT devices that will receive, process, and make decisions according to the complex and real data that travel through the network. In this article, we present IoT-TEG (Test Event Generator) 4.0, which is based on the test event generator system IoT-TEG [1]. IoT-TEG 4.0 provides two new main contributions: the generation of test cases, which can include all the different types of data that the connected I4.0 devices under study can manage, and real-time testing. Additionally, its validation using real IoT programs is included and the results show that IoT-TEG 4.0 allows us to conduct tests that mimic real IoT system behaviors.**

***Index Terms*—Event generator, Industry 4.0, Internet of things (IoT), IoT testing, IoT-TEG, testing.**

I. Introduction

DURING the last few years, the number of devices connected to the Internet has significantly increased thanks to their lower cost. In 2009, Internet of Things (IoT) was defined as [2].

"A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these "smart objects" over the Internet, query their state and any information associated with them, taking into account security and privacy issues." This technological innovation and customer demand for new technology and services are leading to unknown challenges, which is changing the industry. This transformation will influence how entities and organizations will be managed according to new incentives, and environmental and context configuration. This change is being done very quickly, and it is known as the fourth industrial revolution, I4.0 [3].

Because IoT eases the automation and exchange of information, its incorporation is one of the driving forces of the evolution of the industry to I4.0 [4], [5]. In fact, the critical factor for I4.0 is its reliance on the interoperability made possible by IoT [5]. In addition, Rüßmann *et al.* [6] predicted that the number of networked sensors and machines will keep increasing, and more devices, even unfinished products, will enrich the I4.0. This means that the more information is generated, the more decentralized will be the analysis and decision-making, and real-time responses will be needed. The major technical aspects of I4.0, such as the decentralization of the functionalities and the allocation of components in the field, are not addressed by the current testing methods and tools [7]. The validation of I4.0 plays more and more an increasing role and can no longer be handled by isolated solutions.

Vaidya *et al.* [8] enumerated the issues and challenges in I4.0. One of them highlights the necessity to ensure the quality and integrity of the data recorded from the systems. The annotations of the data entities are very diverse and it is a growing challenge to incorporate diverse data with different semantics for data analysis [9].

Therefore, the main challenge to solve the mentioned issue is to develop a testing tool capable of generating test cases no matter the type of I4.0 source. As we mentioned, there is a large number of heterogeneous devices that can be connected to I4.0. An I4.0 testing tool must consider all type of data and be capable to generate it. Other issue to have in mind is the frequency that the information is sent by each connected I4.0 item; the I4.0 testing tool has to generate not only the specific data but also in the right moment.

As we mentioned, IoT is responsible for exchanging the information between the connected I4.0 items. Furthermore, getting information that is transmitted in the form of events, is a very important prerequisite for evaluating and assessing IoT devices [10]. So, the test cases to generate by the I4.0 testing tool must contain events from different connected IoT devices as well as sending them with the same frequency that the real device does.

Manuscript received 26 October 2020; revised 21 April 2021; accepted 6 June 2021. Date of publication 1 July 2021; date of current version 1 September 2022. This work was supported in part by the Ministry of Economy and Competitiveness (Spain) under the National Program for Research, Development and Innovation, Societal Challenges Oriented under Project FAME RTI2018-093608-B-C31 and in part by the Plan Propio de Investigación of the University of Cadiz and Grupo Energético de Puerto Real S.A., under Project GANGES PRCI0003. Associate Editor: F. Wotawa. *(Corresponding author: Lorena Gutiérrez-Madroñal.)*

The authors are with the Department of Computer Science and Engineering, University of Cádiz, 11003 Cádiz, Spain (e-mail: antonio.velez@uca.es; lorena.gutierrez@uca.es; inmaculada.medina@uca.es).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2021.3087781.

Digital Object Identifier 10.1109/TR.2021.3087781

In addition to the mentioned issues, the replication of data generated by real world processes is difficult because in many situations the data to be simulated comes from different devices and/or components, and the behaviors of some of the components must be first analyzed in order to replicate this data. For instance, if the event to be detected is a fall of an elderly person, the involved measured data includes the acceleration of the body, the heart rate, and the length of time, and depending on the type of fall, the body acceleration may have different behaviors. Additionally, it is very challenging, even dangerous, to obtain data from some of the critical situations that we would want to detect, such as adverse environmental conditions, a rise or fall in blood pressure, and a heart attack, among others.

In this article, we define as *events 4.0* the test events that have to mimic the behavior of IoT systems connected in I4.0. They have to simulate the data generated from different event types. Moreover, the generation of distributed events 4.0 over time may be needed because some of that information may be used to build another event type. To the best of our knowledge, these two features are not included in any other testing system in a general way. The other systems are specific to particular use cases.

So the goals of this article are to generate test cases, which cover these events 4.0 as well as sending them mimicking the connected I4.0 devices; so real-time testing will be available. To achieve these main goals the following contributions have been done.

1) Generation of test events 4.0:
   1) Definition of events 4.0. New ways of defining event values are included, allowing any realistic event instances.
   2) Creation of the *ExprLang* language. This language has been created to define complex arithmetic operations, which allow IoT-TEG 4.0 to define events 4.0.
2) Generation of events 4.0 over the time, allowing a real-time testing:
   1) Generation of test events 4.0 that simulate the behaviors of IoT systems with regard to the time and different data types. It has to be taken into account the frequency of the events 4.0 or if sending an event 4.0 triggers the sending of another one. The inclusion of this feature allows a real-time testing.
3) Creation of an I4.0 testing tool:
   1) Providing usability: a graphical interface that makes testing easier.
   2) Integrating all the previous contributions to IoT-TEG 4.0.

The results show that IoT-TEG 4.0 allows us to conduct tests that mimic real IoT system behaviors. In addition, IoT-TEG 4.0 is also multipurpose because it not only allows different behaviors to be defined but also the defining of different event structures and formats. So, thanks to all the functionalities of IoT-TEG 4.0, best testing can be done and more reliable IoT systems can be developed.

IoT-TEG 4.0 is developed based on the IoT-TEG system [1], which was developed to help programmers generate test events for testing programs that process events. IoT-TEG has been used to test queries in event processing language (EPL), a query language used for managing and running events [11]. Although IoT-TEG was created to generate different types of test events, it was limited. The complexity of the event types was bounded to simple arithmetic operations. Moreover, IoT-TEG did not allow to define an event using event attributes from other defined event types, which is essential in the IoT context. Consequently, it was impossible to define and generate the events 4.0. The output of IoT-TEG was a file with the desired event type values, and it was not possible to generate events distributed in time. The frequency of the events was not taken into account, as well as relations between events (event A triggers the sending of event B). Basically, real-time testing could not be done as other tests where the time is critical. Finally, IoT-TEG runs in console, and it was not easy for the users to work with the tool. The inclusion of a graphical interface helps to understand the test event generation process and eases the use of an I4.0 testing tool.

The remainder of this article is organized as follows. In Section II, we detail the background of the research, basic concepts about events and the IoT-TEG system. Section III introduces the ways of the added events 4.0 were defined. Then, Section IV explains the events 4.0 generation proposed in this article. Section V shows the graphical interface of the tool, and Section VI explains how the validation was achieved. Section VII reviews the related work. Finally, Section VIII concludes this article.

II. Background

This section reviews the basis of the events, their characteristics and how they are handled. In addition, the previous version of IoT-TEG will be explained.

A. Events

The information transmitted between IoT systems is sent in the form of events. According to Luckham [12], an event instance is as follows.

1) *Event instance*: every situation that may require a reaction from the system.
2) *Event type*: classifies the event instances and describes their conceptual features. Each event instance belongs to a certain event type.

Luckham [12] also proposed a few recommendations to define an event type.

1) All events must be instances of an event type.
2) An event has the structure defined by its type.
3) The structure is formed as a collection of *event attributes* (an *event attribute* could have a simple or complex data type).

The events could have particular structures that depend on the transmitted information; their structure is defined by the data that the sensor receives and how the IoT device transforms it. The generated information, also known as the event, has to be processed and analyzed, in order to be useful in decision-making. To do that, new tools have emerged that have been integrated into critical processes in industry. Luckham [12] tried to address the problem of making decisions in these contexts by introducing

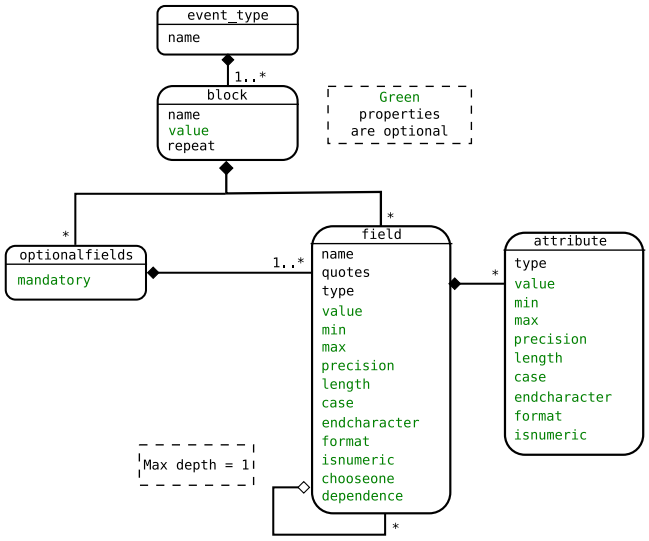


Fig. 1. IoT-TEG event type specification.

complex event processing (CEP). Later, Luckham identified the "top worries about future directions," [13] in which he said, "higher level languages for event patterns and rules, supporting a rich selection of event relations (e.g., temporal, causal, and similarity relations) need to be developed."

According to this view, Schiefer *et al.* [14] come to the conclusion that the specification of a standard language for event processing is a key issue. Today, there are a few EPLs that share the same target of processing events.

In [15], several reasons are listed explaining why is problematic accessing real-world events for testing.

1) Their occurrence and their content is hard to control.
2) Exceptional events have a very low frequency or do not occur at all during the test phase.
3) Sometimes there is not enough event data available.
4) It may be costly to connect to some event sources.

To mitigate these problems of testability of event-driven processes several test events generators have been created (see Section VII), as well as IoT data platforms which offer their channels to save events, the IoT device data (see Section VI).

B. Internet of Things-Test Event Generation

IoT-TEG[1] is an open-source tool to help programmers generate test events to test programs that process events. Fig. 1 shows the event type specification proposed in [1] to generate test events. This specification fills the lack of event type structures, providing a way to define *event instances* using the *event type* specification. The specification was proposed after analyzing more than 500 event types from 10 IoT data platforms: Xively [16], Buglabs [17], GroveStreams [18], SensorCloud [19], ThingSpeak [20], and Zetta [21], among others.

As can be seen in Fig. 1, an event type is composed of blocks. An event type can have as many blocks as the user wants, but one of them must provide the attribute *repeat*, which indicates how many events of that type will be generated. The block can be formed by fields, where each field determines an *event attribute* of the event type that the user is defining. In addition, each field (that is, each *event attribute*), if complex, can be composed of other fields and attributes.

The input of IoT-TEG was an XML file with the event type definition. Then, that definition was validated and the events were generated. Example 1 illustrates how a user of IoT-TEG had to define an event type using XML. The event type *SmartHome* has two *event attributes: temperature* and *humidity*, which are defined using the field parameter.

Example 1: SmartHome event type definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<event_type name="SmartHome">
  <block name="values" repeat="150">
   <field name="temperature" quotes="true"
       type="Integer" min="5" max="45">
   </field>
   <field name="humidity" quotes="true"
       type="Float" precision="3">
   </field>
  </block>
</event_type>
```

The result of generating this event type is 150 events of the *SmartHome* event type (as pointed in the *repeat* attribute in the block tag) in the format selected by the user: XML, CSV, or JSON. The attribute *temperature* is an Integer type, and *humidity* is a Float type. The values of *temperature* are within the [5, 45] range, and *humidity* is a random value with three decimals, as indicated in the field *precision*. An example of a possible result is shown in Example 2.

Example 2: Possible output for SmartHome event type.

```
{"temperature": "10,"
"humidity": "4.342"},
{"temperature": "23,"
"humidity": "78.016"},
{"temperature": "11,"
"humidity": "46.985"},{...}
```

C. Limitations of IoT-TEG

Programmers who test event processing systems usually face problems such as the following.

1) Lack of specific testing data to cover the different situations.
2) Waiting for the source of the data to produce it, or even the absence of data.
3) Not enough data for testing.
4) The event format could be different as a consequence of heterogeneous sources. The programmers have to develop the transformation from the source format to the event-processing program format.

As a solution to solve some of the problems enumerated previously, IoT-TEG was proposed in [1]. The main functionalities of IoT-TEG were as follows.

1) Generates specific test data via the *fixed generation* mode for each field. Using this mode, fields with a specific value can be generated.

[1][Online]. Available: https://gitlab.com/ucase/public/iot-teg

2) Generates random test data within a range through the *random generation* mode for each field, an example is given in Example 2.

IoT-TEG is continuously evolving, and its purpose is to perform different types of tests in systems that manage different event types. For example, in [22], a functionality was implemented to simulate a specific behavior for testing an IoT fall-detection system. This functionality allows for defining the values of the *event attributes* according to the simulation. But these contributions are not enough to cover the Industry 4.0 testing and some research lines are proposed.

1) *Definition of events 4.0.* IoT-TEG generates only *random values* for the events. In this article, we include sequential generation for the simple data types: Integer, Float, Long, String, Alphanumeric, Date, and Time. Moreover the new injectable generation functionality, that has been included (see Section III-B), will allow the user to share the values of the *event attributes* with other, different, events.
2) *Generation of events 4.0.* The time is an important variable in programs that consume events. Therefore, in order to simulate a real device, it would be interesting that IoT-TEG produces N events in a T period (seconds, milliseconds, etc). IoT-TEG only produced results for a single event type definition in each execution. So, with this new functionality the user would be able to generate many events concurrently without having to execute IoT-TEG over and over again.

III. Defining Events 4.0

An event type structure depends on the devices and components that are connected to the IoT system's network. The connected elements emit a large volume of information, which is filtered. That filtered data also affects the event type structure; for instance, an event type structure can be composed of the values of filtered data from one or more devices. Considering these requirements, the event type definition has to be abstract and contemplate a variety of options. The greater the abstraction, the more event type definitions can be designed.

Our main goal is to generate test events that could not be taken apart from the ones generated by the original source of the IoT system that is under test. The importance of the events and their attributes has been reflected in several studies, where their relevance and possible purposes are highlighted.

A. Defining Events Following a Linear Pattern

Luckham *et al.* [23] revealed that complex event processing operates on sets of events and on the relationships between events. Relationships between events can be specified in event patterns in maps and filters. Events can have various relationships to one another; one of them, is *time* with respect to a system clock: say that event A happened before event B. Usually, timing is represented by timestamps (using Time data type or Time and Date data types). Furthermore, a sequence of timestamps could be needed to test, while another sequence of data could be also needed: an ordered list of items, the assignation of plate numbers to cars, and so on.

This way of defining event 4.0 types allows us to simulate linear patterns in the attributes of the generated events 4.0, and we have tried to cover all the sequence possibilities. To develop this functionality four new parameters have been added to the field specification shown in Fig. 1.

1) *begin* defines the initial value of the sequence.
2) *step* defines the value of the increment or decrement within the sequence.
3) *end* defines the maximum value of the sequence.
4) *unit* defines the unit of the last three parameters. It is valid only when dealing with Time and Date sequences.

The sequential generation has been added for the following types.

1) Types-Integer, Long, and Float: The IoT-TEG 4.0 behavior while is generating these types is as follows.

1) The generation starts with the $begin$ value.
2) The generator increments or decrements the $begin$ value according to $step$.
3) If the generated value overcomes the end value, the generation is restarted by giving $begin$ its original value.

An example of this generation, in JSON format, can be seen in Example 3. The parameter values for this example are $step = 10$, $begin = 0$, and $end = 30$.

Example 3: Integer sequential generation example.

```
{"integerSequence": 0 },
{"integerSequence": 10},
{"integerSequence": 20},
{"integerSequence": 30},
{"integerSequence": 0 },
...(depending on the repeat attribute
    in the block)
```

2) Types - String and Alphanumeric: For these types the generation will behave as previously, but with the following differences.

1) The $begin$ and end values need to be of the String type.
2) The $begin$ and end values must match the alphabets of the String and Alphanumeric types, which are $[A - Z]$ and $[0 - 9A - Z]$, respectively.

See Example 4, which is an example using the values $begin = A, step = 2, end = F$.

Example 4: String sequential generation example.

```
{"stringSequence": "A" },
{"stringSequence": "C" },
{"stringSequence": "E" },
{"stringSequence": "A" },
...(depending on the repeat attribute
    in the block)
```

3) Types-Time and Date: The IoT-TEG 4.0 behavior is the same as before. The difference is the addition of the new parameter $unit$, which will state if the $step$ would be in days, hours, and so on. Furthermore, the introduced value for $begin$ and end must match the given $format$ attribute, which is indicated in the *field* that is being generated.

In Example 5, a Time type example is shown. The values are $begin =$ 10:42, $step = 5$, $end =$ 10:53, $unit =$ MINUTES, and format hh:mm.

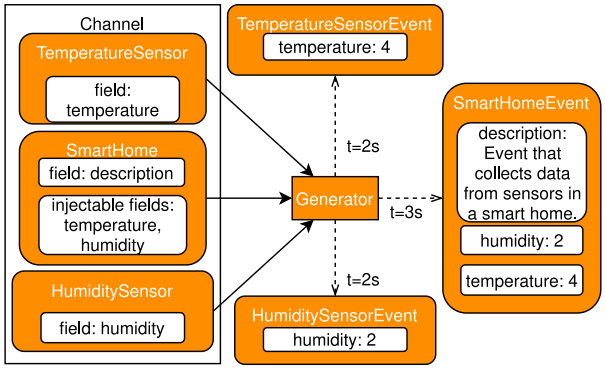


Fig. 2. Injectable generation.

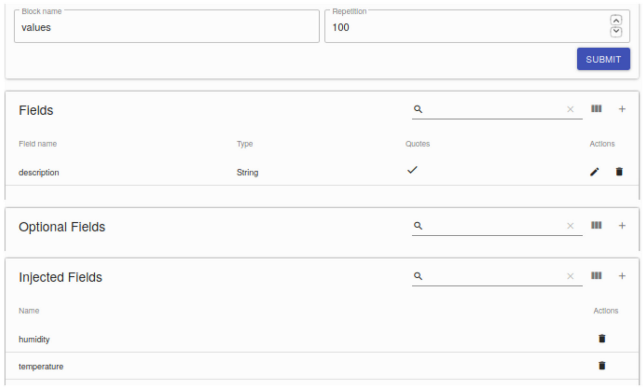


Fig. 3. *SmartHomeEvent* fields definitions using IoT-TEG 4.0.

Example 5: Time sequential generation example.

```
{"timeSequence": "10:42" },
{"timeSequence": "10:47" },
{"timeSequence": "10:52" },
{"timeSequence": "10:42" },
...(depending on the repeat attribute
   in the block)
```

B. Injectable Generation

Ahn and Kim [24] defined *context* as a set of interrelated events or component events with logical and temporal relationships. They explain that "a context detecting fire occurrence" can be detected by inspecting two component events: for instance, the temperature and carbon monoxide content in the air. The component events are interrelated thanks to their structure and their related attributes. This is another type of relationship between events; two attributes A1 and B1 of two different event types A and B are necessary to create a new event type C to trigger a condition, the fire occurrence. Expanding IoT-TEG with this functionality requires keeping a shared generation context among the difierent event types in a channel. A channel is a structure introduced in IoT-TEG 4.0, which represents sets of semantically interrelated event types.

The idea is to inject an event type generated value in another event type that has a different distribution in time (explained in Section IV). For instance, in Fig. 2, we can see a channel with the following circumstances.

1) *TemperatureSensor* event type with
   1) a time planification $\{delay = 0,\ period = 2$ and $unit =$ "seconds"$\}$.
   2) a field *temperature*, Integer type and [0–10]*random generation*.
2) *HumiditySensor* event type with
   1) a time planification $\{delay = 0,\ period = 1$ and $unit =$ "seconds"$\}$.
   2) a field *humidity*, Float type and [0–20]*random generation*.
3) *SmartHome* event type with
   1) a time planification $\{delay = 0,\ period = 3$ and $unit =$ "seconds"$\}$.
   2) a field *description*, String type with *fixed generation* and two injected fields: temperature and humidity.

Once the time distributed generation for these three event types starts, the generator will act as follows.

1) In second 1, a *HumiditySensorEvent* event type will be generated. The value for *humidity* will be, for instance, 4.
2) In second 2, *TemperatureSensorEvent* and *HumiditySensorEvent* event types are generated. Their values are, for instance, four and two, respectively.
3) In second 3, *HumiditySensorEvent* and *SmartHomeEvent* event types are generated. The *SmartHomeEvent* includes a description and two injected values from two event types with different planifications: *humidity* and *temperature*.

Fig. 3 shows the *SmartHomeEvent* fields definition using IoT-TEG 4.0. This event type contains three fields, two of which are injectables. The *humidity* field defined as injectable in the *HumiditySensorEvent* event, and the *temperature*, also defined as injectable in the *TemperatureSensorEvent* event.

This functionality could be used, for example, as a way of collecting data for a dashboard from other event types with different time distributions.

C. Improvement of Custom Behavior Functionality With an Arithmetic Language

Robins indicates in [25] that a large part of CEP is pattern matching, which consists not only of filtering, but also of extracting properties from the event that can be combined with higher level events that compose the output. Those properties can be extracted to be combined or used to identify specific behaviors or classifications. For instance, the functionality implemented to test a fall-detection IoT system not only allows the generation of events that simulate a fall but can also be used to generate a specific type of fall [22]. In the literature, studies focusing on different types of falls can be found, and it is necessary to identify all of them in order to tell them apart from a nonfall. The custom behavior functionality incorporates a specific syntax to perform simple arithmetic expressions in order to meet the necessary requirements. As we can see in Example 6, we have to define variables with values, and then we can use those variables in the rules to create a simulation. A rule, which is composed by a set of parameters, helps us define, partially or totally, the behavior

of an event attribute. The rule definition order affects the event attribute definition. The order of the rules plays a role in the event attribute definition. In Example 6, the set of rules defines the behavior of the acceleration, an event attribute, in a fall.

Example 6: Rules to define a fall.

```
<?xml version="1.0" encoding="UTF-8"?>
<custom_conditions simulations="5">
  <variables>
   <variable name="Base" value="9.81"/>
  </variables>
  <rules>
   <rule weight="0.25" min="$(Base)-0.25"
       max="$(Base)+0.25"/>
   <rule weight="1" min="$(Base)*2"
       max="$(Base)*6"/>
   <rule weight="1" min="0.0"
       max="$(Base)/5"/>
   <rule weight="0" min="$(Base)-0.25"
       max="$(Base)+0.25"/>
  </rules>
</custom_conditions>
```

Nevertheless, the rule expressions syntax was simple and, depending on the fall type, the involved elements to be measured in a fall increase in complexity. The implemented functionality only could manage arithmetic binary expressions {+, -, *, /} and only supported two variables in an expression. Consequently, the functionality had to be improved.

The syntax has been improved by creating a language called *ExprLang* to perform these arithmetic operations. With this language, IoT-TEG 4.0 is capable of defining behaviors using the following.

1) Multiple variables without limit.
2) Arithmetic functions with two parameters: *pow, min, max, copySign, IEEEremainder, hypot, nextAfter,* and *atan2*.
3) Single parameter arithmetic functions are: *abs, sin, cos, tan, sqrt, log, log10, exp, getExponent, signum, asin, acos, atan, toRadians, toDegrees, cbrt, ceil, floor, rint, round, ulp, sinh, cosh, tanh, expm1, log1p, nextUp,* and *nextDown*.
4) Parentheses indicate priority in the expressions.

IV. Generating Events 4.0

In this section, we are going to present the ways of generating test events 4.0, as well as their importance and relevance in IoT testing.

A. Time-Distributed Event Generation

As mentioned in Section III-A, Luckhamand and Frasca [23] indicated the importance of the relationships between events, one of which is *time*. Critical systems always tend to use time as one of the variables for detecting patterns in an event stream. To simulate the events that the event producer systems would emit, a planification for the generation of the events has to be defined. That planification consists of the following.

1) *Period*. This attribute describes how often an event will be generated.
2) *Delay*. This attribute describes how much time IoT-TEG 4.0 has to wait until the first event is generated.
3) *Unit*. This denotes if the last two attributes are in seconds, minutes, and so on.

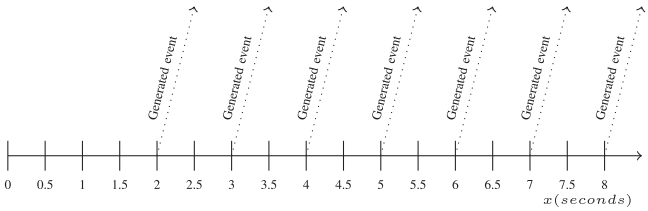


Fig. 4. Time-distributed generation.

Algorithm 1: Formal Description: Time-Distributed Event Generation Algorithm.

Define **ScheduledThreadPoolExecutor** pool;
Define **channelType** ct;
Define **attributes** a(v1, v2,..., vn);
Define **event type** et(ct, a);
Define **time-planning** tp(et);
Define **IoT-TEG 4.0 EventGenerator** eg (et);
Add **(eg, tp)** to **pool**;
Execute **pool**;

In Fig. 4, the event has $\{delay = 2, period = 1,$ and $unit =$ seconds$\}$. This method of event 4.0 generation helps to do a real simulation of the *time* relationship between events for the testing of IoT systems. In order to use the time distributed event generation in other event processing programs, which want to be tested, a new mechanism has to be developed. This mechanism is needed to communicate that a new event has been generated. This mechanism fits perfectly with the producer-consumer model. In that model, publishers do not send the messages to a specific subscriber. On the contrary, they classify them and send them to a message broker, which is responsible for knowing which are subscribed to a certain type of message. In this case, a *topic* classifier has been used.

This has been developed using a *ScheduledThreadPoolExecutor* [26] in order to submit the event generation tasks with a defined schedule and using the parameters explained previously. The Mosquitto broker [27] has been used as a message broker. Mosquitto uses the MQTT protocol [28], which is a publish/subscribe-based messaging protocol. It has been chosen because MQTT is widely used in the IoT context, and it aligns with the model explained earlier. The formal description of the algorithm to generate time distributed events is as follows, see Algorithm 1.

First of all, the ScheduledThreadPoolExecutor and the channelType have to be defined. Then, the event attributes are defined, which are event fields v1, v2... vn that can be any of the provided IoT-TEG 4.0 field types, see Fig. 1. The event type is defined using the channelType and the attributes. The time-planning is established according to the event type as well as the IoT-TEG 4.0 EventGenerator. These two elements are added to the pool which is executed according to the defined time-planning.

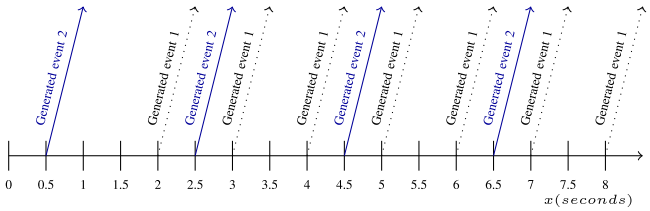


Fig. 5. Example of concurrently generated events.

Algorithm 2: Formal Description: Concurrent Event Generation Algorithm.

/*... same as algorithm 1... */ Add **(eg1, tp1)** to **pool**;
Add **(eg2, tp2)** to **pool**;
/*...*/ Add **(egN, tpN)** to **pool**;
Execute **pool**;

For instance, the event type *HumiditySensor* presented in the following section, is defined within the channelType *Channel*. Then, the set of attributes, in this case, is composed by a single field of type Float with random generation within a range from 0 to 20. After that, we define the time planning to one second. Once the event type and the time planning are defined, an instance of the generator is created for that event type. Finally, the event generator with the time planning are added to the pool and the pool is executed. The ScheduledThreadPoolExecutor will use the time planning to call the generator for that event type every second, according to the time planning frequency.

B. Concurrent Event Generation

Luckham and Frasca [23] discussed how *cause* is another relationship, such as, for instance, event A caused event B to happen. If this relationship occurs several times within a particular time period, a concurrent generation is needed. The changes introduced in the previous sections have made it possible for IoT-TEG 4.0 to generate a huge number of events 4.0 concurrently without the need for executing the program over and over again for each defined event type. This feature allows the user to simultaneously test programs that handle different event types.

For the sake of clarity, an example is provided in the Fig. 5.

1) Event 1 (black dotted line): delay of 2, period of 1, and the unit is *second*.
2) Event 2 (blue solid line): delay of 0.5, period of 2, and the unit is *second*.

The concurrent event generation algorithm is similar to the time distributed generation algorithm, see Algorithm 2. The only difference is that to allow different event types to be generated concurrently, two or more tasks have need to be added to the pool, with the same or different time plannings.

V. Providing Usability

One of the changes made to IoT-TEG was providing it with a graphical interface, which was accomplished by using a Web

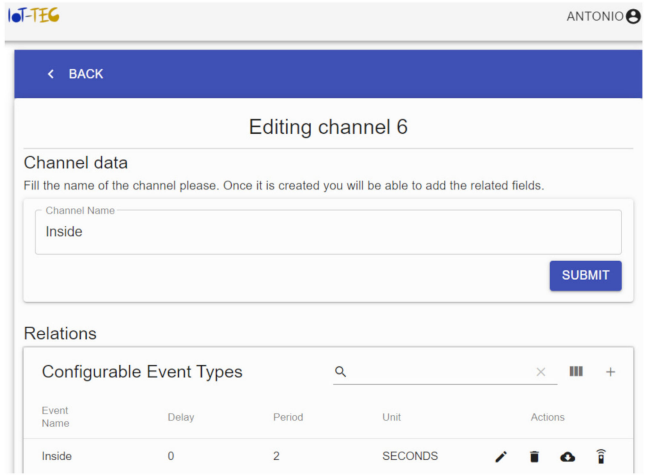


Fig. 6. Inside event type.

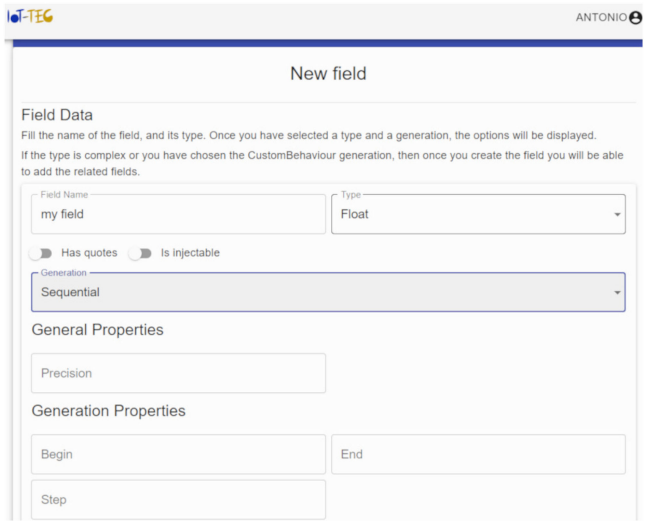


Fig. 7. Definition of a field.

Interface. IoT-TEG 4.0 allows the definition of test events without knowing how the event type is implemented or the generation of test events 4.0. In order to illustrate how the graphical interface works, we have included the more representative parts in Fig. 6. The "Inside event type" screen allows for the modification of event type definitions, starting the time distributed generation and downloading the generated events to a file. Also shown is the screen that allows for the definition of a new field of an event type with regard to its type generation mode and different parameters.

With the IoT-TEG 4.0 graphical interface, we avoid the tedious task of having to define event types via XML (see Section II), giving a more usable experience. This may be checked by comparing the *SmartHome* event type definitions from Example 1, where a previous version of IoT-TEG was used, with figures Figs. 6 and 7, where IoT-TEG 4.0 was used. Moreover, the graphical interface gives us the option to effortlessly inject fields from already defined event types, such as in the example

in Section III-B, where the temperature and humidity fields were previously defined in *TemperatureSensor* and *HumiditySensor*, respectively.

The graphical interface facilitates the definition of events 4.0, and, hence, their generation. The integration of all these contributions gives rise to the 4.0 test event generator system, IoT-TEG 4.0. IoT-TEG 4.0. is an Open Source Software and its source code can be found in the UCASE Research Group repository[2].

VI. Validation

An analysis of real events 4.0 was performed using the following IoT data platforms: Xively [16], Buglabs [17], GroveStreams [18], SensorCloud [19], ThingSpeak [20], and Zetta [21]. These IoT data platforms allow users to connect devices and lodge their data in channels.

Depending on the IoT data platforms, the data are offered in different formats such as JSON, XML, and CSV. If the user wants to obtain the file with the event values, it can be downloaded or the program can be modified to get the event values from the channel and use them as its input. This last option is not always available.

Our first step was to check if the generated test events 4.0 could be differentiated from the ones that the connected sensors and devices were sending to the network. So, we analyzed more than 500 event types from the mentioned IoT data platforms: their structures and their values. Next, each event type was defined using IoT-TEG 4.0, and several test cases were generated.

After checking the transmitted events and comparing them with the IoT-TEG 4.0 test generated events, we can confirm that there were no differences between the real and the generated events. We have compared not only the event type structure but also the possible range of values of the event attributes.

A. Esper Demo Nuclear Case Study

In addition, we took a further step in the validation of IoT-TEG 4.0 and generated events to test the following programs: terminal self-service [29], Transaction [30], DENMEvaP [31], Domotic [32], and Esper Demo Nuclear [33]. All of them worked as expected when tested with IoT-TEG 4.0 generated events. If we look at the Esper Demo Nuclear test case, we can see that this program uses EPL queries to process the incoming events in an EsperTech engine [34]. To test this program, a time-distributed event generation is needed. The program measures the temperature in place through a reading of the core temperature taken every second, and it sends the data to a central monitoring system. The data are Temperature events in degrees Celsius. The program generates random Temperature events and sends them through the Esper processor, where the program has three types of alerts.

1) *Monitor*. This computes the average temperature every 10 s.
2) *Warning*. A warning is sent if there are two consecutive temperatures above 400 °C.
3) *Critical*. It emit an alert if there are four consecutive events and each subsequent one is greater than the previous, where the last is 1.5 times greater than the first and where the first was greater than 100 °C.

To perform the validation, we modified Esper Demo Nuclear program to listen to the broker where IoT-TEG 4.0 sends the generated events 4.0 (see Section IV-A). This modification allowed us to simulate the sensors to test if the alerts were emitted correctly, so we prepared one test case to test if the three alerts were correctly emitted. The created event type for this test case consists of the event type Inside, with a frequency of 2 s. The Inside event type has a block named "block" with one repetition, and the block has a field, "temperature," which is an Integer type that has a sequential generation with the parameters begin $=$ 10, end $=$ 600, and step $=$ 50. We could randomly generate the events as with the original program, but the goal of the test was to check if the three alerts were be correctly emitted. So we decided to simulate a more realistic behavior, where the temperature would increase or decrease steadily. Once the generation started, IoT-TEG 4.0 sent the generated events to the message broker, and Esper Demo Nuclear received those events, producing the three alerts (see Table I). This way, we can define test event types to trigger specific alerts, providing us a quick method of testing consumer systems.

Table I shows the Temperature events generated by IoT-TEG 4.0 and those generated by the original software. IoT-TEG 4.0 sent events every 2 s, and the original program sent Temperature events randomly, so the number of Temperature events generated is different. Given that the main goal was to test that the three alerts appeared according to their implementation, we needed to focus on the time and Temperature event values. The columns Temperature show the value of the Temperature event from IoT-TEG 4.0 and the originals, respectively. As we mentioned, the values of the original events where randomly generated and the IoT-TEG 4.0 events were generated increasing its value, a more realistic behavior. The 10-s average is shown in the *Monitor* column. The alert in this column appears every 10 s, and shows the average temperature according to the received Temperature event values in the 10 s range. In the *Warning* and *Critical* columns, an X indicates when that alert was triggered. As it can be seen, there is an X in the *Warning* alert after receiving the two consecutive values greater than 400, this occurs using either real or IoT-TEG 4.0 generated events. An X appears in the *Critical* column in the IoT-TEG 4.0 scenario, when the received values are 110, 160, 210, 260; the system detects four consecutive events, all are greater than the previous and the last is at least 1.5 times higher than the first which, is also greater than 100. The same happens in the original scenario after receiving the values 133, 196, 391, 395. So, it can be checked that the three alerts were correctly emitted.

B. Fall Detection Case Study

As we mentioned in Section III-C, the custom behavior functionality has been improved using *ExprLang*. This functionality was incorporated into IoT-TEG when the fall detection system prototype [22] needed to be tested. In that first version of

[2][Online]. Available: https://gitlab.com/ucase/public/iot-teg

TABLE I
"ESPER DEMO NUCLEAR" EVENT COMPARISON

Time	Temperature	Monitor	Warning	Critical	Temperature	Monitor	Warning	Critical
	IoT-TEG 4.0 events scenario				Original events scenario			
0sec	10	-	-	-	...	-	-	-
	...	-	-	-	471	-	-	-
	160	-	-	-	497	-	X	-
	...	-	-	-	133	-	-	-
10sec	260	135	-	X	196	304.64	-	-
	310	-	-	-	391	-	-	-
	360	-	-	-	395	-	-	X
	410	-	-	-	311	-	-	-
	460	-	X	-	78	-	-	-

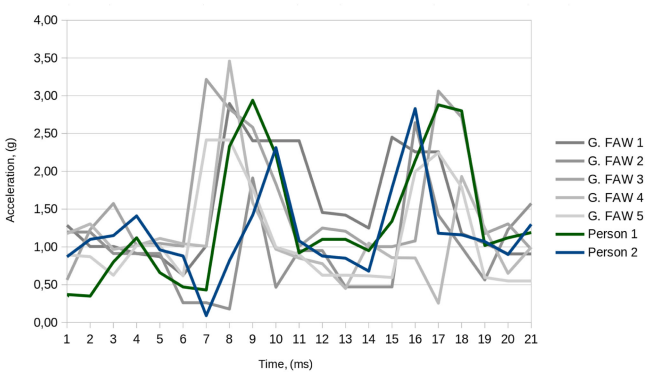


Fig. 8. FAW falls comparison.

TABLE II
VARIABLES FOR FAW FALL DEFINITION

Var. name	Min. value	Max. value	Value
Base	-	-	9.81
ImpactWall	Base+(Base*0.7)	Base*3	-
Impact	Base+(Base*0.7)	Base*3	-

TABLE III
RULES FOR FAW FALL DEFINITION

Weight	Min. value	Max. value	Value	Sequence
0.25	0	ImpactWall*0.35	-	inc
0.25	0	ImpactWall*0.35	-	dec
0.05	-	-	ImpactWall	-
0.25	0	Impact*0.35	-	-
0.05	-	-	Impact	-
0	Base+(Base*0.10)	Impact*0.35	-	-

the fall detection system prototype, one specific type of fall was detected. The generated by IoT-TEG test events helped to validate the prototype.

A second version of the fall detection system prototype has been developed, and with it, a more complex type of fall needs to be detected [35]. The type of fall considered in the second prototype consists of the impact of a person against a wall, followed by that person falling to their knees and then hitting the floor with their chest or back, i.e., a *fall against a wall* (FAW). The test events to generate using IoT-TEG 4.0 have to simulate the acceleration of a person when a FAW fall is produced. In order to analyze the acceleration during a FAW fall, some volunteers replicated this type of fall. The goal is to study the acceleration behavior during a FAW fall to generate test events; thus, the acceleration values are conformed to standard gravity g ($1\ g = 9.81$ m/s^2). There are two peaks during a FAW. The first one is the impact of the person against the wall and the second one occurs when the person hits the floor. A peak is considered a variation in the acceleration greater than 2 g in a time period of up to one second. Fig. 8 shows several examples of the acceleration behavior during a FAW fall.

Once the fall acceleration behavior has been observed, the next step is to define the fall event to generate test events with IoT-TEG 4.0. It is essential to emphasize that to define the behavior of the standard gravity acceleration, several tests must be performed using the custom behavior functionality. The used variables and rules to define a FAW fall using the improved functionality are depicted in Table II and Table III. As we can observe, the complexity of this type of fall is greater than the one explained in Example 6. In the following lines how the FAW fall was defined in IoT-TEG 4.0 is explained.

There are three variables involved in Table II, one is the base of the movement, the standard gravity g, and its value, which is 9.81m/s^2, 1 g. The other two variables represent the two impacts during the FAW fall: hits against the wall and the floor. Their values are calculated once at the beginning of the generation and the value range is from $Base + (Base * 0.7)$ to $Base * 3$.

Six rules are needed to define a FAW fall, see Table III. The weight determines the number of events to be generated that will follow the rule. For instance, if 20 events are going to be used to define a FAW fall, there will be 5 events that follow the first rule, 5 events following the second rule, 1 event representing the wall impact (third rule), 5 events for the fourth rule, 1 event with the floor impact value (fifth rule), and the 3 rest of events will unitil 20 generated following the last rule.

It is needed to highlight that *ImpactWall* and *Impact* variable definitions, as well as the sixth rule definition were not possible in the previous version of the custom behavior functionality. The arithmetic functions were limited, and the parentheses could not be used.

Once we obtain the desired results, the test events are generated as needed. Fig. 8 shows the acceleration values of some of the FAWs generated using IoT-TEG 4.0 with the improved

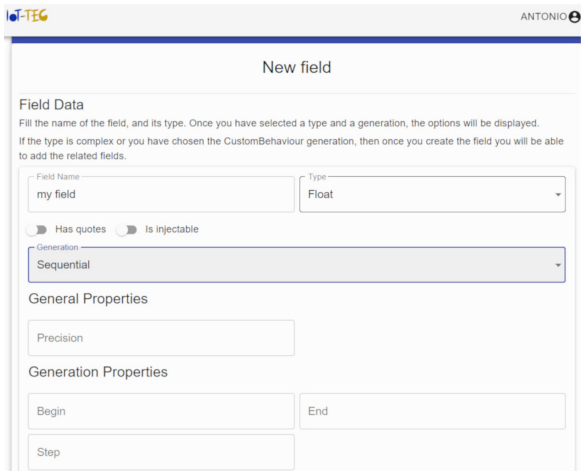
Example 7: EPL output detecting a FAW fall.

```
...
[PersonID, accelS1, accelS2, TimeStamp] [1, 5.63002557906268, 30.5415550991252, 34739]
===== Acceleration variation detected:{a1.accelS1=4.2334213623026,
 a1.accelS2=6.46860718110166, a2.accelS2=30.5415550991252, a2.accelS1=5.63002557906268}
[PersonID, accelS1, accelS2, TimeStamp] [1, 7.66452355613185, 33.9375273406033, 34788]
[PersonID, accelS1, accelS2, TimeStamp] [1, 7.66452355613185, 33.9375273406033, 34790]
[PersonID, accelS1, accelS2, TimeStamp] [1, 27.7464087982103, 19.9658372946816, 34835]
===== Acceleration variation detected:{a1.accelS1=4.2334213623026,
 a1.accelS2=6.46860718110166, a2.accelS2=19.9658372946816, a2.accelS1=27.7464087982103}
[PersonID, accelS1, accelS2, TimeStamp] [1, 27.7464087982103, 33.2498137705071, 34836]
...
```

custom behavior functionality and two real FAW falls from two volunteers.

Fig. 8 illustrates that the IoT-TEG 4.0 generated falls (G. FAW) cannot be taken apart from two real FAW falls. With the aim of testing the second version of the fall detection system prototype not only the real FAW falls were used but also the IoT-TEG 4.0 generated ones. Thanks to IoT-TEG 4.0 generated FAW falls nobody has to replicate a FAW fall anymore. Moreover, false falls as well as noise data can be included in the test cases. The second prototype is a wearable belt with four-sensor-node body area network. The system receives the acceleration from the four sensors. If any of received data contains an acceleration variation, the EPL query of the prototype triggers a message, see Example 7.

For the sake of clarity, the example only uses two sensors in the EPL query instead of four. The EPL query sends information about the events that are being processed in real time. In Example 7, the information from each event is shown: a person identification, *PersonID*, the acceleration from sensor 1 and sensor 2, *accelS1* and *accelS2*, and the time stamp, *TimeStamp*. When an acceleration variation is detected, the message shows the acceleration values that meet the constraints. For instance, sensor 2 acceleration value is 6.468 and, during one second range, the next value of sensor 2 is 30.541; the acceleration variation, in that range, is greater than 2 g (19.62 m/s^2). In the next detected acceleration variation, the first value of sensor 1 is 4.233 and the second value, during one second range, is 27.746; again, the acceleration variation is greater than 2 g. The second fall system prototype [35] was validated using the real FAW fall events and the ones generated using IoT-TEG 4.0 [36].



VII. Related Work

A review of current event generators shows that almost all of them are specific to particular use cases. We will briefly comment on and compare the following four event generators with IoT-TEG (see Section II-B).

1) *Unicorn* [37] is an event processing platform developed by Business Process Technology. It provides services that capture and process real-world events from sources. Unicorn includes an event generator component used to test event-driven applications. The generated events are instances of an event type that was previously defined, and the generated data are random, with two types of distributions: normal and uniform.
2) The *timing system* [38] provides a timing distribution system that includes timing signal generation. The event generator in this tool is responsible for creating and sending timing events to an array of event receivers.
3) *Starcom systems* [39] developed an event generator to deal with huge numbers of events. They ensure that their generator can control the end event action so the tool can filter the exact needed requirements.
4) *WebLogic integration solutions* [40] allows for the management and monitoring of entities and resources required for WebLogic Integration applications. This software includes an event generator module that can trigger events in associated systems.

These four event generators are specific to their use cases. For instance, Unicorn is focused on business process, road, and airfreight transportation. The timing system as well as starcom systems event generators have a commercial purpose, so depending on the client necessities and the business systems to be tested, the events will be generated. The WebLogic Integration Solutions users may integrate a built-in event generator into their projects. This event generator has to be modified afterwards in order to fit the needs of the user and the event. IoT-TEG 4.0 is multipurpose because it not only allows different behaviors to be defined but also the definition of different event structures and formats. Moreover, it lets us simulate the emission of events 4.0 (see Section III). The differences of the aforementioned event generators can be seen in Table IV.

Table IV compares the main features that we have considered as necessary for an I4.0 testing tool after analyzing several works in the literature. The authors highlight the necessity to incorporate diverse data with different semantics [7]–[10]. That

TABLE IV
EVENT GENERATORS COMPARISON

Tools	**Event definition**				**Event generation**		**Usability**
Features	Custom event schema	Linear pattern definition	Injectable generation	Language to define custom behaviour	Time-distributed generation	Concurrent event generation	Graphical Interface
IoT-TEG 4.0	X	X	X	X	X	X	X
UNICORN	X	-	-	-	-	-	X
Timing System	-	-	-	-	X	-	-
Starcom Systems	-	-	-	-	X	X	-
WebLogic Integration Solutions	-	-	-	-	X	X	X

is why the customization of the event type definition is important, the *custom event schema* feature. Related to the event definition, in order to mimic the events from the heterogeneous connected I4.0 devices, the tools have to allow: *linear pattern definition*, *injectable generation*, and a *language to define custom behaviors*. These features enable to incorporate in the event definitions the events relationships defined by Luckham and Frasca [23] and Ahn and Kim [24], as well as the event properties studied by Robins [25]. Moreover, another challenge is to send the information according to the frequency of each connected I4.0 item, an issue identified in [23]. This is solved by allowing different event generations: *time-distributed generation* and *concurrent event generation*. A *graphical interface* whose main purpose is to ease the event definition and generation. The analyzed event generators are listed and an X indicates if they include the feature or not.

Testbeds for IoT can be found in the literature. In fact, a survey of the available testbeds for IoT is presented in [41], where its authors examine different aspects of testbeds. However, the one we are going to focus our attention on is the *emulation of sensor events* by attaching digital-to-analogue converters. The following testbeds emulate sensor events.

1) *w-iLab.t* testbeds provide environmental emulators. These emulators trigger sensor readings on the individual sensor nodes that correspond to emulated environmental events. This also allows replaying of sensor events obtained from a real-world environment on the testbed [42].
2) *FlockLab 2* an extension of FlockLab [43]. FlockLab 2 architecture consists of a testbed server hosting data services and the web interface. A multiplexer crossbar manages the target device. This allows running tests on different target device architectures [44].
3) *SensLAB* testbeds are used for hardware components as well as its software, allowing experiments to be conducted that can also emulate channel characteristics between IoT nodes [45].
4) *Kansei* testbeds support sensor data generation as well as real-time data and event injection [46].

Bouckaert *et al.* [42] recognized that it is hard to represent a real networking environment, even with the most advanced simulation models or desktop testbeds. But their data cannot be adapted to the needs of the different types of tests, such as functional, negative, integration, stress, etc. For instance, IoT-TEG 4.0 can generate specific event types in order to test if the I4.0 connected device processes them correctly (see Section VI-B). The negative testing consists of analyzing how the connected I4.0 devices react when the received event contains a nonexpected value type (a string instead of an integer). IoT-TEG 4.0 could generate the expected event type changing the type of one or more event attributes. For the integration testing, IoT-TEG 4.0 generates test events like the ones that the I4.0 to be connected device will be sending. The mentioned type of test may be done using a real-time testing or not, but the stress test, in the I4.0 context, should be performed in real-time. IoT-TEG 4.0 can generate a huge and large number of events to simulate different connected I4.0 devices. Moreover, these events can be generated with a short period of time and with a short delay, so the I4.0 system under test can be stressed.

VIII. CONCLUSION

In this article, we presented IoT-TEG 4.0, a system that integrates the events 4.0 definition and a graphical interface that eases event generation. The obtained test events 4.0 allow us to be more accurate while testing IoT systems. They can mimic the data that enriches Industry 4.0, such as events that are distributed over time, different types of event, events built with the injection of other values, and so on. IoT-TEG 4.0 covers a wider number of use cases and different types of tests.

Future work: IoT-TEG 4.0 generates time-distributed events and sends them to a queue using the MQTT protocol. Then, the software tested has to wait for the event generation. Future work involves using discrete event simulation (DES) to generate test cases supported by a simulation model. DES would be managed by a controlled clock in which the state variables of the model evolves over time [47]. The DES-based event generator would allow to implement the following features.

1) *Aperiodical events*. Only at an specific date or randomly within an interval of time scheduled each week, month, i.e.: every Monday among 15:00 and 18:00 CET generate an instance of TemperatureEventType.
2) Test a system with a set of events previously generated.
3) Model an IoT system with different types of events emitted by several devices to use it as fake input. The generated data provided by IoT-TEG 4.0 would allow the development of other system without deploying the whole physical IoT platform.

In addition, the model could be defined with a graphical tool as it has been done in the work by Boubeta *et al.* [32]. To achieve

the mentioned approach, the first step is to develop the meta-model of the possible event types IoT-TEG 4.0 offers. Then, an approach to do it is to use the eclipse modeling framework [48] to build a meta-model and corresponding graphic tool such as the work previously mentioned.

APPENDIX

APPENDIX A DATASETS

This[36] is a dataset repositorythat contains fall simulation data, fall analysis, IoT-TEG 4.0 generated test events, a JAVA simulator with the EPL query to detect FAW falls, and fall simulation video clips.

REFERENCES

[1] L. Gutiérrez-Madroñal, I. Medina-Bulo, and J. Domínguez-Jiménez, "IoT-TEG: Test event generator system," *J. Syst. Softw.*, vol. 137, pp. 784–803, 2018.

[2] S. Haller, S. Karnouskos, and C. Schroth, "The internet of things in an enterprise context," in *Future Internet - FIS*2008, J. Domingue, D. Fensel, and P. Traverso, Eds. Berlin, Heidelberg: Springer, 2009, pp. 14–28.

[3] L. Barreto, A. Amaral, and T. Pereira, "Industry 4.0 implications in logistics: An overview," *Procedia Manuf.*, vol. 13, pp. 1245–1252, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978917306807

[4] E. Hofmann and M. Rüsch, "Industry 4.0 and the current status as well as future prospects on logistics," *Comput. Ind.*, vol. 89, pp. 23–34, Aug. 2017.

[5] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *J. Ind. Inf. Integration*, vol. 6, pp. 1–10, 2017.

[6] M. Rüßmann *et al.*, "Industry 4.0: The future of productivity and growth in manufacturing industries," *Boston Consulting Group*, vol. 9, no. 1, pp. 54–89, 2015.

[7] T. Glock, B. Sillman, M. Kobold, S. Rebmann, and E. Sax, "Model-based validation and testing of industry 4.0 plants," in *Proc. Annu. IEEE Int. Syst. Conf.*, 2018, pp. 1–8.

[8] S. Vaidya, P. Ambad, and S. Bhosle, "Industry 4.0 - a glimpse," *Procedia Manuf.*, vol. 20, pp. 233–238, 2018.

[9] K.-D. Thoben, S. Wiesner, and T. Wuest, ""industrie 4.0" and smart manufacturing - a review of research issues and application examples," *Int. J. Automat. Technol.*, vol. 11, pp. 4–19, Jan. 2017.

[10] H. Foidl and M. Felderer, "Data science challenges to improve quality assurance of internet of things applications," in *Proc. Int. Symp. Leveraging Appl. Formal Methods*, Springer, 2016, pp. 707–726.

[11] L. Gutiérrez-Madroñal, A. García-Domínguez, and I. Medina-Bulo, "Evolutionary mutation testing for IoT with recorded and generated events," *Softw., Pract. Experience*, vol. 49, no. 4, pp. 640–672, 2019.

[12] D. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. New York, NY, USA: Wiley, 2011. [Online]. Available: https://books.google.es/books?id=TTePtrngMTIC

[13] D. Luckham, "A view of the current state of event processing," 2006. Accessed: Jun. 2021. [Online]. Available: http://complexevents.com/wp-content/uploads/2006/03/dluckham-workshop-03-2006.pdf

[14] J. Schiefer, S. Rozsnyai, C. Rauscher, and G. Saurer, "Event-driven rules for sensing and responding to business situations," in *Proc. Inaugural Int. Conf. Distrib. Event-Based Syst.*, 2007, vol. 233, pp. 198–205.

[15] M. Völker, S. Mandal, and M. Hewelt, "Testing event-driven applications with automatically generated events," in *BPM*, 2017.

[16] Xively, "Xively Website." Accessed: Jun. 2021. [Online]. Available: https://cloud.google.com/solutions/iot

[17] Buglabs, "Buglabs Website." Accessed: Jun. 2021. [Online]. Available: https://www.buglabs.net/

[18] Grovestreams, "Grovestreams Website." Accessed: Jun. 2021. [Online]. Available: https://grovestreams.com/index.html

[19] Sensorcloud, "Sensorcloud Website," Accessed Jun. 2021. [Online]. Available: http://sensorcloud.com/

[20] ThingSpeak, "Thingspeak Website." Accessed: Jun. 2021. [Online]. Available: https://thingspeak.com

[21] Zettajs, "Zettajs Website." Accessed: Jun. 2021. [Online]. Available: http://www.zettajs.org/

[22] L. Gutiérrez-Madroñal, L. La Blunda, M. F. Wagner, and I. Medina-Bulo, "Test event generation for a fall-detection IoT system," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6642–6651, Aug. 2019.

[23] D. C. Luckham and B. Frasca, "Complex event processing in distributed systems," Computer Systems Laboratory, Stanford University, Stanford, CA, USA, Technical Report CSL-TR-98-754, 1998.

[24] S. Ahn and D. Kim, "Proactive context-aware sensor networks," in *Proc. Eur. Workshop Wirel. Sensor Netw.*, Springer2006, pp. 38–53.

[25] D. Robins, "Complex Event Processing," in *Proc. 2nd Int. Workshop Educ. Technol. Comput. Sci.*, Citeseer, 2010, pp. 1–10.

[26] Oracle, "Java SE8 ScheduledThreadPoolExecutor." Accessed: Jun. 2021. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html

[27] Eclipse, "Mosquitto." Accessed: Jun. 2021. [Online]. Available: https://mosquitto.org

[28] OASIS, "MQTTStandart." Accessed: Jun. 2021. [Online]. Available: http://mqtt.org/

[29] EsperTech, "Self-service terminal." Accessed: Jun. 2021 [Online]. Available: http://esper.espertech.com/release-5.4.0/esper-reference/html/examples.html#examples-terminalsvc-J2EE

[30] EsperTech, "Transaction." Accessed: Jun. 2021 [Online]. Available: http://esper.espertech.com/release-5.4.0/esper-reference/html/examples.html#examples-transaction-3-event-challenge

[31] R. Gad, "Event-driven principles and complex event processing for self-adaptive network analysis and surveillance systems," Ph.D. dissertation, Univ. Appl. Sci. Frankfurt am Main, Frankfurt, Germany, 2015.

[32] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "MEdit4CEP: A Model-driven solution for real-time decision making in soa 2.0," *Knowl.-Based Syst.*, vol. 89, pp. 97–112, 2015.

[33] A. Milne, "Esper demo nuclear." Accessed: Jun. 2021. [Online]. Available: https://github.com/adrianmilne/esper-demo-nuclear

[34] EsperTech, "Esper." Accessed: Jun. 2021. [Online]. Available: http://www.espertech.com/esper/

[35] L. La Blunda, L. Gutiérrez-Madroñal, M. F. Wagner, and I. Medina-Bulo, "A wearable fall detection system based on body area networks," *IEEE Access J.*, vol. 8, pp. 193060–193074, 2020.

[36] L. Gutiérrez-Madroñal, "Gitlab Repository - Fall Detection," Accessed: Jun. 2021. [Online]. Available: https://gitlab.com/lorgut/fall-events-validation.

[37] B. P. T. Group, "UNICORN." Accessed: Jun. 2021. [Online]. Available: https://bptlab.github.io/Unicorn/

[38] Micro-Research_Finland_Oy, "Timing System." Accessed: Jun. 2021. [Online]. Available: http://www.mrf.fi/index.php/timing-system

[39] Starcom_Systems, "The event generator." Accessed: Jun. 2021. [Online]. Available: https://www.starcomsystems.com/2012/03/26/the-event-generator/

[40] Oracle, "Oracle web logic integration." Accessed: Jun. 2021. [Online]. Available: https://www.oracle.com/middleware/technologies/weblogic-integration.html#learning

[41] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental internet of things research," *Commun. Mag., Inst. Elect. Electron. Engineers*, vol. 49, no. 11, pp. 58–67, 2011.

[42] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester, "The w-ilab.t testbed," in *Testbeds and Research Infrastructures. Development of Networks and Communities*. New York, NY, USA: Springer, 2011, pp. 145–154.

[43] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, Association for Computing Machinery, 2013, pp. 153–165.

[44] R. Trüb *et al.*, "Flocklab 2: Multi-modal testing and validation for wireless Iot," Zurich: ETH Zurich, Comput. Eng. Netw. Lab., 2020.

[45] C. Burin des Roziers *et al.*, "Using senslab as a first class scientific tool for large scale wireless sensor network experiments," in *Proc. Int. Conf. Res. Netw.*, Springer, 2011, pp. 147–159.

[46] A. Arora, E. Ertin, R. Ramnath, M. Nesterenko, and W. Leal, "Kansei: A high-fidelity sensing testbed," *IEEE Internet Comput.*, vol. 10, no. 2, pp. 35–47, Mar./Apr. 2006.

[47] A. M. Law and W. D. Kelton, *Simulation Modeling & Analysis*, *2nd ed. Singapore*: McGraw-Hill, 1991.

[48] E. Foundation, "Eclipse modeling framework," Accessed: Jun. 2021. [Online]. Available: https://www.eclipse.org/modeling/emf/

Antonio Velez-Estevez received the bachelor's degree in computer science and the master's degree in research in systems and computer engineering in 2019 and 2020, respectively, from the University of Cadiz, Cádiz, Spain, where he is currently working toward the Ph.D. degree.

He is a R&D Support and Management Technician with the University of Cadiz. His main interests include data science, scientometrics, artificial intelligence, and software engineering.

Mr. Velez-Estevez is member of the IntellSOK Intelligent Social Knowledge Based Systems research group.

Lorena Gutierrez-Madroñal received the first-class honor's degree in computer systems management, and the B.Sc., M.A.S., and Ph.D. degrees in computer science from the University of Cádiz, Cádiz, Spain, in 2007, 2009, 2010, and 2017, respectively.

She has been working with the Department of Computer Science and Engineering as a Full Time Lecturer since 2009. To prove the usability of the test-generated events, she uses them to apply mutation testing to EPL query languages, such as the event processing language. She has participated in research projects with software-engineering-related aspects. She was in programs and organizing committees at different conferences. She is a Researcher of the UCASE Software Engineering Research Group. Her research is focused on the Internet of Things and test event generation for any event-processing program.

Inmaculada Medina-Bulo (Member, IEEE) received the Ph.D. degree in computer science from the University of Seville, Seville, Spain, in 2003.

She has been an Associate Professor with the Department of Computer Science and Engineering, University of Cádiz, Spain, since 1999. She has been a member of the Council of the School of Engineering as well as a Socrates/Erasmus Program Coordinator for several years. From July 2010 to 2011, she was appointed Degree Coordinator for the Computer Science Studies and a member of the Board of the ESI. From September 2013 to July 2019, she was Chief Information Officer of the university. Her research was supported by research stays in the USA, the U.K., and Germany. She has served in program or organizing committees at different conferences and for differently journals. She has coordinated the development of several open-source testing tools, such as the MuBPEL mutation testing tool for WS-BPEL, the GAmeraHOM tool for locating "hard-to-kill" mutants, the Rodan test case generation tool and the Takuan dynamic invariant generator for WS-BPEL. She has participated in and led research projects all involved in software-engineering-related aspects. She has authored and coauthored numerous papers in international journals, and international conferences and workshop proceedings. She is the main Researcher of the UCASE Software Engineering Research Group. Her main research interests include software verification, software testing, web service compositions, model-driven engineering, and complex event processing.