Knowledge-Based Systems 213 (2021) 106682

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

MEdit4CEP-SP: A model-driven solution to improve decision-making through user-friendly management and real-time processing of heterogeneous data streams

David Corral-Plaza *, Guadalupe Ortiz, Inmaculada Medina-Bulo, Juan Boubeta-Puig

UCASE Software Engineering Research Group, Department of Computer Science and Engineering, University of Cadiz, Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Cádiz, Spain

ARTICLE INFO

Article history:
Received 13 April 2020
Received in revised form 10 December 2020
Accepted 11 December 2020
Available online xxxx

Keywords:
Model-Driven Development
Stream Processing
Complex Event Processing
Heterogeneous data
Decision-making

ABSTRACT

Organisations today are constantly consuming and processing huge amounts of data. Such datasets are often heterogeneous, making it difficult to work with them quickly and easily due to their format constraints or their disparate data structures. Therefore, being able to efficiently and intuitively work with such data to analyse them in real time to detect situations of interest as quickly as possible is a great competitive advantage for companies. Existing approaches have tried to address this issue by providing users with analytics or modelling tools in an isolated way, but not combining them as a one-in-all solution. In order to fill this gap, we present MEdit4CEP-SP, a model-driven system that integrates Stream Processing (SP) and Complex Event Processing (CEP) technologies for consuming, processing and analysing heterogeneous data in real time. It provides domain experts with a graphical editor that allows them to infer and define heterogeneous data domains, while also modelling, in a user-friendly way, the situations of interest to be detected in such domains. These graphical definitions are then automatically transformed into code, which is deployed in the processing system at runtime. The alerts detected by the system, in real-time, allow users to react as quickly as possible, thus improving the decision-making process. Additionally, MEdit4CEP-SP provides persistence, storing these definitions in a NoSQL database to permit their reuse by other instances of the system. Further benefits of this system are evaluated and compared with other existing approaches in this paper.

1. Introduction

In the past few years, the volume of the data that organisations have to store, process and analyse has increased considerably. Moreover, the sources which produce these data are heterogeneous, which means that these huge amounts of data are represented in a wide range of data structures. These two problems lead us to the term Big Data, which is grounded in the three V's [1]: the first, Volume, refers to the amount of data generated; the second, Velocity, is related to the speed at which the data are generated; and the third, Variety, encompasses the heterogeneity of these data. Being able to work with these large and heterogeneous datasets, performing specific domain analytics and reacting in real time to situations of interest gives a company a notable competitive advantage. Hence, there is a need for any kind of user, regardless of their computer science knowledge, to be able to operate, in a user-friendly way, with these heterogeneous data, consuming, processing and analysing them.

In this context, Stream Processing (SP) is a paradigm that allows us to consume streams of data and perform transformations on them. This means huge amounts of heterogeneous data can be processed and transformed in order to make them ready for analytics. Once these large streams of heterogeneous data are ready to be used, Complex Event Processing (CEP) comes into play. CEP is a technology that provides the capabilities of analysing data streams, correlating them and detecting real-time situations of interest, previously defined using a specific syntax. In a previous work [2], we demonstrated the benefits of combining these two technologies, resulting in an SP architecture for heterogeneous data in the Internet of Things (IoT).

Despite the great advantages of SP combined with CEP, the complexity of these technologies is a major issue for users who are not computer scientists. Thus, from our perspective, the users, who are experts in the domain, should be able to focus on the definition of the problem and not on the implementation of these technologies. In order to respond to this challenge, a graphical tool should be provided that allows such users to ignore the

* Correspondence to: School of Engineering, Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Cádiz, Spain.
E-mail addresses: david.corral@uca.es (D. Corral-Plaza), guadalupe.ortiz@uca.es (G. Ortiz), inmaculada.medina@uca.es (I. Medina-Bulo), juan.boubeta@uca.es (J. Boubeta-Puig).

https://doi.org/10.1016/j.knosys.2020.106682
0950-7051/

technical issues so they can focus on specifying the problems of the domain in which they are experts. Some of the authors of this paper already proposed MEdit4CEP [3], a model-driven approach which facilitates CEP use for non-technological users. More specifically, MEdit4CEP proposes a graphical editor that provides non-technological users with the capabilities to define CEP domains and event patterns – situations of interest to be detected – in a drag and drop canvas. In addition, the graphical editor itself is capable of automatically transforming those definitions into deployable code.

We have thus detected three challenges to be addressed. First, due to the variety of existing datasets, we have to be able to consume and process heterogeneous data in real time. Second, we have to be able to model these heterogeneous data in order to detect situations of interest, also in real time. Finally, we have to provide domain expert users with a set of tools that provides them with both functionalities in a friendly and intuitive way.

Let us illustrate these three problems in the smart water management domain. Current water management organisations often operate in large areas where smart water metres monitor several values of the water currently running through their pipes. It is worth noting that more than one kind of smart water device can be found in these areas, so the information generated by these devices is mostly heterogeneous. In such a scenario, these organisations may wish to analyse large amounts of heterogeneous information in real time, to detect anomalies in their networks, such as leaks or disruptive consumption. Thus, companies need a system or tools that allow them: (1) to define the heterogeneous models of data to be analysed, and (2) to describe the anomalies they aim to detect within their data in real time, in a friendly and intuitive way. As a result, companies would see their decision-making improved since they would not have to deal with details of implementation or coding procedures.

The aim of this work is to develop a system capable of consuming and processing heterogeneous data in real time, allowing experts in the field to graphically design these heterogeneous data to detect situations of interest, also in real time, in a friendly and intuitive way, thus improving their decision-making when reacting to these situations. Decision-making is the process by which one action is chosen from among others based on the additional benefits it can provide [4]. Thus, this paper presents three major contributions that address the challenges previously stated and which lead us to the desired system.

Firstly, we present an improved version of the SP architecture [2], which will allow us not only to consume, process and analyse heterogeneous data in IoT domains, but also to include new definitions autonomously (without any kind of human intervention). Secondly, MEdit4CEP has also been enhanced in order to achieve new functionalities, such as persistence of domains and event patterns in a database, generation of a new format for domain definitions, inference of existing domains, and detection of changes in domains and event patterns, among others. Thirdly, a series of adjustments have been also made to these two solutions so they can be integrated and work with each other. We call the result of this integration MEdit4CEP-SP, a novel extension of the original MEdit4CEP editor combined with an updated version of our SP architecture for heterogeneous data, resulting in a completely new system which facilitates decision-making for domain experts. Thanks to MEdit4CEP-SP, a domain expert can now add, update and remove event types and event patterns from the CEP engine embedded in the SP architecture quickly and easily, using the new version of the editor. The evaluation carried out shows that the combination of MEdit4CEP and the SP architecture provides a system that facilitates graphical definition, as well as the process and analysis of heterogeneous data in real time for non-technical users.

Finally, we aim to answer the following research questions:

- (RQ1) How can organisations model, process and analyse their heterogeneous data?
- (RQ2) How important is it to provide organisations with a set of tools that facilitates such tasks and the decision-making process, and for the proposed integration (SP architecture + MEdit4CEP) to be able to provide users with such capabilities?
- (RQ3) Is the proposal more effective and efficient than other existing alternatives in achieving the modelling and definition of heterogeneous CEP domains and event patterns in a friendly and intuitive way, as well as their analytics in real time?
- (RQ4) How important is it to be able to add, update and remove existing event types and patterns at runtime and is the proposed integration able to provide users with such capabilities?

The remainder of the paper is organised as follows: Section 2 includes the background to the technologies involved in this work. Section 3 describes the new changes made to the original SP architecture in order to improve it. Section 4 shows the contributions included in the MEdit4CEP editor. Section 5 shows the proposed system, MEdit4CEP-SP, which results from the integration between the new versions of the SP architecture and MEdit4CEP. In Section 6, we illustrate the use of the proposal in the smart water management domain. Section 7 contains the evaluations of the proposed MEdit4CEP-SP system. In Section 8, we review the related works that are similar to our approach, and compare them to our proposal. Section 9 contains the answer to the research questions stated at the beginning. Finally, Section 10 highlights the conclusions achieved in this work and the future research lines.

2. Background

The key technologies and paradigms used in this work are explained in this section.

2.1. Stream processing

For many years, the best solution for sending messages from one point to another in architectures was messaging systems. These systems allow us to send information as messages in a synchronous or asynchronous way. Over time, the architectures became more complex as did the messaging systems and the requirements of these architecture, giving rise to the SP paradigm [5].

SP is a paradigm that allows us to interact with streams of data being transmitted over a channel. SP involves computing data directly as it is produced or received, i.e. on the fly. It was designed to process data in real time. A considerable number of operations can be accomplished with the data that are processed, such as analytics, transformations, MapReduce operations, etc.

Several platforms integrate SP capabilities in their solutions, e.g., WSO2 SI [6], Apache Storm [7] or Apache Kafka Streams [8]. In our proposal, we chose Kafka Streams for several reasons: each message is processed exactly once, is easily embedded in Java and Scala programming languages, and perfectly suits scenarios with low latency requirements. In addition, Kafka Streams works appropriately with Apache Avro, which is a Data-Serialisation System (DSS) allowing data to be quickly and compactly transported. Avro is also compatible with some CEP engines in input event format, which makes the integration between SP and CEP technologies easier.

Using Kafka Streams, we are thus able to consume heterogeneous messages from Kafka topics, perform transformations on such data, and prepare them to be used in the analytics stage.

2.2. Complex event processing

CEP is a well-known technology that allows us to perform stream analytics over events. That is, it processes, analyses, and correlates large amounts of information in the form of simple events, pursuing the real-time detection of previously defined situations of interest.

A simple event is a representation of a change of state in the real world, i.e., measurements from a temperature sensor, WiFi-transmitted packets, open or closed door sensor, etc. In addition, these events are used to feed the CEP engine, while patterns are provided in order to detect situations from among the events received. These patterns are specified using a particular syntax, which depends on the chosen CEP engine. In CEP, unlike other real-time analytics technologies, the events (data) are not stored, but are compared with these patterns in order to check whether the conditions specified are satisfied, meaning that storage resources are not required.

In our previous work, from the alternatives that currently exist for CEP engines, we chose Esper [9]. Esper is one of today's most popular CEP engines on the market since it is open-source. As commented in the previous subsection, it can analyse data represented as Avro events, among others, making integration with Kafka Streams almost immediate. Additionally, the Esper Event Processing Language (EPL) provides a syntax for defining the event patterns (situations of interest) to be detected that is quite similar to SQL. More specifically, EPL provides the SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY clauses, which are well known to SQL developers.

Finally, one of the main advantages of Esper over other CEP solutions is the capability of adding new configurations and definitions at runtime, without having to stop the whole system in order to add, change or remove event types or patterns.

2.3. Model-driven development

Model-Driven Development (MDD) is a software development paradigm that focuses on the main aspects of the system [10], delaying questions of implementation, which are dealt with later. In MDD, a model is a simplified representation of a given domain that helps us better understand it. These models are graphically represented within a canvas and are called diagrams. A model is created by using a Domain-Specific Modelling Language (DSML), which is composed of four key elements: (1) a meta-model, a special kind of model that specifies the abstract syntax of a modelling language [11]; (2) the restrictions that these models have to accomplish in order to be valid; (3) the specific syntax of the DSML that is to be used, a set of graphical elements that will be used to draw the diagrams; and (4) the transformations between models and model to text for the software automation process.

The restrictions and the transformations are the most essential part in the developing of these automatic systems. Regarding restrictions, they are typically specified using Object Constraint Language (OCL), which are limitations to be satisfied by all the models of a given meta-model in order to be valid. Concerning transformations, there are two kinds:

- Model to Model transformation (M2M): *n* source models will be transformed into *m* target models. Source and target models are different.
- Model to Text transformation (M2T): a model is transformed into text, which mainly represents an output that can be used by any other tool.

Finally, when using a MDD tool, the steps are as follows: a user draws the model in the canvas according to the domain itself; the tool validates the model using the restrictions that have to be satisfied in design time; and, depending on the tool, the user has the option to transform the defined model into another model using M2M transformations or into output text, using M2T transformations.

2.4. MEdit4CEP

MEdit4CEP [3] is a model-driven solution designed to provide support for real-time decision-making in Service-Oriented Architectures (SOA) 2.0. Its main goal is to facilitate the definition of domains and situations of interest for domain experts who are untrained in the implementation details. This is achieved through a drag and drop canvas, in which the users graphically define, first, the structure of the information that their system is processing and, second, the event patterns that can be detected from these data.

MEdit4CEP is composed of a model-driven approach for CEP in SOA 2.0: a graphical modelling editor for CEP domain definition, in which event types are modelled with their event properties, a graphical modelling editor for both event pattern and action definition, and an automatic EPL code generator.

MEdit4CEP allows users to ignore all implementation details, because once the definitions have been modelled, they are then transformed into code, using M2T transformations. Thanks to this approach, programming knowledge is completely unnecessary.

In Fig. 1, we can see how the original proposal works. It is structured in two layers, design time (on the left of the figure), which is composed of the MEdit4CEP editor and the users; and runtime (on the right of the figure), which comprises an Enterprise Service Bus (ESB) application integrated with a CEP engine running inside, and the Data Producers and Data Consumers.

The editor is divided into two parts, one for the domain model definition and another for the event pattern model definitions. In the first one, the domain expert will model all the event types that compose the domain. In the second, the event patterns (situations to be detected) are modelled by the user with the elements available in the palette. When a model is saved, the editor checks whether this model conforms to the ModeL4CEP meta-model proposed in [12], i.e. whether it is syntactically correct. If there are no problems within the model, it is automatically transformed into code using M2T transformations. The result of this process is EPL code, which is ready to be deployed in the Esper CEP engine. It is worth mentioning that during the whole process of modelling, definition and validation, the EPL technical aspects are hidden from the users and the resulting EPL code is free of syntactic errors.

This editor was implemented using Eclipse Epsilon [13], which is a project working out of the box with Eclipse Modelling Framework (EMF). In particular, it provides a family of languages for model validation (Epsilon Validation Language, EVL), M2M transformation (Epsilon Transformation Language, ETL) and M2T transformation (Epsilon Generation Language, EGL). Additionally, Epsilon provides a tool for graphical model editor creation (EuGENia), which is a front-end for Graphical Modelling Framework (GMF).

Since MEdit4CEP was designed following the principles of modularity and adaptability, it has already been appropriately extended to deal with other paradigms, such as Petri nets and gamification. As an example, MEdit4CEP-CPN [14] extends MEdit4CEP with a Prioritised Coloured Petri Net formalism to support the modelling, simulation, analysis and both syntactic and semantic validation of CEP-based systems. In addition, MEdit4CEP-Gam [15] is an extension of MEdit4CEP for defining gamification domains as well as designing gamification strategies, monitoring them, and automatically transforming these strategies into code to be deployed in CEP-based systems.

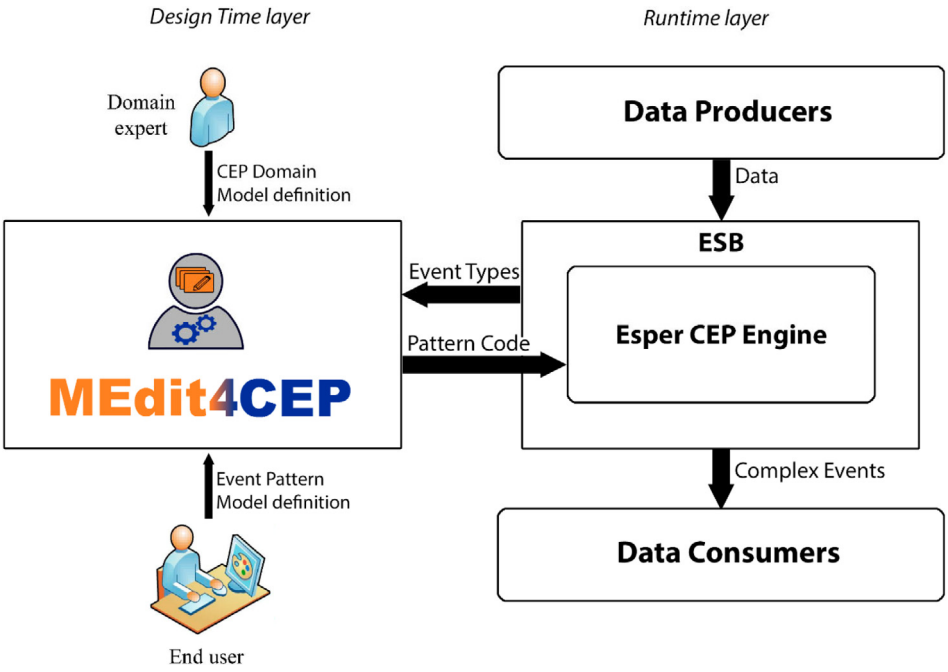

Fig. 1. An overview of the MEdit4CEP approach.

3. Heterogeneous IoT data stream processing architecture

In [2], we proposed a SP architecture for heterogeneous data that can be applied to any kind of domain in the IoT field. As previously discussed, the present work enhances this architecture in order to achieve integration with MEdit4CEP. The primary purpose of the SP architecture is to consume, process, transform and analyse heterogeneous data in IoT domains, such as air quality and smart water networks. For such a goal, this architecture combines the SP paradigm with CEP technology.

On the one hand, to implement the SP paradigm, we use Kafka Streams. Kafka Streams is in charge of consuming heterogeneous data from Kafka topics, which are channels through which data travels from one point to another. Then, the Kafka Streams Processors process the heterogeneous data, transforming them into Generic Avro events, which will be sent to the CEP engine to be analysed.

On the other hand, in order to perform CEP analytics, we use Esper. This Esper CEP engine receives the simple events from the Kafka Stream Processors as Avro events and analyses them using the rules that are also included at runtime. Once the conditions specified in one pattern are satisfied, this pattern triggers up and is notified to consumers, who will use this valuable knowledge.

In the previous version of the SP architecture, there were only two topics: one for the inputs (heterogeneous data and event patterns) and another for the output (the alerts detected by the CEP). The main contribution of this new version of the architecture is three new topics added to the Kafka Cluster: "Pattern Deployment" topic, "Event Type Deployment" topic and "Pattern and Event Type Undeployment" topic. Thanks to the introduction of these three new topics, the Data Processors are now in charge of managing not only heterogeneous input data and event patterns, but also event type schema definitions and the deployment or removal of such event types and patterns in real time. The addition of these topics, as well as the coded behaviour within the Data Processors, provides us with the significant advantage of being able to autonomously deploy, update and remove event types and event patterns from the CEP engine embedded in the SP architecture.

The updated architecture is shown in Fig. 2. It maintains the three-layer structure, one for the Data Producers, other for the Data Processing and Analytics, and the last one, which encompasses the Data Consumers.

3.1. Data sources

The Data Sources layer encompasses any kind of device, means or user that communicates with the architecture sending information to a specific Kafka Topic. In this new version, we can distinguish between four types of information:

- **Heterogeneous data:** the data to be processed and analysed in the whole system. In this case, these data will be sent to the Input Data Topic.
- **EPL definitions:** the event patterns to be deployed in the system in order to detect specific situations. These will be sent to the Pattern Deployment Topic.
- **Schema definitions:** the event types definitions to be registered in the CEP engine. They will be sent to the Event Type Deployment Topic.
- **Deployment IDs:** the deployment IDs, which uniquely identify event types and patterns in the CEP engine, to be removed from the CEP engine at runtime. These will be sent to the Pattern and Event Type Undeployment Topic.

Only heterogeneous data sources were considered by our previous version of the architecture. This improved version of the architecture now includes those previously mentioned data types (EPL definitions, event type schema definitions and deployment IDs). As a result of these additions, new EPL patterns and event type schemas can be automatically deployed within the CEP engine, and can also be removed using their deployment IDs. Additionally, such changes facilitate the integration between this SP architecture and MEdit4CEP.

3.2. Data processing

There are two main components in this layer: a Kafka Cluster, which is used to receive information in its Kafka Topics, and a

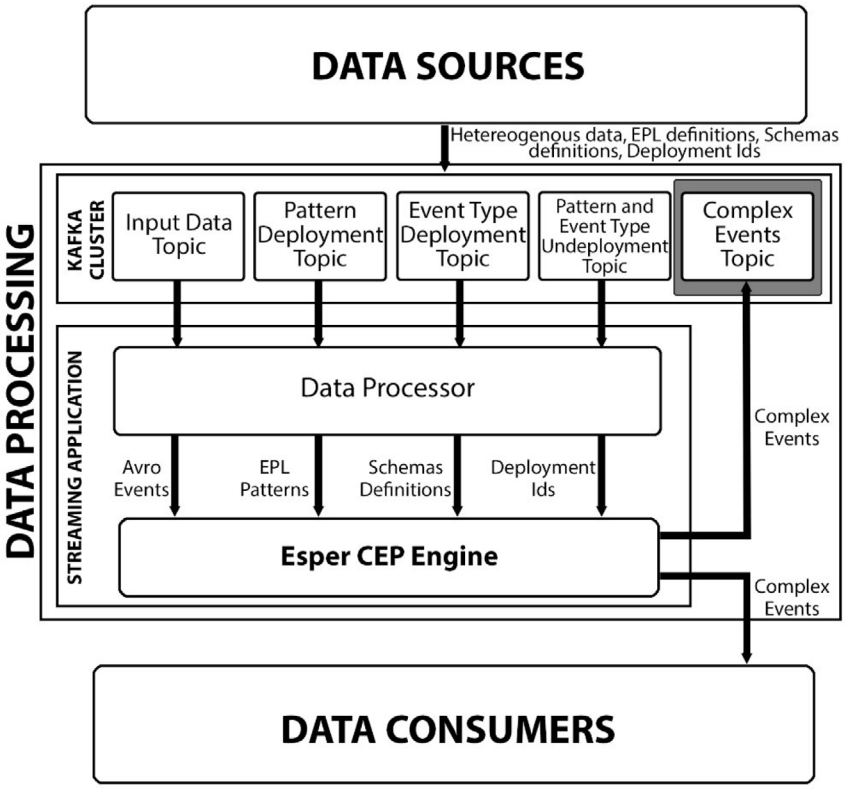

Fig. 2. Stream processing architecture.

Kafka Stream Application in charge of processing the heterogeneous data and analysing them thanks to the embedded CEP. Both are described in more detail below.

3.2.1. Kafka Cluster

The Kafka Cluster contains up to five topics, each with its own purpose: (1) the input data topic, where the heterogeneous data are received; (2) the pattern deployment topic, where the new EPL patterns to be deployed are sent; (3) the event type deployment topic, where the definitions of new event types are collected; and, (4) the event type undeployment topic, which is used to transmit the desired event types and patterns to be removed from the CEP engine. Note that, as shown in Fig. 2, there is a fifth topic, the output topic, where the alerts detected by the CEP engine are published. Despite this topic being located in the Data Processing layer (due to the implementation itself), its nature is to act as Data Consumer.

This Kafka Cluster can be optionally deployed in a different machine from that we are using for our Kafka Stream Application, in order to save processing and storage resources. The topics can have from 1 to N partitions, which result in 1 to N processors. These partitions are used to provide balance and scalability to the solution. Note that having a unique partition, resulting in a single processor receiving and processing 2000 messages per second, is not the same as having two partitions, resulting in two processors, with each one receiving and processing 1000 messages per second. More details about this scalability balance can be found in [16].

In this case, only the input data topic will be partitioned because it will be the one responsible for constantly receiving the information, while the other topics (Pattern deployment, Event Type deployment, Pattern and Event Type undeployment, and Output) will be used at very specific moments, so there is no need to have them partitioned.

3.2.2. Kafka Stream Application

The Kafka Stream Application is the main component of this architecture. Along with the Kafka Processors, we have an embedded Esper CEP engine, which will be responsible for analysing the heterogeneous data that arrives as Avro Events and detecting the situations of interest provided to the CEP engine. These alerts will be published to the output topic and notified to the Data Consumers.

The following tasks are performed in the components of this layer:

- The heterogeneous data, as Kafka messages, are consumed from the input data topic partitions by the Kafka Stream processors associated with that topic. As mentioned, we will have a processor for each input data topic partition, so if the input data topic is partitioned in 8, we will have 8 instances of the data processor.
- The data processors will perform the homogenisation task that will transform these heterogeneous messages into Generic Apache Avro Events.
- The data processors will send the already homogenised information as Avro Events to the Esper CEP engine.
- The Esper CEP engine will analyse and correlate these events with the patterns, which can be also added at runtime. If a pattern triggers up, the complex event detected will be published to the Kafka output topic.
- The information published in the remaining topics (Pattern deployment, Event Type deployment, Pattern and Event Type undeployment, and Output) will also be handled by the data processors in order to deploy new EPL patterns, to deploy new event type schemas and to remove existing event types or patterns, respectively, in real time.

3.3. Data consumers

Finally, the Data Consumers layer is in charge of encompassing any kind of endpoint that can use the information produced or detected by the Esper CEP engine. As an example, a smartphone, a database to store information or a web application to display information could be used as endpoints, which can consume the knowledge generated by the processing layer. Moreover, as previously mentioned, the Output topic from the Kafka Cluster is a Data Consumer because the result of the analytics performed in the CEP engine are sent to that topic to make these complex events available for consumption by any other application.

4. MEdit4CEP improvements

As mentioned, MEdit4CEP is a graphical tool that allows us to define event types and patterns using a drag and drop canvas. Originally, using M2T transformations, MEdit4CEP had the capacity to transform these model definitions into EPL code that can be deployed in an Esper CEP engine. Such a deployment process could be automatised using an ESB application as proposed in the original work, but the use of a Mule Server [17] is no longer the best alternative, as we discuss in the related work section.

In order to achieve the integration between MEdit4CEP and the SP architecture for the previously described heterogeneous data, a series of improvements were performed in the original version. These changes did not affect the original meta-models of the published work, but did impact the source code and the functionalities of the editor itself.

This new extension maintains MEdit4CEP's original capabilities and provides extra functionalities: persistence of the definitions modelled with the editor, definition of Avro schemas, autodetection of CEP domains from databases, and detection of changes in domain and event pattern models, as explained in the following paragraphs.

Firstly, one of the key contributions in this new version of the editor is persistence. Several users of the MEdit4CEP original tool reported that one of the main limitations was the possibility of persistently storing the models. In this updated version, this issue has been resolved: users can now permanently store their models in the cloud and these will then be available for other domain experts interested in reusing them. To this end, a REST API with an embedded NoSQL database has been included with the editor. When a domain or event pattern is saved, it is now automatically stored in the NoSQL database to ensure its persistence. Such a NoSQL database, specifically a MongoDB solution, is in the cloud and available in a NodeJS server that implements a RESTful API. This then receives the event types and patterns to be stored, updated, or removed from the database. It should be noted that MongoDB is schemaless, i.e. it is perfectly suitable to work with heterogeneous domains in which data are not following the same structure. In addition, being schemaless means that the management of the relations between these collections within the database might be struggling, and we thus defined a series of relations between the collections, which are managed within the API (see Fig. 3).

Thanks to this REST API, users can now permanently store their CEP domains and patterns. They can also reuse them in other graphical editor instances, using HTTP GET requests to retrieve them. From a researcher's point of view, this feature is highly time-saving for users and improves decision-making since they do not have to waste time on defining the same event types and event patterns again. This then saves time that might be useful for focusing on the data analytics. Moreover, these definitions are stored in a NoSQL database and accessible for consumption through the REST API, and can therefore be used not only for MDD software but for any other kind of software that might find them useful.

Regarding Avro Schema definitions, these represent the event type structure regardless of the data itself. They are represented in JSON format, so fit perfectly in the MongoDB. These schemas facilitate the integration of heterogeneous data sources, as they allow us to extract the schema of the data being transmitted and transform it into a useable format for the analytics. In order to ensure that the new version of MEdit4CEP is able to work with heterogeneous data sources, we extended the editor's capabilities, providing the capacity to automatically transform the CEP domain models (event types with their properties) into Apache Avro code, which is then registered in the CEP engine. More specifically, we provided the editor with two new capacities:

- Avro EPL definitions: the editor is now able to generate the EPL code that we can use to register a new event type as Avro format in the Esper CEP engine.
- Avro Schema representation: the editor is now able to generate the Avro Schemas from the event types that comprise a CEP domain. Furthermore, the editor is able to read these schemas and transform them into models when retrieving an already defined schema from the API.

Thanks to these new features, our system is now capable of consuming, processing, and emitting information as Avro. Additionally, the latest serialisation strategy and schemas allow for the transportation of large amounts of data in Big Data environments [18].

Concerning the capacity to automatically detect a CEP domain, in the previous version of the editor, event types could be retrieved from the ESB app and then automatically drawn on the canvas. The main drawback was that only event types currently deployed in the CEP engine could be retrieved, i.e. it was impossible to reuse the event types from a domain that was defined in previous usages. Since the definitions are now stored in a cloud database, they can be used by any other instance of MEdit4CEP at any time, regardless of whether or not they are deployed in the CEP engine. Hence, other users can benefit from these stored definitions with just a click in the editor. When clicking on the Autodetection CEP domain option, the editor now sends an HTTP GET request to the REST API to retrieve the existing event types. Such a request will return the Avro Schema representations stored in the NoSQL database. The benefit of this new feature is closely related to those described for the first one: persistence in the NoSQL database.

As mentioned, the editor now has the ability to interpret event types represented in Avro Schemas, meaning that an Avro Schema representation can be transformed into a modelled event type within a CEP domain in the canvas. Thus, the editor will model these already defined event types in the canvas and users can choose to use for their own case study.

In order to illustrate these two functionalities, Avro Schema definitions and CEP Domain Autodetection, we used the editor to model the Air Quality Management domain. This domain is formed by one event type, *AirMeasurement*, shown in Fig. 4. Once the editor has checked the model is valid, it will generate the Avro EPL definition in Listing 1 in order to be able to register this event type as Avro in an Esper CEP engine. At the same time, the editor will send an HTTP POST request to the REST API to permanently store this new event type in the database, and, in that request, the Avro Schema representation (see Listing 2) of the modelled event type is sent in the payload. All three resources represent the same event type but in different ways. This same process could be done upside down; the Avro Schema representation (see Listing 2) could be retrieved from the database using an HTTP

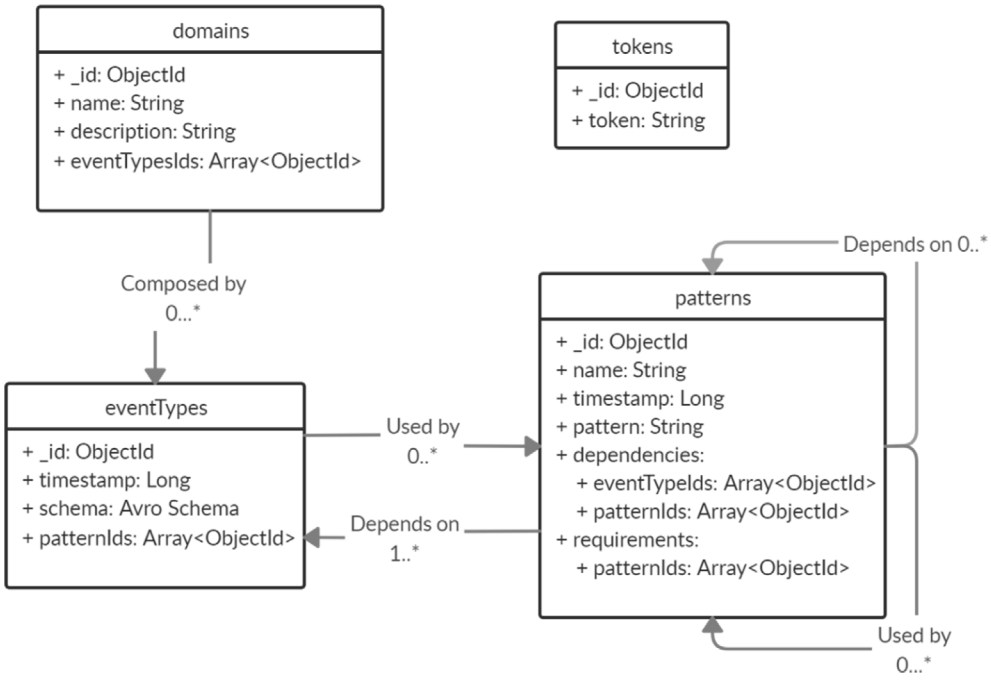

Fig. 3. REST API collections relationships.

GET request, thanks to the Autodetect CEP domain option, and the editor would model it into the canvas, as illustrated in Fig. 4.

Finally, one of the main benefits of Esper compared to other CEP engines is its capability of adding and removing configurations and definitions at runtime. This means that event types and event pattern definitions can be deployed and removed from the CEP engine in real time, without having to stop the execution of the whole system. Thus, the last major contribution added to the editor is the capacity to detect changes in the event types and event pattern models, so these changes can be applied at runtime within the CEP engine and stored in the database.

In CEP, a pattern uses one or more simple event types and/or none or more patterns in order to be defined. This means there could be dependencies between event types and patterns, and between patterns themselves. These dependencies imply that when the root event type or pattern changes (a change could be, for example, that an attribute changes its data type, e.g. from Integer to String), the patterns that depend on it are no longer valid, since this event type or event pattern has changed its behaviour. Therefore, as these patterns are no longer valid, they must be removed from the CEP engine.

In our solution, when an event type or pattern changes, we do remove them (the original and the dependent ones) from the CEP engine but do not completely remove them from the editor; rather we turn them into non-valid patterns that the user must fix in order to make them valid again. Through experience, we have noticed that users prefer to solve these issues manually rather than model the pattern again from scratch. Hence, thanks to this new feature, researchers will be able to add, update, and remove definitions from the CEP engine at runtime, without the having to stop the entire system and redeploy the new configurations. Moreover, as mentioned, if users wish to remove the pattern from the CEP engine but not from the graphical editor, the event pattern will be kept in the editor in a non-valid state, so users can fix it or export it for futures usages.

To conclude, we would like to highlight that we have performed several kinds of optimisations to the algorithms and source codes of the tool itself. Some of these are related to code

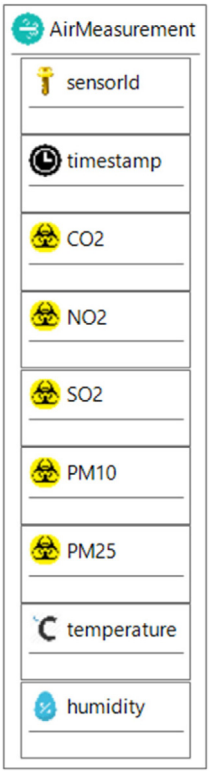

Fig. 4. AirMeasurement event type modelled in the graphical editor.

efficiency, factorisation, and the usage of software design patterns such as the facade pattern. We have also improved the validation and transformations rules available in the original version, which were in charge of ensuring that the models were correct and of transforming the models to deployable EPL code. Now, these validations and transformations are more efficient and effective, providing EPL code which is ready to be used in a CEP engine.

Listing 1 – EPL code generated for creating AirMeasurement event type

```
@public @buseventtype create avro schema AirMeasurement as (sensorId
string, timestamp long, CO2 double, NO2 double, SO2 double, PM10 double,
PM25 double, temperature double, humidity int);
```

Listing 2 – Avro Schema representation generated from the AirMeasurement event type

```
{"schema":{

      "name":"AirMeasurement",
      "type":"record",
      "fields":[

            {"name":"sensorId","type":"string"},
            {"name":"timestamp","type":"long"},
            {"name":"CO2","type":"double"},
            {"name":"NO2","type":"double"},
            {"name":"SO2","type":"double"},
            {"name":"PM10","type":"double"},
            {"name":"PM25","type":"double"},
            {"name":"temperature","type":"double"},
            {"name":"humidity","type":"int"}

      ]

}}
```

All these changes were implemented not only to improve the editor itself, but also to facilitate the communications between MEdit4CEP and the SP architecture. However, new architectures or systems, such as the FIWARE-based one proposed in [19], can easily be integrated with the editor. In this scenario, it would be necessary to create additional M2T rules for generating the code to send the defined event types and patterns through HTTP requests to FIWARE.

5. MEdit4CEP-SP

In this section, we present our proposal, which is the result of integrating the SP architecture with MEdit4CEP. The former allows us to consume, process and analyse huge amounts of heterogeneous information in real time, while the latter allows us to graphically and intuitively define and model CEP domains and event patterns to be detected.

5.1. Proposed architecture

In Sections 3 and 4, respectively, we described the modifications made to the SP architecture and the MEdit4CEP editor, with the aim being not only to enhance them but also to integrate them. Fig. 5 illustrates the complete proposed system, MEdit4CEP-SP, in which we combine both. The system has the following components: the updated version of MEdit4CEP, along with the NodeJS REST API and the NoSQL database, and the modified SP architecture. The integration of such components will provide us with unique features that differentiate our proposal from the other alternatives analysed in Section 8, such as deployment, removal and update of event types and event patterns in real time.

The interactions between these components are represented in Fig. 5 and a brief description follows:

(1) All the communications in which the REST API is involved are HTTP requests and responses that will manage the information stored in the NoSQL database. In this HTTP, communications are sent the event types and patterns that the user wishes to store, update or remove from the database.
(2) The interaction between MEdit4CEP-SP and the SP architecture manages the submission of event types and patterns definitions to be deployed in the CEP engine, as well as the removal and updates of those desired by the user. Such interaction is done through messages received in the Kafka Topics, which are sent from the editor to the SP architecture.
(3) The interactions within the SP architecture, which is receiving the EPL patterns, Event Types definitions and Deployment IDs from MEdit4CEP-SP in their corresponding Kafka topics: Note that these messages are consumed by the Data Processor, processed and sent to the Esper CEP engine.

As mentioned in Section 3, MEdit4CEP-SP keeps all the native capabilities of MEdit4CEP, but some have been improved and new ones have been added. Thanks to these new functionalities of the editor and the changes incorporated in the SP architecture, the integration of the two systems provides us with unique features such as real-time deployment, removal and update of event types and patterns from the CEP engine. Using the proposed system, MEdit4CEP-SP, the user now has full control of the event types and patterns currently deployed in the CEP engine. The situations detected, in real time, by the system allow domain experts to improve their decision-making since they can react instantly.

Firstly, regarding the capacity to add new event types and patterns in real time, when an event type or event pattern is ready to be deployed, MEdit4CEP-SP transforms them into deployable code, which it sends to the SP architecture using the Kafka topics created for that purpose, at runtime, without the need to stop the whole system. On the one hand, if a new event type is defined, it will be transformed into an Avro Schema definition and sent to the Event Type Deployment Topic. On the other hand, if an event pattern is modelled, it will be transformed into EPL code and sent to the Pattern Deployment Topic.

Secondly, in relation to the removal of existing event types and patterns of events in real time, when the event type or event pattern is deleted from the model, MEdit4CEP-SP warns the user

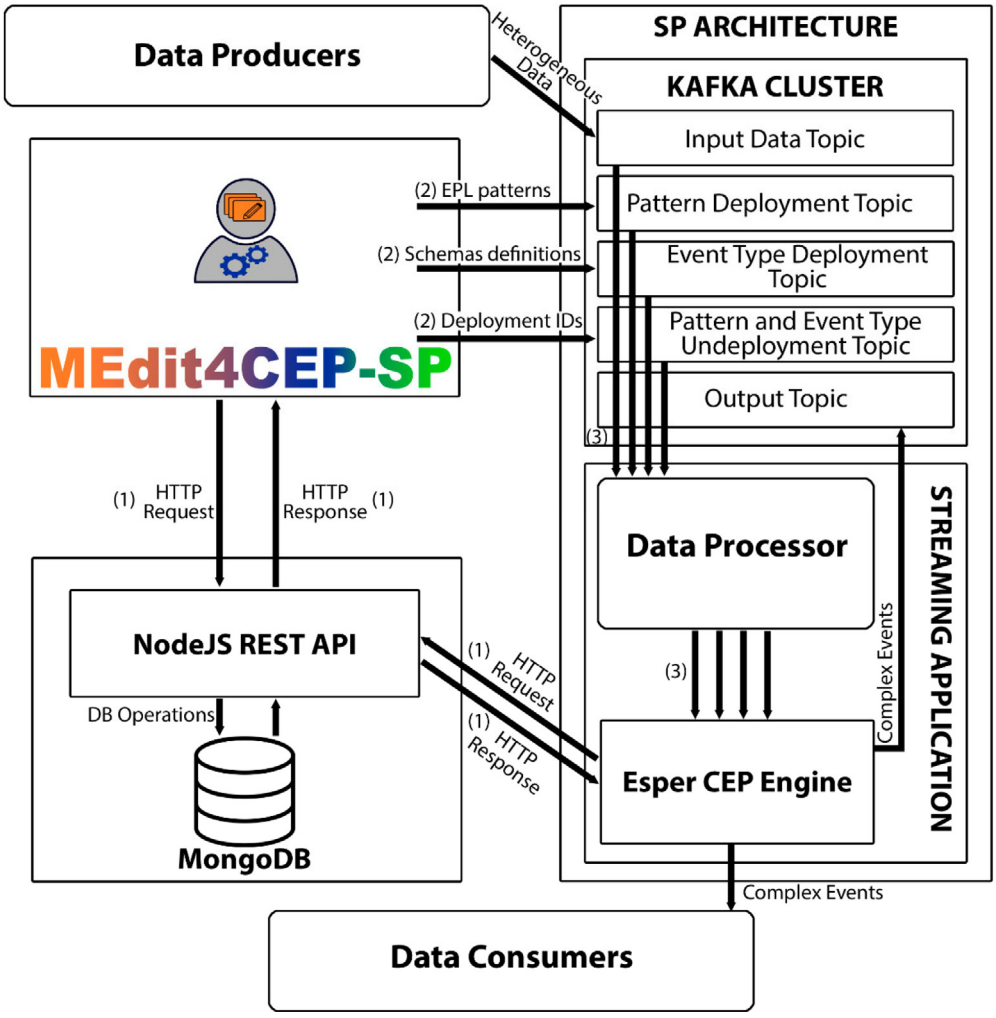

Fig. 5. Proposed integrated architecture, MEdit4CEP-SP.

about the affected event types or patterns that will be removed from the CEP engine due to such change. Then, if the user accepts, such definitions are automatically removed from the CEP engine.

Thirdly, the updating of event types and patterns is in real time. As mentioned, MEdit4CEP-SP is capable of detecting changes in the event types and event patterns in real time. If the user wishes to apply such changes in the CEP engine of the SP architecture, the editor warns the user that these changes will be applied, at runtime, in the CEP engine. Specifically, when updating a definition, whether it is an event type or an event pattern, it first has to be removed and the updated definition will then be deployed in the CEP engine.

Finally, all these changes and functionalities lead us to a collaborative architecture. As the SP architecture can be running in the cloud, several users can be deploying, updating, and removing event types and patterns definitions from the same CEP engine embedded in a shared SP architecture deployed on a cloud-server, with each user using their own instance of the updated MEdit4CEP-SP editor in their own computer.

5.2. MEdit4CEP-SP functionalities

In this subsection, the capabilities resulting from the integration of the SP architecture with the editor MEdit4CEP-SP are explained step by step and illustrated using sequence diagrams.

Deployment of new Event Types and Event Patterns at Runtime

As shown in Fig. 6, the sequence when creating a new domain in MEdit4CEP-SP is as follows:

1. The user models all the event types to be included in the domain.
2. When the user saves the model in the editor, these event type definitions are automatically transformed into Avro Schemas using M2T. They are then sent to the database using POST HTTP requests, so the persistence of that event types is accomplished.
3. Once the event type definitions are correctly saved and stored in the database, the user may want to deploy them into the SP architecture. If so, these event type definitions will be sent to the Event Type Deployment topic, which the SP architecture uses to consume these Avro Schemas and deploy them in the CEP engine. When the event type is correctly deployed in the CEP, the engine generates a unique ID that will be used if the event type must be removed or updated. This unique ID is also saved in the database using a PUT HTTP request.

At the same time, these steps can be replicated in order to create a new event pattern:

1. In the section of the editor designed to create a new event pattern, the user may use the already defined event types

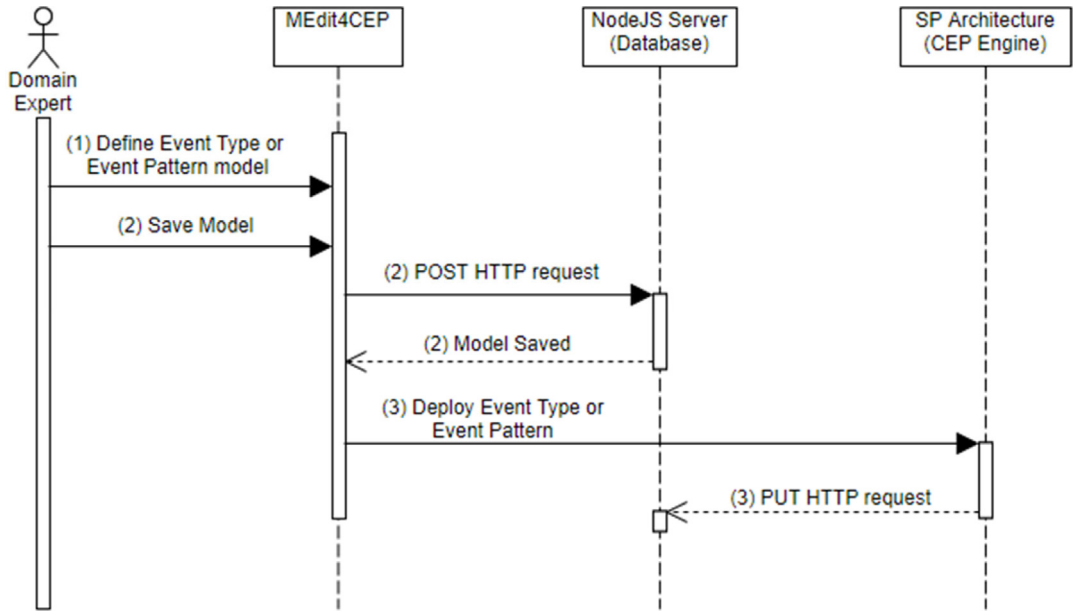

Fig. 6. Creating event type or event pattern sequence diagram.

in order to model the situation to be detected by the engine.
2. When the user saves the event pattern model, they are validated and sent to be stored in the NoSQL database using a POST HTTP request.
3. Once the event patterns definitions are correctly saved, the user may want to deploy them in the SP architecture. If so, these event patterns will be sent to the Pattern Deployment Kafka topic, which the SP architecture uses to consume the EPL patterns to be deployed in the CEP engine. When the event pattern is correctly deployed, a unique ID is generated by the CEP engine. This unique ID is also saved in the database using a PUT HTTP request.

Autodetection of CEP domains

Another contribution of MEdit4CEP-SP is that it can autodetect a CEP domain. Thanks to this functionality, if other users, from completely different computers with another instance of the editor, now wish to reuse some of these simple events already defined by a first user or even by the SP architecture itself, they will be able to retrieve them from the database, which is in the cloud. They can then reuse these already defined domains in order to customise their own. Note that these event type definitions are stored in the database, so they do not necessarily have to be deployed in the engine at that exact moment. The steps to achieve this domain inference are described below and illustrated in Fig. 7:

1. The user clicks on the option to retrieve an already defined domain.
2. The editor sends a GET HTTP request to the server in order to obtain the event type definitions that encompass a specific domain.
3. The NodeJS server receives the request, processes it and returns to the editor the Schema definitions of the event types that are associated to the requested domain.
4. Finally, the editor receives these Schemas, processes them and automatically models them in the canvas, so the user can visualise all these event types that already exist. The user can then choose to deploy these defined event types or can go to the next step and define new event patterns using these retrieved event types.

Removal of existing Event Types and Event Patterns at Runtime

A further incorporation in MEdit4CEP-SP is the capability of removing event types and event patterns at runtime. If some of these definitions are no longer required, they can be removed from the database and undeployed from the CEP engine using this capability. Fig. 8 depicts the required steps for such a goal, which are described below:

1. When an event type or pattern definition is removed from the model or the editor, it warns the user about it. If the user wishes to apply such changes to the event type or event pattern, they have to confirm it in a dialogue.
2. If such changes are accepted, a DELETE HTTP request to the REST API is sent to remove such definitions from the database.
3. Once these definitions have been removed from the database, the HTTP request returns their deployment IDs to the editor, which uniquely identifies these event types or patterns in the CEP engine that have to be removed, as well as the deployment IDs of the event patterns dependent on those that will be undeployed.
4. The user has the option to create a text backup file with the EPL code that will be removed from the CEP engine.
5. The editor sends these deployment IDs to the Pattern and Event Type Undeployment topic in the Kafka Cluster. The SP architecture consumes these IDs from the topic and begins to remove the event types or patterns which are identified by these provided IDs. Note that when an event type or event pattern is removed, all patterns depending on the one to be removed are also removed, in cascade, from the CEP engine. These dependencies are previously considered and these event types and patterns are removed in the appropriate order.

Update of existing Event Types and Event Patterns at Runtime

Finally, MEdit4CEP-SP also provides the functionality of detecting changes in the models and applying them at runtime execution, without the need to stop the whole system. The steps to achieve this are described below and shown in Fig. 9:

1. When the editor detects a change in an event type or event pattern model, it warns the user about it. If the user wishes

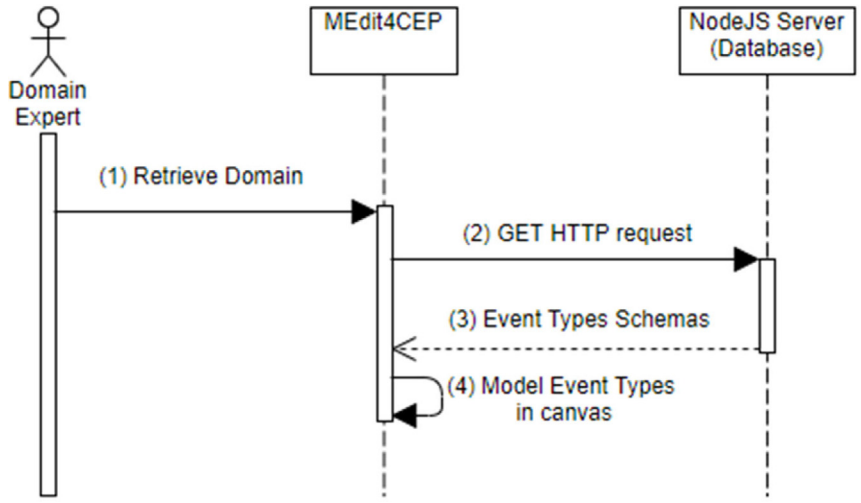

Fig. 7. Autodetect CEP domain sequence diagram.

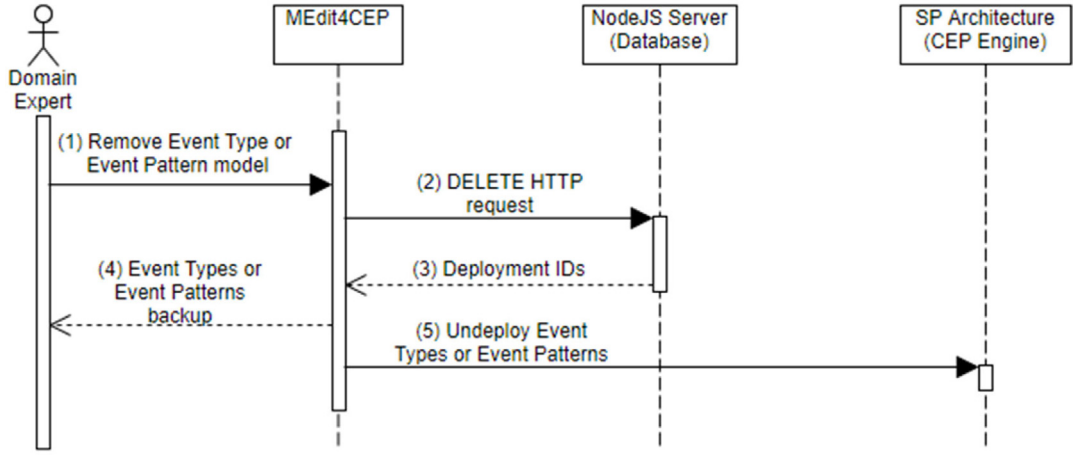

Fig. 8. Removing model sequence diagram.

to apply such changes to the event type or event pattern, they have to confirm it in a dialogue.

2. If the changes are accepted by the user, we first have to determine whether the changes will affect more than one definition. If an event type changes and is then used in an event pattern definition, this pattern is no longer valid due to the event type it uses having changed. Taking this into account, the event types and patterns affected by these changes are removed from the database using a DELETE HTTP request which returns their unique IDs.
3. The user has the option to create a backup with the EPL code that will be removed from the CEP engine.
4. The IDs of the event types and patterns affected are sent from the editor to the SP architecture, for them to be undeployed from the CEP engine.
5. Once all the previous versions of these affected event types and patterns have been removed from the CEP engine, the editor sends the new definitions (the updated ones) to the database using a POST HTTP request to store them.
6. The editor then also sends these updated definitions to the SP architecture, which will consume them and deploy them again in the CEP engine.
7. When the new definitions (simple events types or event patterns) are correctly deployed in the CEP engine, the SP architecture sends a PUT HTTP request to the database in order to set the new deployment ID, which uniquely identifies the updated event type or event pattern.

6. Case study

This section presents the case study in which the proposed solution was tested.

6.1. Case study description

In order to show the usefulness of our proposal, we have chosen the smart water network management domain. According to [20], the main goal of a water network is to supply water from sources to consumers. In smart water networks, the components in charge of such a goal are smart water metres rather than traditional ones. These smart water metres offer new features; for example, they can be read remotely and/or more frequently, while traditional water metres are usually read monthly or bi-monthly by a metre reader [21].

Water is a vital socioeconomic resource and managing it properly within these distributing networks is of great importance. Thanks to the smart water metres that comprise the smart water networks, the companies now have access to huge amounts of data related to the water running through these pipes. Being able to operate with such datasets and to detect currently occurring

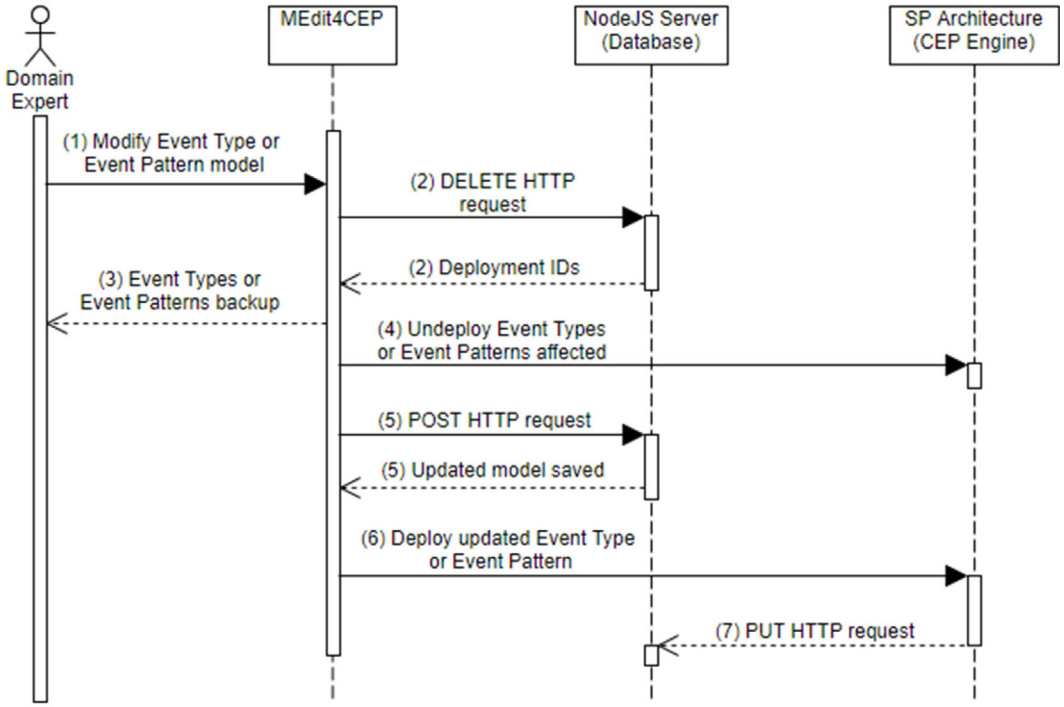

Fig. 9. Updating model sequence diagram.

situations in the water network is essential. One of the most common situations that all companies need to detect is leaks within their distribution networks, which cause excessive customer billing [22]. Moreover, being able to ensure that the smart water sensors are working properly would also avoid billing problems [23].

In [2] we presented *Grupo Energético de Puerto Real S.A.* (GEN) [24], the local company in charge of the water and electricity supply in the city of Puerto Real (Spain). In this previous work, we successfully integrated the data produced by their water distribution network into the SP architecture, with the aim of detecting critical situations, following the company's requirements.

The information provided by GEN is extracted from 111 smart water metres. From each water metre reading, we can obtain the following values:

- **serialNumber:** the serial number of the water metre.
- **dateTime:** date and time of the water metre reading.
- **volumeM3:** the volume of water detected by the water metre in cubic metres.
- **volumeL:** the volume of water detected by the water metre in litres.
- **type:** the use of the water metre (domestic, industrial, etc.).
- **starts:** number of times that the water metre has detected water passing through.
- **batteryLevel:** battery level of the water metre as Integer.
- **batteryLevelStr:** battery level of the water metre as String.
- **sleepingTime:** time in seconds without consumption.
- **leakingTime:** time in seconds from which the water metre detects an unusual or low consumption.
- **normalTime:** time in seconds that the water metre works without problems.

The CEP domain and event patterns that we manually defined using EPL, in the evaluation of the previous work, will be modelled using MEdit4CEP-SP and automatically deployed, removed or updated in the CEP engine, in order to prove the correctness of the proposed integrated solution. The purposes of these patterns are:

- **Reading errors**: these happen when one or more of the sensors of a specific smart water metre are not working properly, i.e., negative values in water consumption.
- **Water leaks**: these happen when the metre detects no start but there is consumption.
- **Unusual consumption**: this happens when the consumption value is unusual, i.e., zero consumption over a period of time.

6.2. Case study implementation

In [2] we evaluated the data provided by GEN in order to detect the three patterns described above. The results of the experiments were successful, being able to detect the defined anomalies in real time. In addition, thanks to this evaluation, the company was able to detect and correct problems in some of the readings of their water metres. It should be noted that the company was unaware of these problems.

Previously, we manually defined such patterns using EPL in order to test the SP architecture. We will now model the domain and the event patterns using MEdit4CEP-SP. As mentioned, with MEdit4CEP-SP, the domain experts are able to graphically model and design such domains and patterns, without knowing the EPL syntax.

For this case study, the domain to be modelled is the *Water* domain, which encompasses the *WaterMeasurement* event type, containing all the properties included by a water-metre reading, as well as the *ReadingErrors*, *WaterLeaks* and *UnusualConsumption* event patterns. This event type and the event patterns are modelled using MEdit4CEP-SP and automatically deployed in the CEP engine running within the SP architecture.

In the following sub-subsections, the implementation of the case study is presented. First, in Section 6.2.1, the management of the received data along with the CEP domain definition is presented. Second, in Section 6.2.2, the CEP domain model is validated, then stored in the NoSQL database and finally deployed in the SP architecture. Third, in Section 6.2.3, the event patterns to be detected by the CEP engine are modelled. Fourth,

in Section 6.2.4, these modelled patterns are validated, sent to the database and finally deployed in the SP architecture. Finally, in Section 6.2.5, an example of updating and removing event patterns in execution time is shown.

6.2.1. Data reception and CEP domain definition

Concerning data acquisition, the only requirement is that the information provided by the company must be sent to the Kafka Input Data topic. Once the data are received in that topic, there are two alternatives:

- The SP architecture can, at runtime, infer the data structure of the event types that we are processing and assign an event type name. This data structure would be automatically stored in the NoSQL database, so these definitions can be retrieved in the editor using the Autodetect CEP Domain option provided for that in the tool menu.
- The domain expert can use MEdit4CEP-SP in order to manually model the event type that we are about to receive, in this case *WaterMeasurement*: It will then be deployed in the CEP engine and stored in the database.

In the first alternative, the SP architecture consumes the data, infers the schema, registers it in the CEP engine and, finally, sends it to the REST API in order to store it in the NoSQL database. The data consumed by the SP architecture could be heterogeneous, which means there are no restrictions about the data structure, or could be structured (JSON, XML or CSV) or unstructured (raw data). As an example, Listing 3 shows a raw data event to be processed. In this case, the first position indicates the event type name value, namely, *WaterMeasurement*.

Once the raw data is processed, registered in the CEP engine and stored in the database, the domain experts can reuse these already defined schemas in MEdit4CEP-SP through the novel "Autodetect CEP Domain" option. When clicking on the "Autodetect CEP Domain" option on the menu, an HTTP GET request is sent from the editor to the REST API in order to retrieve the existing definitions. Fig. 10(a) shows the modelled event type inferred by the SP architecture. Note that, as we processed raw data (unstructured without any key/header information), the property names are generic (p1, p2, p3, ...).

In the second alternative, the domain experts may wish to model these event types in advance, in this case, the *WaterMeasurement* event type. Once the event type has been modelled and registered in the CEP engine, when the data related to this event type are received in the SP architecture, the CEP engine will already know about them, with it thus being unnecessary to infer its data structure at runtime. The domain manually modelled by the domain expert is shown in Fig. 10(b). The main differences between these two models are the property names and the customisation.

In both cases, if the domain expert fits the previously presented case study description, the domain *Water* will contain only one event type, *WaterMeasurement*, with 11 properties for the attributes water metre readings.

6.2.2. CEP domain validation, storage and deployment

Once the CEP domain is modelled, the user may save it. When saving the model, it is first validated to ensure that all the restrictions have been taken into account. If there are no validation errors, the CEP domain and the event types that integrate it are automatically saved in the NoSQL database and are immediately deployed in the SP architecture. The SP architecture will consume, process and deploy such event types in to the CEP engine. In Listing 4, the EPL code automatically generated from the model of Fig. 10(b) is shown.

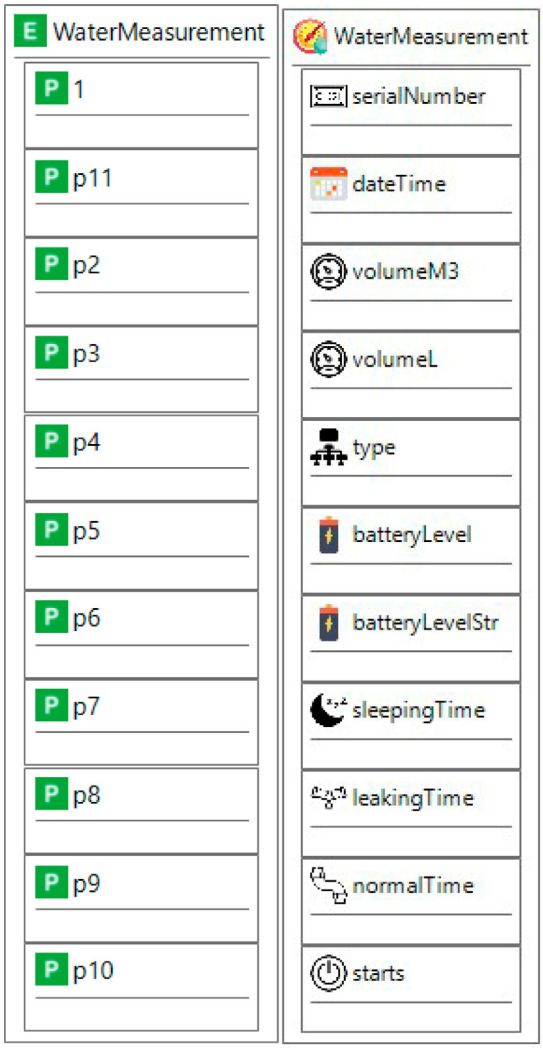

Fig. 10. (a) Water CEP domain automatically inferred by the SP architecture (b) Water CEP domain definition modelled by a domain expert.

6.2.3. Event pattern model definition

The next step is to model the complex situations that our CEP engine should detect using the event types available in the previously defined domain. For this case study, we will model the three previously mentioned event patterns (*AnomalyReadingErrors*, *AnomalyWaterLeaks* and *AnomalyUnusualConsumption*). Fig. 11 illustrates the model of the first event pattern; the remaining figures of the other two event patterns are available at [25].

Reading Error pattern definition

- **Name**: *AnomalyReadingErrors.*
- **Definition**: this pattern allows us to detect problems in one or more of the sensors available within the smart water metre.
- **Pattern conditions**: the value of any of the numeric properties of the event type is negative.
- **New complex event**: *AnomalyReadingErrors*. A new complex event will be created with the same properties of the water metre that presents the anomaly, as well as the current timestamp (in order to know the exact moment at which the anomaly was detected) and a small anomaly message that describes the alert.

Listing 3 – WaterMeasurement event as raw data

```
WaterMeasurement, A87SAA77188, 2018-07-06T05:00:00.000Z, 5.4366397,
5436.6397, INDUSTRIAL, 1048, 3, 6739610, 45816, 47613, HIGH
```

Listing 4 – EPL code generated for creating WaterMeasurement event type

```
create avro schema WaterMeasurement as (serialNumber string, dateTime
string, volumeM3 double, volumeL double, type string, batteryLevel int,
batteryLevelStr string, sleepingTime integer, leakingTime integer,
normalTime integer, starts integer);
```

Water Leak pattern definition

- **Name**: *AnomalyWaterLeaks.*
- **Definition**: this pattern allows us to detect when a leak is happening in a water metre.
- **Pattern condition**: the value of the *leakingTime* property of the event type is positive.
- **New complex event**: *AnomalyWaterLeaks*. A new complex event will be generated with all the details of the affected water metre, as well as the timestamp at which the leak is detected and the description of the anomaly.

Unusual Consumption pattern definition

- **Name**: *AnomalyUnusualConsumption.*
- **Definition**: this pattern allows us to detect anomalies related to the consumption of a smart water-metre.
- **Pattern conditions**: the value of any of the properties related to the consumption of the water metre (*volumeM3* or *volumeL*) is less or equal to 0.
- **New complex event**: *AnomalyUnusualConsumption*. A new complex event will be created with the same properties of the water metre affected as well as the current timestamp and a small message describing the anomaly detected.

6.2.4. Event pattern validation, storage and deployment

Once each of the patterns have been designed and modelled, the user may wish to save them. When saving an event pattern, the editor will automatically check whether the inner restrictions are satisfied. If so, the editor will refresh the palette in order to show this new complex event type just created, so it can be reused by any other event pattern.

At the same time, the pattern is transformed into code using M2T transformations and is stored in the NoSQL database using an HTTP POST request. The user might also want to deploy it in the SP architecture, so the EPL generated previously in the M2T transformation will be sent to the SP architecture, where this EPL pattern definition will be consumed, processed and deployed in the CEP engine.

The EPL code generated by the editor using the M2T transformations, for the very first pattern, is shown in Listing 5; the rest are available in the previously mentioned dataset. First, with the @public keyword, we ensure that this pattern can be used by new event patterns making it public. Second, some information is added to the pattern, such as the name, using @NAME, the description, using @DESCRIPTION, and a tag, using @TAG. Third, using the clause INSERT INTO we define the name of the new event flow (new complex event) that will be created when the pattern triggers up. Fourth, with the SELECT keyword all the properties of the new complex event are defined. Finally, the clause FROM is used to declare the event types that will be used by the pattern, as well as the different conditions that the properties of these event types have to satisfy to detect this pattern.

6.2.5. Event pattern real time updating and removal

As mentioned in previous sections, another capability of MEdit4CEP-SP is to detect changes in the event types or patterns and apply them at runtime within the CEP engine. In order to illustrate this, we have modified the *AnomalyUnusualConsumption* pattern changing the *'less than or equal to'* comparison operator for the *'less than'* comparison operator. When the editor detects changes in the current model, it notifies that to the user, so if the changes are accepted, these will be applied to the CEP engine at runtime.

Additionally, MEdit4CEP-SP is also able to remove existing event types or event patterns from the CEP engine at runtime. The possibility of removing event types or patterns already existed in the editor, but this functionality has now been extended so the event type or pattern model is not just deleted from the disk, but also removed from the NoSQL database and the CEP engine. When the user chooses to delete an event pattern, they can save a backup of the pattern to be deleted, as well as the dependent patterns that will also be deleted along with the selected one. Screenshots of both functionalities (update and remove pattern) can be visualised in the dataset at [25].

7. Evaluation

In this section, we evaluate the proposed system in order to prove that the integration and the improvements in the components are useful and make it easier to define the heterogeneous IoT domains and event patterns to be detected. Furthermore, we aim to demonstrate that MEdit4CEP-SP also facilitates the deployment of such definitions within the SP architecture, in order to detect the situations, previously defined by the users, in real time. We also underline the capability of the system to detect changes in the definitions and apply them at runtime. Moreover, we present an evaluation based on technical metrics to show the effectiveness and scalability of the processing system.

7.1. MEdit4CEP-SP usability

Concerning the graphical editor, the aim is to determine whether users are able to use and understand the functionalities of MEdit4CEP-SP without significant difficulties. For that reason, a group of 28 users, some already experienced in MEdit4CEP, were asked to use MEdit4CEP-SP to define the CEP domain and the event patterns described in Section 6, and then to deploy such definitions.

The steps all these users had to follow to complete the assigned task are the following:

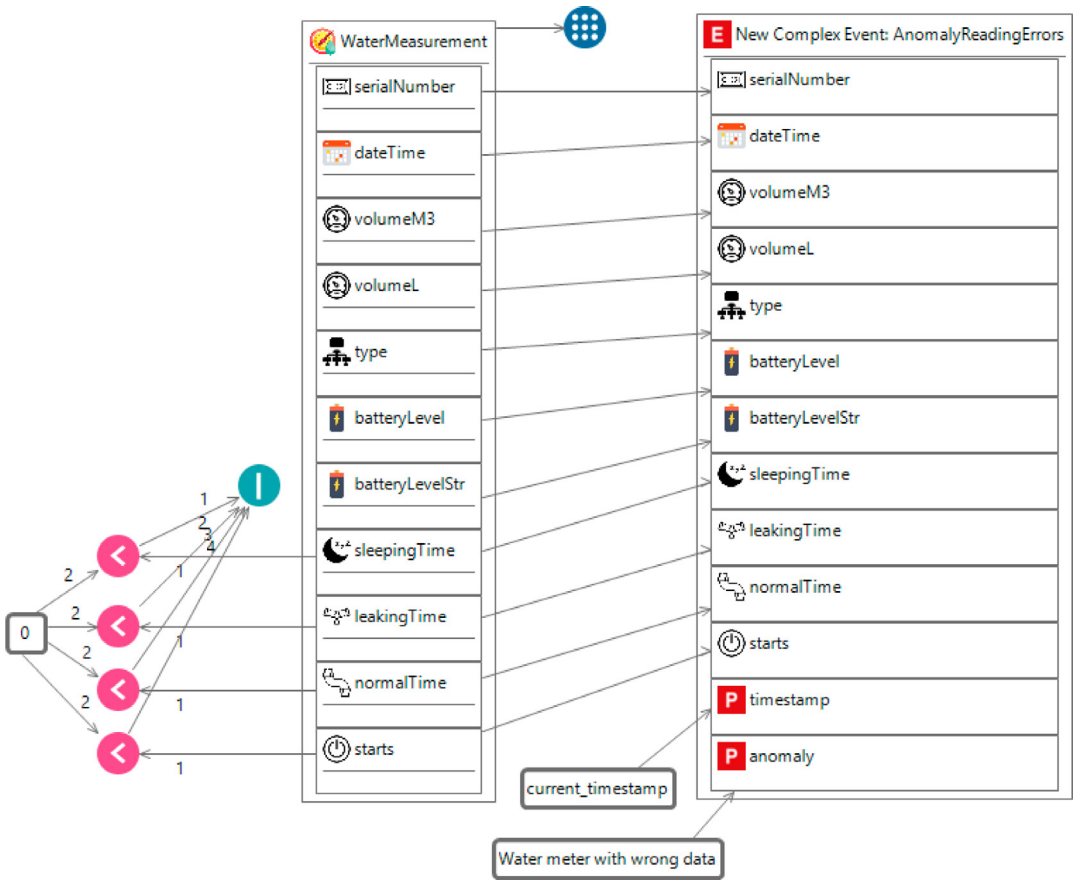

Fig. 11. Anomaly reading errors pattern modelled using the pattern editor.

Listing 5 – EPL code generated for the Anomaly Reading Errors event pattern

```
@public @buseventtype @NAME("AnomalyReadingErrors")
@DESCRIPTION("This pattern allows us to detect problems in one or more
of the sensors available within the smart water-meter")
@TAG(name="domainName", value="Water")
INSERT INTO AnomalyReadingErrors
SELECT a1.serialNumber as serialNumber, a1.dateTime as dateTime,
a1.volumeM3 as volumeM3, a1.volumeL as volumeL, a1.type as type,
a1.batteryLevel as batteryLevel, a1.batteryLevelStr as
batteryLevelStr, a1.sleepingTime as sleepingTime, a1.leakingTime as
leakingTime, a1.normalTime as normalTime, a1.starts as starts,
current_timestamp() as timestamp, 'Water meter with wrong data' as
anomaly
FROM pattern [a1 = WaterMeasurement((a1.sleepingTime < 0 or
a1.leakingTime < 0 or a1.normalTime < 0 or a1.starts < 0))];
```

1. **Modelling the CEP domain** → the users will model the *Water* domain, which is composed of a single event type (*WaterMeasurement*).
2. **Validating, storing and deploying the CEP domain** → once the domain is modelled, the editor will check whether there are no errors within the domain definition, store the model in disk, store the event type definition in the database, and deploy the event type in the SP architecture.
3. **Modelling the event patterns** → the users will model the three anomalies detailed in Section 6 (*Reading Errors*, *Water Leaks*, and *Unusual Consumption*).
4. **Validating and storing the event patterns** → once each pattern model is finished, the editor will check whether there are no problems in the model and will store the pattern in the disk as well as in the database.
5. **Deploying the event patterns** → after each pattern is saved, the users will send them to the SP architecture, where they will be consumed and deployed in the CEP engine.
6. **Updating an existing event pattern** → the users will have to update an existing event pattern in the editor, so this pattern will be automatically updated in the CEP engine as well.
7. **Removing an existing event pattern** → the users will have to remove an existing event pattern in the editor, so this pattern will be deleted from the CEP engine as well.

8. **Removing the CEP domain** → the users will have to remove the existing CEP domain, *Water*, within the editor, so the event type *WaterMeasurement* and the remaining event patterns will be removed from the CEP engine and the REST API.

After completing all the assigned tasks, the participants were asked to complete an 18-question survey to assess the functionalities of MEdit4CEP-SP. These questions were as follows:

- Q1: Do you consider yourself an expert in complex event processing? (No; Yes)
- Q2: How would you like to define event patterns? (Using a graphical editor; Coding the pattern using a programming language; Both using a graphical editor and coding the pattern using a programming language)
- Q3: What type of user is most suitable to use the editor? (Domain experts — programming knowledge not required; Domain experts and programmers; do not know; Others)
- Q4: Is the purpose of the editor clear? (No; A little; Moderately; A lot; Completely)
- Q5: In general, has the editor satisfied your expectations? (No; A little; Moderately; A lot; Completely)
- Q6: Do you consider the functionalities provided by the editor useful? (No; Yes)
- Q7: Do you consider any functionalities provided by the editor irrelevant? (No; Yes)
- Q8: Have you been able to successfully create the required event patterns for a particular domain? (No; A little; Moderately; A lot; Completely)
- Q9: Once the event patterns are graphically defined, is the purpose of these patterns clear? (No; A little; Moderately; A lot; Completely)
- Q10: Do you think the editor could reduce the time needed to define event patterns by using a specific EPL instead of manually implementing them? (No; A little; Moderately; A lot; Completely)
- Q11: How long did you need to perform the given task? (in minutes)
- Q12: How do you rate the speed with which the editor stores and deploys domains and patterns in the CEP engine? (Very bad; Bad; Fair; Good; Excellent)
- Q13: How do you rate the performance of the editor in detecting changes in domains and event patterns that have already been deployed in the CEP engine? (Very bad; Bad; Fair; Good; Excellent)
- Q14: How do you rate the ease of deploying and removing both new and existing domains and event patterns from the CEP engine at runtime? (No; A little; Moderately; A lot; Completely)
- Q15: How do you rate the possibility of exporting event patterns in EPL language when they are removed from the CEP engine? (Very bad; Bad; Fair; Good; Excellent)
- Q16: How do you rate the possibility of reusing, in other editors, the domains and event patterns already defined through a RESTful API? (Very bad; Bad; Fair; Good; Excellent)
- Q17: How do you rate the integration of the editor with the SP architecture and the RESTful API? (Very bad; Bad; Fair; Good; Excellent)
- Q18: Do you have any suggestions for improving the editor? (No; Yes)

The answers of the users were classified into two groups: non-experienced users and experienced users who had already used MEdit4CEP, although not MEdit4CEP-SP. The results from the survey are shown in Table 1.

Analysing the results obtained from the evaluation, it is worth highlighting that most respondents (61%) prefer to use the editor to define the domains and event patterns, rather than coding it using the EPL syntax (see Q2). This is specifically appreciated by the non-experts users (who have not coded EPL patterns before). Furthermore, most of the users (82%) believe that a domain expert is the most suitable profile to make use of the editor (see Q3), which means that one of the purposes of the editor is quite clear: it facilitates CEP and SP to the expert users within the companies, people who know what problems need to be solved but who do not necessarily know how to implement those solutions.

Regarding the editor itself (see Q4–Q10), its features satisfy most of the respondents (75%), and all the participants agreed that the functionalities provided are useful and that none of them are irrelevant or not of any use. Nevertheless, as some of the users pointed out in their questionnaires, these functionalities should be explained within a manual, which is part of our ongoing work.

With regard to the task proposed, almost all respondents (75%) agreed that using the editor made it easier for them to accomplish the task in less time than if they had to code it using a programming language (see Q10); this is one of the main goals when using MEdit4CEP-SP. In addition, if we look at the time required to perform the assignment (see Q11), nearly all the users (97%) were able to define the domain and implement the event patterns in less than one hour; this is a very short time considering that no manual or training was provided prior to the evaluation.

The remaining questions (see Q12–Q17) evaluate the functionalities provided by MEdit4CEP-SP. All the respondents (an average of 78%) valued the editor's capabilities positively, highlighting the possibility of saving the patterns that to be removed from the CEP engine and the API, as well as the capacity of reusing the defined event types and patterns in other editors thanks to the REST API.

Most respondents (75%) were also able to understand the overall architecture behind the editor as well as the integrations that comprise it (see Q17). This is highly important to make them see the work saved by using the proposed architecture instead of other legacy solutions described in Section 8, which do not allow definitions to be automatically deployed, edited and removed at runtime.

Finally, a few improvements (39%) were suggested in the evaluation (see Q18), most of which were related to bugs that have been already corrected; others are in consideration for a future version of the editor. The most widely requested proposal was the possibility of providing a manual with the editor, which makes us think that if the evaluated users were trained before testing the editor, it would make it much easier for them to understand, define, implement, and deploy the domains and event patterns.

7.2. Stream processing architecture performance

Concerning the SP architecture from our proposed system MEdit4CEP-SP, we conducted a thorough evaluation in terms of technical metrics. These tests aimed to determine the maximum number of events per second (e/s) that our processing architecture can handle, and to demonstrate its scalability. The following technical features of the computers used to perform these tests are as follows: Windows 10 Enterprise, 64 bits, Intel® Core™ i7-4770 CPU, 12 GB RAM, and 115 GB HDD.

We simulated several rates of incoming events to detect the maximum incoming event rate that our proposal is able to process, deployed in one and more machines with the features previously described, in different scenarios. To address this, we used Apache JMeter as the data producer for our platform, since this open source Java application is designed for load testing and performance evaluation. Since JMeter does not provide official functionality for publishing events in a Kafka topic, we used it in

Table 1
Results obtained from the questionnaire.

Question	Answer	Group 1 (%)	Group 2 (%)	Total (%)
Q1	No	100	0	75
	Yes	0	100	25
Q2	Using a graphical editor.	36	50	61
	Both using a graphical editor and coding the pattern using a programming language.	64	50	39
Q3	Domain experts — programming knowledge not required	85	66	82
	Domain experts and programmers	5	17	8
	Do not know	10	17	10
Q4	Moderately	32	0	25
	A lot	32	33	32
	Completely	36	67	43
Q5	No	5	0	4
	Moderately	23	17	21
	A lot	45	50	46
	Completely	27	33	29
Q6	Yes	100	100	100
Q7	No	100	100	100
Q8	Moderately	18	17	18
	A lot	27	17	25
	Completely	55	66	57
Q9	Moderately	23	0	18
	A lot	36	33	36
	Completely	41	67	46
Q10	No	9	0	7
	A little	5	0	4
	Moderately	14	17	14
	A lot	45	33	43
	Completely	27	50	32
Q11	10–30 min	63	50	68
	30–60 min	33	50	29
	60–90 min	4	0	3
Q12	Bad	5	0	4
	Fair	22	33	25
	Good	23	17	21
	Excellent	50	50	50
Q13	Bad	5	0	3
	Fair	23	0	18
	Good	27	33	29
	Excellent	45	67	50
Q14	Fair	27	17	25
	Good	23	17	21
	Excellent	50	66	54
Q15	Bad	0	17	3
	Fair	14	0	11
	Good	36	0	29
	Excellent	50	83	57
Q16	Bad	4	0	4
	Fair	14	17	14
	Good	41	17	36
	Excellent	41	66	46
Q17	Fair	27	17	25
	Good	41	33	39
	Excellent	32	50	36
Q18	Yes	36	50	39
	No	64	50	61

conjunction with Kafkameter [26], which is an unofficial plugin developed by the community for this purpose. Furthermore, we used Throughput Shaping Timer [27], another plugin that facilitates customising the number of requests per second we wish to simulate. Thanks to these plugins, we are able to produce a certain number of e/s, and send them to a specific partition of a Kafka Input Data topic.

For these tests, we simulated several scenarios with multiple producers sending events to eight or more partitions of the Kafka Input Data topic. We configured these producers to send heterogeneous events.

In order to test the scalability and the effectiveness of the embedded SP architecture, we performed four test scenarios, each with a different configuration:

- (1) Using 8 partitions, in the Kafka Input Data Topic, deploying the SP architecture in a single computer. We used 7 computers in this test: 2 as JMeter producers, 4 as Kafka Brokers, and 1 for the SP architecture.
- (2) Using 16 partitions, in the Kafka Input Data Topic, deploying the SP architecture in 2 computers. We used 12 computers in this test: 4 as JMeter producers, 6 as Kafka Brokers, and 2 for the SP architecture.

Table 2
Average time required by each task to process an event.

Task name	Mean time (ns)
Data homogenisation	8183 ns
Schema generation	12 966 ns
Creation of the Avro event	2863 ns

- (3) Using 24 partitions, in the Kafka Input Data Topic, deploying the SP architecture in 3 computers. We used 17 computers in this test: 6 as JMeter producers, 8 as Kafka Brokers, and 3 for the SP architecture.
- (4) Using 32 partitions, in the Kafka Input Data Topic, deploying the SP architecture in 4 computers. We used 18 computers in this test: 6 as JMeter producers, 8 as Kafka Brokers, and 4 for the SP architecture.

As described above, we use one machine for each set of eight partitions since the machines used for the testing have eight threads. In this way, each thread of the CPU can deal with a partition of the topic. Once the limit of one machine is reached, we add another one and, at the same time, we incorporate a further 8 partitions for the new machine. We thus seek to show that the solution can scale and handle more data if we incorporate more nodes for processing in a distributed system.

Fig. 12 shows the results of these evaluations. In the *x*-axis, we represent the number of e/s produced by JMeter in the 10 min test, which starts with 210 000 e/s and increases to 870 000 e/s. In the *y*-axis, we represent the total of events produced during each test, which starts with 125 532 960 events and ends up with 521 789 558 events.

As we can see, for the first test scenario, the system can handle the workload without problems until 310 000 e/s. If we increase the e/s rate to 330 000 e/s or above, the system suffers a delay when processing the events instantly, requiring more time to complete the processing. For the second test scenario, we manage to process up to 470 000 e/s, but with higher input rates. As happened before, the system requires more time to process the rest of events. In the third test scenario, the limit is set at 690 000 e/s without problems. Finally, in the fourth test scenario, we are able to process up to 850 000 e/s perfectly; if we increase to 870 000 e/s the system needs a few more seconds to process the remaining events produced, resulting in 511 million events consumed out of 521 million generated.

In addition, we performed a large 60-min simulation, using 4 partitions receiving events at a rate of 150 000 e/s, in order to verify that our SP application of MEdit4CEP-SP does not collapse when executing the simulation for longer execution periods. The simulation was successful; a total of 539 325 935 simple events were processed without problems; i.e., the data was processed in real time without extra time being required to process excess data.

Moreover, in order to measure the time the SP architecture takes to process each event, we performed a further evaluation. We measured the average time, in nanoseconds (ns), that each task takes. In this case, we sought to measure the mean time for each task, not to overwhelm the system. We therefore used 130 000 e/s and 2 partitions. Table 2 shows the results of this evaluation. An approximate time of 24 012 ns is required to process each event. Bear in mind that the most expensive task, schema generation, is only performed once per event type, as once you have the schema for an event type, it is not supposed to change unless it is forced to do so. Thus, the average time each event needs to be processed is around 0.011 ms.

In conclusion, the SP architecture embedded in our proposal can satisfy the requirements of processing large amounts of information: consuming, homogenising, and analysing these data. It can perfectly perform long simulations with a high workload with no latency (such numbers of nanoseconds are insignificant when processing thousands of events per second). The system, in the last test scenario (32 partitions within 4 computers), was able to process up to 509 124 775 events with an average size of 199 B, resulting in a total amount of 101 GB of heterogeneous information processed within 10 min. The scalability of the solution was thus successfully tested, being able to process larger numbers of e/s when we scale our solution, using more partitions with more machines.

8. Related work

In this section, we present other approaches that combine some of the technologies used in this work, and conduct a comparative analysis of our proposal, MEdit4CEP-SP, and those approaches.

8.1. Stream processing & complex event processing

We can highlight the following solutions that integrate the SP paradigm along with CEP to consume, process and analyse data:

First, Carcillo et al. [28] propose an architecture to detect fraud when using credit cards, SCARFF. In this case, Kafka is used as fault-tolerant transaction collection to transport the data streams. The data is then transmitted to the analytics engine which, in this case, is implemented using the Spark Streaming API. Moreover, they use Cassandra for storing purposes. The functionality of the system is as follows: the data, collected in Kafka topics, is consumed in batches from the Spark application, where the data pre-processing is done. The data is normalised using the historical records stored in the database. Next, the data is classified online using their machine learning (ML) [29] models and then stored in the database. SCARFF achieves an incoming rate of 240 transactions per second, which is limited compared to our proposal. In addition, SCARFF, given the nature of its components, processes the data in batches not in pure real time.

D'Silva et al. [30] also propose an architecture to process real-time and historic streams of IoT data. Any kind of IoT device is used as a data source; these devices will be required to send the generated data to Kafka topics. Then, Spark Streaming is used to consume batches of these data, from the Kafka topics, and to process them. The results of the analytics are then passed to the GraphX module of Spark as operations. Next, the output is forwarded to be displayed on a dashboard. Its architecture is able to process not only real-time data, but also historical records that can be streamed into the Spark application directly. They present an evaluation with 1000 events, which is a very limited amount, considering the scope of IoT.

Jung et al. [31] develop UTOPIA, an architecture for processing real-time streams of information in the cloud from heterogeneous IoT devices. In this case, Kafka is used to receive the real-time transmission of a large-capacity data stream. Then, the evaluation of this information is performed with Storm, using a Web Ontology Language Model. Their architecture is also capable of performing Distributed Remote Procedure Calls within the Storm nodes. They present an evaluation, using an average desktop machine, in which the maximum incoming rate of the platform is of about 10 000 stream data per second, far from the performance of our proposal.

Zeydan et al. [32] present AlarmAVEA, which is a scalable application for real-time alarm data analysis. In their work, they combine the SP paradigm with CEP technology in order to predict upcoming alarms based on the analysis of current streams of alarms information. The heterogeneous information is captured in a first module, which will transform these sources into a common

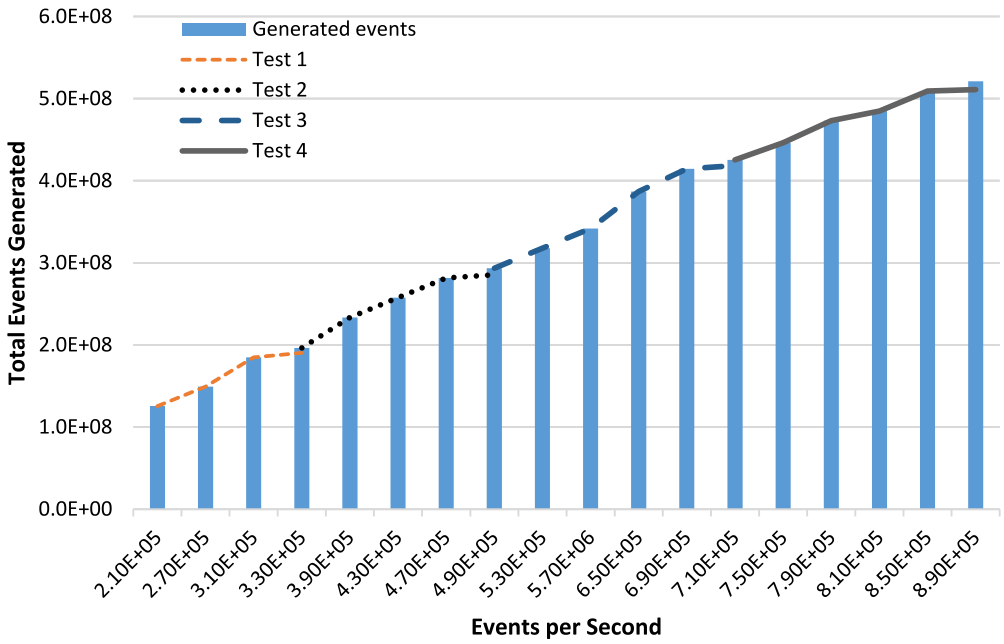

Fig. 12. Results of the performance tests.

structure. Then, these data are transmitted and processed using Apache Kafka and Apache Storm. In addition, the alarm data are analysed in the CEP engine module. The results are displayed on a web dashboard. In this proposal, the authors use Kafka for message transmission and Storm for processing; the main limitation of using Apache Storm is the necessity of stopping the whole system in order to update the configurations, which could be resolved by using Esper as CEP engine, as we propose.

The proposal by Malek et al. [33] deserves special attention. They present an architecture in which they combine several IoT, CEP and Big Data technologies. Their architecture aims to process sensor data in order to develop context-aware applications and services. It is composed of three layers: the first, data acquisition, uses Kaa in order to gather information from IoT devices; the second, data processing, runs an Apache Storm engine in order to analyse the data; the last, data storage and visualisation, uses MongoDB to store the data. A web application is also provided in order to visualise them. Additionally, in order to transmit the data from the collection stage to the processing layer, Apache Flume is used. The authors state that these technologies might be able to process up to 10 000 e/s, although this is far from the performance provided by our proposal, which is 850 000 e/s. A drawback when using Kaa is the heterogeneity of the data input sources: if you wish to add another IoT device, you will have to add all the process logic and redeploy it in your architecture. Furthermore, they mention nothing about heterogeneous data formats as we do in our proposal. Finally, they use Apache Storm as analytics engine, with an already implemented logic in it, which means that if new logics or processing patterns need to be added, the entire execution of the system must be stopped. This issue could be solved using an analytics engine, such as Esper, the one used in our proposal, that allows updates of definitions at runtime.

Stripelis et al. [34] present an architecture that comprises several SP and analytics technologies in order to collect, integrate and analyse data for detecting asthma attacks. These data are collected from medically-related sources, such as AirNow and sensors. The data are then streamed through the architecture using Kafka. They transform these heterogeneous data into a common harmonised structure within an integration layer using Spark SQL. Finally, the data are stored for offline analytics and are also streamed into an online analytics layer, which builds upon Spark, to build models that can predict asthma attacks. The analytics, in this case, focus on statistical models and not on detecting situations in real time. In addition, the author's proposal uses Spark as the main technology, which in some cases, such as processing and integrating data, can be done using Kafka Streams, as we propose, which is much more efficient due to its low latency. In others, such as pure analytics, Spark also presents some disadvantages compared to other stream analytics technologies like Esper, which allow us to change definitions in real time without having the whole solution stopped.

Additionally, Estévez-Ayres et al. [35] propose an architecture, Lostrego, which analyses educational data from heterogeneous data sources in real time. Students and teachers produce events when they use educational resources thanks to monitoring agents that run within these resources. When the user interacts with a resource, the agents generate an event with the information of such an interaction, which is sent to the backend using an HTTP request. The events sent to the analysis layer are stored as a historical record. In the analysis layer, they run services to monitor how students work in groups and to monitor how students work on specific assignments. Moreover, they evaluate the proposed solution that generated 384 702 events in two academic years, each of them processed in less than 13 s. They have plans to apply ML techniques to detect patterns within the generated data. One of the main limitations of this proposal is that events can take up to 13 s to be processed, which is a considerable amount of time compared to our proposal, in which we process each event with an average time of 0.024 ms. In addition, the authors use Ztreamy [36] to stream the information, which they state can process 25 000 events per minute, a much smaller amount of data than what Esper or Kafka Streams can handle.

In general, if we compare our SP architecture to these proposals, most of them use Kafka as a messaging system between components in their architecture, while most of their transformations and processing tasks could be more easily achieved using Kafka Streams, as we do. Concerning analytics, some of the works described cannot detect alerts or situations of interest in real time since they prefer to focus on processing historical records rather than streams of information in real time. In addition, none of

the analysed proposals considers the option of adding, updating or deleting event definitions and event patterns in real time. Finally, the performance of our architecture far surpasses the performance of the systems presented in the cited works.

8.2. Model-driven development integrated with stream processing

A few works we can be found in the literature regarding solutions that integrate MDD and SP for modelling situations that will be automatically deployed. Taking this into consideration, we would like to highlight the most significant ones.

Although the current work is an improved version, the original proposal by Boubeta-Puig et al. [3] must be considered. As mentioned, the authors propose MEdit4CEP, which is a graphical tool that enables CEP analytics to non-expert users through the use of a drag and drop canvas. Once the domain, which represents the event types to be processed, and the complex events, which represent the situations to be detected, are defined, a series of M2T transformations are performed in order to generate the Esper EPL code necessary to deploy such event types and complex events into an Esper CEP engine. One of the limitations of the current solution was persistence, due to the domains and event patterns models not being saved in a database, and thus another user with another computer could not use these models until the first one shared the exported file with the second one. Another limitation was the online deployment, removal and update of existing event types and event patterns within an Esper CEP engine. This limitation has been addressed in the new version developed in this paper, allowing the user to deploy, remove and update event types and complex event definitions at runtime, thanks to the integration of the editor with the SP architecture proposed in this work. In addition, this integration provides us with better performance thanks to the combination of SP and CEP instead of deploying the event types and patterns definitions in a Mule ESB.

The proposal by Clemente at al. [37] deserves special attention, being the most similar approach found in the current literature. In their work, the authors propose an approach, OpenData2CEP, to achieve the integration between MDD and CEP, in order to pursue the analysis of open data sources. Their solution provides M2T transformations that enable the deployment of the defined application in an ESB, which in this case is Mule ESB. The user will set the open data source, define the event type to be consumed, and define the EPL patterns with the actions to be carried out. In their proposal, Esper is used as CEP engine within the ESB application. However, in their tool, it is necessary to know the Esper EPL syntax in order to define the situations to be detected, while in MEdit4CEP-SP, a non-expert user will be able to graphically define not only the domain to be processed but also the complex situations to be detected. In addition, they generate the executable code for a Mule ESB application, whose performance is somewhat inadequate when a high input rate of e/s has to be consumed, while our proposed SP architecture is able to appropriately consume 850 000 e/s using four computers. Finally, regarding format limitations, OpenData2CEP only allows CSV data to be processed, while our proposal can manage the heterogeneity of data in CSV, XML, JSON, as well as raw data itself.

Guerriero et al. [38] present StreamGen, an MDD tool that simplifies the definition and deployment of streaming applications. In their tool, a user can graphically model the steps required to implement the desired streaming application. They then perform M2T transformations in order to generate the code of an Apache Flink application, which will contain all the defined operators and steps. Although their proposal is not focused on performing CEP, Apache Flink could be used for such purpose, and, as they mention, further definitions can be added to the existing meta-model, one of which is Spark. In both cases, using Flink or Spark, these SP and CEP technologies do not allow us to add new definitions in real time, so stopping the execution of the system or architecture will be necessary in order to update existing definitions used within the processing or analytics stage.

The primary obstacle presented in these proposals is the use of Mule ESB as a solution to deploy their definitions. Firstly, according to the Mule Runtime Enterprise Engine Performance Whitepaper [39], the Throughput or Transactions Per Second (TPS) when consuming messages and transforming them, using a Mule Runtime Engine, within a 24-core machine and 36 GB ram, is of 8 000 processing 20 kb messages payload. This represents 160 000 kb/s, resulting in 160 Mb/s, a great distance from the performance achieved in our SP architecture, which was tested in a 4-core machine and 12 GB ram, resulting in 135 080 Mb/s. Secondly, the lack of persistence is another major issue addressed in our solution. Finally, managing heterogeneous input data is currently a must, so being able to use several data structures formats, such as JSON, XML and CSV, is required not only in the definition or processing time but also in the analytics part. That said, we can affirm that our proposal differs from the others not only in terms of efficiency but also in terms of functionalities.

8.3. Comparative analysis

In this subsection, we compare our proposal, MEdit4CEP-SP, against the approaches previously analysed. This work identifies eleven key features that are required when modelling and analysing heterogeneous data. Some of these features are related to modelling capacities, while others to processing functionalities. The key features used to evaluate the proposal are as follows:

- (F1) Scalability: the system must be scalable, allowing growing amounts of data to be processed.
- (F2) Low latency: the system must provide a low latency processing strategy that enables it to perform its actions as fast as possible.
- (F3) Real time analytics: the system must be able to analyse these data in real time in order to detect situations of interest.
- (F4) Data Storage: the system must be able to provide means to store the data being processed.
- (F5) Heterogeneous data: the system must be capable of modelling, consuming, processing and analysing heterogeneous data.
- (F6) Predictive Tools: the system should also be able to provide certain Predictive Technologies to improve the results and anticipate situations to be detected.
- (F7) Graphical domain and alert definitions: the system must be able to graphically and intuitively model the event types to be processed and the situations of interest to be detected.
- (F8) Runtime actions: the system must be able to add, update and remove such definitions at runtime, without the execution having to be stopped.
- (F9) Persistence of definitions: the system must be able to store the domain and alert definitions so anyone can use them again.
- (F10) Performance evaluation: the system must provide an evaluation to show its feasibility and correctness.
- (F11) Replicability: the proposal must provide means to be used by anyone.

In Table 3, the analysed approaches and our proposal are directly compared using the mentioned features.

Although the previous subsections already noted some of the differences between our proposal and the related work, Table 3

Table 3
Comparative of analysed proposals.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
Our proposal	X	X	X	X	X	–	X	X	X	X	X
Carcillo et al. [28]	X	–	–	X	–	X	–	–	–	X	X
D'Silva et al. [30]	X	–	X	X	X	–	–	–	–	X	–
Jung et al. [31]	X	–	X	–	–	–	–	–	–	X	–
Zeydan et al. [32]	X	–	X	X	X	X	–	–	–	–	–
Malek et al. [33]	X	–	X	X	–	–	–	–	–	X	–
Stripelis et al. [34]	X	–	X	X	X	X	–	–	–	–	–
Estévez-Ayres et al. [35]	X	X	X	X	–	–	–	–	–	X	–
Boubeta-Puig et al. [3]	X	–	X	X	–	–	X	–	–	–	X
Clemente et al. [37]	X	–	X	X	–	–	X	–	–	–	–
Guerriero et al. [38]	X	X	X	–	–	X	X	–	–	–	X

shows the key functionalities provided by each of the approaches already discussed.

Scalability (F1) can be achieved by all the approaches thanks to the use of modern technologies, such as Apache Kafka or Apache Storm, in their data transmission layers, which allow them to process huge amounts of data and increase their input rates, adding more processing nodes. Concerning low latency (F2), only those proposals using a software solution implementing a low latency strategy to process the data (not just to transmit it) can achieve this essential requirement. Real time analytics (F3) is also achieved in most proposals except [28], which is focused on processing historic records of data. Being able to analyse data in real time, with the aim of detecting situations of interest as soon as possible is a must in IoT domains.

Regarding data storage (F4), all the approaches except [31,38] are capable of storing the processed data or detected alerts, which might be useful to perform offline analytics and to keep these alerts as historical records, among others. It is worth mentioning that most of these architectures tend to implement NoSQL databases within their architectures, which are especially useful in heterogeneous IoT domains, where the schema or structure is variable, and addition operations are more common than queries.

As for heterogeneous data (F5), the bulk of these proposals are able to process data from several sources, but these data sources produce the information in a homogenised or normalised data structure. By heterogeneous data, we refer to the capacity of accepting multiple input data structures for analysis. This is a special feature which, in this case, is achieved in [30,32,34], and in our proposal. As mentioned before, being able to consume, process and analyse heterogeneous data formats and data sources is currently a must in these complex architectures.

Predictive tools (F6) are implemented in a few of these proposals, while it is not currently accomplished in our work, but we plan to introduce it (see Section 10). As for the definitions of domains and alerts in a graphical editor (F7), this is achieved in our approach and in [3,37,38], while the others focus on the analytics part and do not provide the users with this essential functionality that facilitates the definition of CEP domains and alerts to be analysed.

The next feature, runtime actions (F8), is only achieved in our proposal, because most of these approaches use technologies that do not allow them to perform changes at runtime, while the ones using Esper do not implement these updates in real time. Regarding the persistence of the definitions (F9) of such models, this is only achieved in our approach; the rest make no mention of the possibility of storing the modelled definitions in a database to be reused in the future.

The following feature to be analysed is the evaluation of the proposed solution (F10). Most of these proposals perform an evaluation of their solutions presenting a case study with no kind of benchmark results. Thus, in some cases, it is complicated to compare our proposal against others in terms of technical metrics. In [28], the authors manage to process up to 240 transactions per second. In [30], a simulation is performed, sending only 1000 simple events to the Kafka Cluster. We expect to have larger number of events in an IoT scenario; this is why we provided tests with a much higher number of incoming events. In [31], the authors perform tests with batch sensor data, proving that their proposed solution can handle around 10 000 messages per second as maximum throughput, which is considerably lower than the maximum throughput performed by our proposal. Furthermore, in [33], the authors state that the solutions used in their implementation would be able to process up to 10 000 e/s. In [35], the authors perform an evaluation with real users consuming and processing large amounts of information. However, they state that, in some cases, the system takes up to 13 s to process an event, which is a considerably long time, which would have to be improved to be able to react to complex situations as quickly as possible. The proposals in [3,37] have the CEP embedded in a Mule ESB application, which is no longer the best alternative to process large amounts of data in real time. According to the Mule Runtime Enterprise Engine Performance Whitepaper [39], the Throughput or TPS when consuming messages and transforming them, using a Mule ESB Runtime Engine, within a 24-core machine and 36 GB RAM, is of 8 000 processing 20 kb messages payload. This corresponds to 160 000 kb/s, which results in 160 Mb/s, far from the performance achieved in our SP architecture, which was tested in a cluster of four 4-core machines with 12 GB RAM resulting in 135 080 Mb/s.

Finally, concerning the last key feature, replicability (F11), beyond Boubeta-Puig et al.'s proposal [3], only two of the remaining proposals have the code of the tools they present publicly available, which are [27] and [37]. We have downloaded them to replicate the results presented in their manuscripts. Regarding Carcillo's proposal [28], which is available at [40], unfortunately, we replicated the steps present on their Docker Hub page with no success. As for Guerriero's proposal [38], which is available at [41], we have cloned the repository and installed the required tools, despite the lack of installation documentation. We have tried the tool with the examples provided and we have experienced great difficulties to develop a new application within their tool itself. From our perspective, their tool is not user-friendly and it would be extremely difficult to be used by a non-expert user. Our proposal's components together with the dataset are online available for replication of the results in the following GitLab repositories [42–44] along with their detailed instructions for usage, as well as in our dataset [25].

Therefore, these previously analysed proposals fail to accomplish one or more of the required features that are key in any system for modelling, consuming, processing, transforming and detecting situations of interest from heterogeneous data in the IoT. We can affirm that most existing proposals do not benefit from using Kafka as a SP application, at best creating individual components to pre-process the information. Our proposal uses Kafka as a communication module and Kafka Streams to perform the transformations, both working as one component, thus providing additional advantages: on the one hand, the already well-known advantages provided by Kafka itself, such as fault-tolerant storage, communications management or durability, and, on the other, the benefits provided by using Kafka Streams API when processing streams of data in real time, such as high scalability, exactly-once processing or high throughput rate. This results in a complete Kafka Stream Application able to consume, transform and process huge amounts of information in real time, with a very low latency strategy. While Spark and Storm are commonly used for the analysis, our solution uses Esper, providing us with the advantage of not having to stop the system execution

to add a new required analysis of events. In addition, some of these proposals integrate modules for ML and predictive analysis in their architecture, which we plan to do as future work. Most of these proposals provide no way to define, in a friendly and intuitive manner, the situations we wish to be detected within the CEP engine, as we do with our graphical editor MEdit4CEP-SP. Finally, as demonstrated in previous sections, the performance of our proposal surpasses all the others.

9. Answers to research questions

In Section 1, we presented four RQs to be addressed in this research. The answers to these questions are as follows:

(RQ1) How can organisations model, process and analyse their heterogeneous data?

In order to model or define domains, organisations and users may use several existing tools [3,37,38], while for processing and analysing heterogeneous data, there are proposals such as [30,32,34]. These solutions are adequate for providing such functionalities in an isolated way, but none of them provide all these functionalities in an all-in-one integrated solution. In this work, we have presented MEdit4CEP-SP, designed to model CEP domains and event patterns with an SP architecture, and which allows us to consume, process and analyse heterogeneous data. MEdit4CEP-SP is a tool that provides organisations with all of these capabilities, which are of great value in heterogeneous IoT domains.

(RQ2) How important is it to provide organisations with a set of tools that facilitates such tasks and the decision-making process, and for the proposed integration (SP architecture + MEdit4CEP) to be able to provide users with such capabilities?

There are employees in companies who are specialists in the domain itself but who lack programming skills, so it is important to provide them with the tools that allow them to model such domains and analyse the information they have. It is also essential to provide these functionalities in an intuitive and friendly way, so they can be used without significant difficulties. As a result, such a toolkit would help domain experts to avoid spending a lot of time on defining the domains and situations to be detected, allowing them to be more efficient in their work and providing them with more valuable knowledge that is essential for decision-making. According to the results of the evaluation carried out in Section 7, users found the proposed integration, MEdit4CEP-SP (MEdit4CEP + SP architecture) to be very useful, allowing them to model, process, analyse and detect situations of interest, in real time, from heterogeneous data in IoT domains. The vast majority of respondents agree that using MEdit4CEP-SP facilitates the definition of domains and event patterns. They also appreciate the integration with the SP architecture, which enables them to consume, process and detect situations of interest using the definitions previously modelled within the editor.

(RQ3) Is the proposal more effective and efficient than other existing alternatives in achieving the modelling and definition of heterogeneous CEP domains and event patterns in a friendly and intuitive way, as well as their analytics in real time?

The answer is affirmative. The proposal is divided into two key components, one for modelling domains and event patterns, and another for processing and analysing such data. Concerning the editor, the evaluation shows that the users are able to easily define domains and event patterns in a very short period of time (less than 30 min). Related to the analytical part, our performance evaluation shows that the SP architecture is able to consume, process and analyse up to 850 000 e/s using 4 desktop computers, being much more efficient than other alternatives analysed. As a result, the integration of both is a better solution than others integrating MDD solutions with ESB applications, in which the performance of such ESB applications is not as good as that of our SP architecture, having a TPS of 160 Mb/s with a high-spec machine, while our SP architecture is able to achieve a TPS of 135 080 Mb/s using an average desktop machine. In addition, of the functionalities arising from such integration (MEdit4CEP + SP architecture), it is worth noting the persistence of event types and event patterns in the database, which is not accomplished by any of the previously analysed proposals. Such definitions can be retrieved, updated, and deleted at any time by anyone using MEdit4CEP-SP. Finally, the deployment, update and removal of such definitions from the editor to the CEP engine are done at runtime, without the need to stop the entire system to apply such changes.

(RQ4) How important is it to be able to add, update and remove existing event types and patterns at runtime and is the proposed integration able to provide users with such capabilities?

As mentioned, one of the main advantages of the Esper CEP engine, compared to other CEP alternatives, is the capability of adding or removing existing event types and patterns at runtime, meaning the system does not need to be stopped or paused for this addition or deletion. Stopping the entire system, for just a few seconds, can mean the loss of essential information that needs to be analysed. In MEdit4CEP-SP, it is not necessary to stop the system to perform such changes. MEdit4CEP-SP allows the addition, update or removal of event types and event patterns from the CEP engine at runtime, and, what is more, in a friendly and intuitive way. According to the results of the questionnaire from Section 7, all the respondents found it positive to be able to perform such actions and also found it very useful and easy to do, by means of the set of tools provided.

10. Conclusions and future work

This paper presents MEdit4CEP-SP, a model-driven system that integrates CEP and SP technologies for consuming, processing and analysing heterogeneous data, allowing domain experts to model these heterogeneous data domains and the situations of interest to be detected, in a friendly and intuitive way. Since these situations are detected in real time, domain experts can react immediately, thus improving the decision-making process in their organisations.

MEdit4CEP-SP simplifies and brings complex technologies, such as CEP or SP that require great deal of experience in their use, to any kind of users, regardless of their knowledge in computer science, in a simple and friendly way. More specifically, MEdit4CEP-SP provides a graphical editor that allows users to define the event types and event patterns typical of their domain. Once such definitions are ready, they are automatically deployed in the CEP engine of the SP architecture so the incoming heterogeneous data can be analysed, at runtime. Moreover, thanks to MEdit4CEP-SP, users can update and remove these modelled event types and event patterns that are already deployed in the CEP engine at runtime, meaning there is no need to stop the entire system if changes are to be applied.

MEdit4CEP-SP also provides users with persistence thanks to the inclusion of the REST API with the NoSQL database. When an event type or event pattern model is saved within the editor, it is now automatically stored in the database. Such definitions are available to be reused by other domain experts using other MEdit4CEP-SP instances from different computers. It is worth noting that, as the SP architecture can be deployed in the cloud, different users have the option to share this CEP engine. This means they can add, remove, and update CEP domains and event patterns from the same CEP engine using MEdit4CEP-SP, making the proposed system collaborative.

Regarding evaluation, we tested MEdit4CEP-SP, in terms of usability and technical metrics. On the one hand, we evaluated the usability of the system to prove it is user-friendly and allows users to improve their decision-making. On the other hand, we evaluated the performance of the system to prove it is capable of consuming, processing, and analysing huge amounts of heterogeneous data in real time. Concerning usability, we tested the graphical editor with a group of 28 real users. They were asked to complete a task similar to that presented in Section 6. Subsequently, they answered a questionnaire designed to evaluate their experience with the proposed set of tools. The survey showed the participants were highly satisfied (75%) with MEdit4CEP-SP. Regarding technical metrics, we thoroughly evaluated our SP architecture. The results of the performance tests proved the scalability of the solution, with it being able to process more data using more computing nodes. We also found the efficiency of the SP architecture was superior to others, presenting TPS of 135 080 Mb/s in a cluster of 4 machines. Additionally, we compared our proposal with the other previously analysed approaches, underlining that it fulfils almost all the functionalities required by a system to model, consume, process, and analyse heterogeneous data domains.

Concerning future work, some of the respondents provided interesting feedback to improve the usability of the tested editor, which will be considered in the future version of the editor. They also suggested new functionalities, such as adding actions to be performed when a complex event is detected or including a wizard to guide the user when they first use the editor. Additionally, we plan to apply the editor to other application domains such as air pollution, road traffic, e-health and distributed environments [14,45–47]. Furthermore, we would like to incorporate predictive analysis techniques to improve the analytics results and to anticipate the situations to be detected. Finally, we plan to add new ways to integrate MEdit4CEP-SP with other cloud platforms, such as FIWARE, so the user will not only be able to automatically interact with the SP architecture, but also with other solutions to process and analyse big data in real time.

CRediT authorship contribution statement

David Corral-Plaza: Investigation, Software, Validation, Writing - original draft. **Guadalupe Ortiz:** Conceptualization, Writing - review & editing, Resources, Supervision, Project administration, Funding acquisition. **Inmaculada Medina-Bulo:** Conceptualization, Writing - review & editing, Resources, Supervision, Project administration, Funding acquisition. **Juan Boubeta-Puig:** Conceptualization, Methodology, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partly supported by the Spanish Ministry of Science and Innovation and the European Regional Development Fund (ERDF) under project FAME (RTI2018-093608-B-C33), and also by the pre-doctoral program of the University of Cádiz, Spain (2017-020/PU/EPIF-FPI-CT/CP).

References

[1] S. Nadal, et al., A software reference architecture for semantic-aware Big Data systems, Inf. Softw. Technol. 90 (2017) 75–92, http://dx.doi.org/10.1016/j.infsof.2017.06.001.
[2] D. Corral-Plaza, I. Medina-Bulo, G. Ortiz, J. Boubeta-Puig, A stream processing architecture for heterogeneous data sources in the Internet of Things, Comput. Stand. Interfaces 70 (2020) 103426, http://dx.doi.org/10.1016/j.csi.2020.103426.
[3] J. Boubeta-Puig, G. Ortiz, I. Medina-Bulo, MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0, Knowl.-Based Syst. 89 (2015) 97–112, http://dx.doi.org/10.1016/j.knosys.2015.06.021.
[4] J.A. Morente-Molinera, I.J. Pérez, M.R. Ureña, E. Herrera-Viedma, On multigranular fuzzy linguistic modeling in group decision making problems: A systematic review and future trends, Knowl.-Based Syst. 74 (2015) 49–60, http://dx.doi.org/10.1016/j.knosys.2014.11.001.
[5] SOA Vs EDA: Is not life simply a series of events? | confluent, 2020, https://www.confluent.io/blog/soa-vs-eda-is-not-life-simply-a-series-of-events/(accessed Apr. 06, 2020).
[6] Streaming integrator, 2020, https://wso2.com/integration/streaming-integrator/ (accessed Apr. 06, 2020).
[7] Apache storm, 2020, http://storm.apache.org/ (accessed Apr. 06, 2020).
[8] Apache kafka streams, 2020, https://kafka.apache.org/documentation/streams/ (accessed Apr. 06, 2020).
[9] Complex event processing streaming analytics streaming SQL - espertech, 2020, http://www.espertech.com/ (accessed Apr. 06, 2020).
[10] G. Ortiz, J. Hernández, Integrating extra-functional properties in model-driven web service development, 2007.
[11] Object management group a proposal for an MDA foundation model, 2005.
[12] J. Boubeta-Puig, G. Ortiz, I. Medina-Bulo, ModeL4CEP: Graphical domain-specific modeling languages for CEP domains and event patterns, Expert Syst. Appl. 42 (21) (2015) 8095–8110, http://dx.doi.org/10.1016/j.eswa.2015.06.045.
[13] Epsilon, 2020, https://www.eclipse.org/epsilon/ (accessed Jul. 06, 2020).
[14] J. Boubeta-Puig, G. Díaz, H. Macià, V. Valero, G. Ortiz, MEdit4CEP-CPN: An approach for complex event processing modeling by prioritized colored Petri nets, Inf. Syst. 81 (2019) 267–289, http://dx.doi.org/10.1016/j.is.2017.11.005.
[15] A. Calderón, J. Boubeta-Puig, M. Ruiz, MEdit4CEP-Gam: A model-driven approach for user-friendly gamification design, monitoring and code generation in CEP-based systems, Inf. Softw. Technol. 95 (2018) 238–264, http://dx.doi.org/10.1016/j.infsof.2017.11.009.
[16] Apache kafka streams architecture, 2020, https://kafka.apache.org/24/documentation/streams/architecture (accessed Apr. 06, 2020).
[17] What is mule esb? | mulesoft, 2020, https://www.mulesoft.com/resources/esb/what-mule-esb (accessed Apr. 06, 2020).
[18] Why avro for kafka data? - confluent, 2020, https://www.confluent.io/blog/avro-kafka-data/ (accessed Jul. 06, 2020).
[19] D. Corral-Plaza, J. Boubeta-Puig, G. Ortiz, A. Garcia-de Prado, An internet of things platform for air station remote sensing and smart monitoring, Int. J. Comput. Syst. Sci. Eng. 32 (1) (2020).
[20] A. Ostfeld, Water distribution networks, Stud. Comput. Intell. 565 (2015) 101–124, http://dx.doi.org/10.1007/978-3-662-44160-2_4.
[21] Metering and submetering | alliance for water efficiency, 2020, https://www.allianceforwaterefficiency.org/resources/metering (accessed Apr. 06, 2020).
[22] Toronto water says $2, 500 bill caused by leaking toilet, scarborough man disagrees, 2020, https://toronto.citynews.ca/2018/05/14/water-bill-leaky-toilet-scarborough/ (accessed Apr. 06, 2020).
[23] Holly springs family gets $49, 000 water bill - ABC11 raleigh-durham, 2020, https://abc11.com/finance/outrageous-holly-springs-family-gets-{$}49000-water-bill/5460685/ (accessed Apr. 06, 2020).
[24] Grupo Energético de puerto real s.a, 2020, http://www.grupoenergetico.es/gen/ (accessed Apr. 06, 2020).
[25] D. Corral-Plaza, G. Ortiz, I. Medina-Bulo, J. Boubeta-Puig, Dataset for MEdit4CEP-SP: a model-driven solution to improve decision-making through user-friendly management and real-time processing of heterogeneous data streams, Vol. 5, 2020. http://dx.doi.org/10.17632/RYBGTBBHS9.5.
[26] Brighttag/kafkameter: Kafka jmeter extension, 2020, https://github.com/BrightTag/kafkameter (accessed Jul. 06, 2020).
[27] Throughput shaping timer, 2020, https://jmeter-plugins.org/wiki/ThroughputShapingTimer/(accessed Jul. 06, 2020).
[28] F. Carcillo, A. Dal Pozzolo, Y.A. Le Borgne, O. Caelen, Y. Mazzer, G. Bontempi, SCARFF: A scalable framework for streaming credit card fraud detection with spark, Inf. Fusion 41 (2018) 182–194, http://dx.doi.org/10.1016/j.inffus.2017.09.005.
[29] T.M. Mitchell, Machine learning: A guide to current research, 1986.

[30] G.M. D'Silva, A. Khan, Gaurav, S. Bari, Real-time processing of IoT events with historic data using apache kafka and apache spark with dashing framework, in: RTEICT 2017-2nd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology, Proceedings, Jul. 2017, 2018-January, 2017, pp. 1804–1809, http://dx.doi.org/10.1109/RTEICT.2017.8256910.

[31] H.S. Jung, C.S. Yoon, Y.W. Lee, J.W. Park, C.H. Yun, Cloud computing platform based real-time processing for stream reasoning, in: 2017 Sixth International Conference on Future Generation Communication Technologies (FGCT), 2017, http://dx.doi.org/10.1109/FGCT.2017.8103400.

[32] E. Zeydan, U. Yabas, S. Sözüer, Ç. Ö. Etemoğlu, Streaming alarm data analytics for mobile service providers, in: Proceedings of the NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium, 2016, pp. 1021–1022, http://dx.doi.org/10.1109/NOMS.2016.7502953.

[33] Y.N. Malek, et al., On the use of IoT and big data technologies for real-time monitoring and data processing, Procedia Comput. Sci. 113 (2017) 429–434, http://dx.doi.org/10.1016/j.procs.2017.08.281.

[34] D. Stripelis, J.L. Ambite, Y.Y. Chiang, S.P. Eckel, R. Habre, A scalable data integration and analysis architecture for sensor data of pediatric asthma, in: Proceedings - International Conference on Data Engineering, 2017, pp. 1407–1408, http://dx.doi.org/10.1109/ICDE.2017.198.

[35] I. Estévez-Ayres, J. Arias Fisteus, C. Delgado-Kloos, Lostrego: A distributed stream-based infrastructure for the real-time gathering and analysis of heterogeneous educational data, J. Netw. Comput. Appl. 100 (2017) 56–68, http://dx.doi.org/10.1016/j.jnca.2017.10.014.

[36] J.A. Fisteus, N.F. García, L.S. Fernández, D. Fuentes-Lorenzo, Ztreamy: A middleware for publishing semantic streams on the web, J. Web Semant. 25 (2014) 16–23, http://dx.doi.org/10.1016/j.websem.2013.11.002.

[37] P. Clemente, A. Lozano-Tello, Model driven development applied to complex event processing for near real-time open data, Sensors 18 (12) (2018) 4125, http://dx.doi.org/10.3390/s18124125.

[38] M. Guerriero, A. Nesta, E.D.I. Nitto, Streamgen: A UML-based tool for developing streaming applications, in: MiSE '18: Proceedings of the 10th International Workshop on Modelling in Software Engineering, 2018, pp. 57–58, http://dx.doi.org/10.1145/3193954.3193964.

[39] TY - Mule runtime engine enterprise performance | mulesoft, 2020, https://www.mulesoft.com/ty/wp/esb-enterprise-performance (accessed Apr. 06, 2020).

[40] Fabriziocarcillo/scarff: SCARFF (scalable real-time frauds finder) is a framework which enables credit card fraud detection, 2020, https://github.com/fabriziocarcillo/scarff (accessed Dec. 02, 2020).

[41] Micheleguerriero/streamgen, 2020, https://github.com/MicheleGuerriero/streamgen (accessed Dec. 02, 2020).

[42] UCASE / public / SP-architecture · gitlab, 2020, https://gitlab.com/ucase/public/sp-architecture (accessed Dec. 02, 2020).

[43] UCASE / public / medit4cep-SP · gitlab, 2020, https://gitlab.com/ucase/public/MEdit4CEP-SP (accessed Dec. 02, 2020).

[44] UCASE / public / API-medit4cep-SP · gitlab, 2020, https://gitlab.com/ucase/public/API-MEdit4CEP-SP (accessed Dec. 02, 2020).

[45] G. Díaz, H. Macià, V. Valero, J. Boubeta-Puig, F. Cuartero, An intelligent transportation system to control air pollution and road traffic in cities integrating CEP and colored Petri nets, Neural Comput. Appl. 32 (2) (2020) 405–426, http://dx.doi.org/10.1007/s00521-018-3850-1.

[46] I. Calvo, M.G. Merayo, M. Núñez, A methodology to analyze heart data using fuzzy automata, J. Intell. Fuzzy Systems 37 (6) (2019) 7389–7399, http://dx.doi.org/10.3233/JIFS-179348.

[47] R.M. Hierons, M.G. Merayo, M. Núñez, Bounded reordering in the distributed test architecture, IEEE Trans. Reliab. 67 (2) (2018) 522–537, http://dx.doi.org/10.1109/TR.2018.2800093.