Automated Decision Support IoT Framework

Willian Tessaro Lunardi*†, Leonardo Amaral†, Sabrina Marczak†, Fabiano Hessel†, Holger Voos*
*University of Luxembourg, Interdisciplinary Centre for Security, Reliablity and Trust (SnT)
Luxembourg, Luxembourg
{willian.lunardi, holger.voos}@uni.lu
†Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
leonardo.amaral@acad.pucrs.br,
{sabrina.marczak, fabiano.hessel}@pucrs.br

***Abstract*—During the past few years, with the fast development and proliferation of the Internet of Things (IoT), many application areas have started to exploit this new computing paradigm. The number of active computing devices has been growing at a rapid pace in IoT environments around the world. Consequently, a mechanism to deal with this different devices has become necessary. Middleware systems solutions for IoT have been developed in both research and industrial environments to supply this need. However, decision analytics remain a critical challenge. In this work we present the Decision Support IoT Framework composed of COBASEN, an IoT search engine to address the research challenge regarding the discovery and selection of IoT devices when large number of devices with overlapping and sometimes redundant functionality are available in IoT middleware systems, and DMS, a rule-based reasoner engine allowing to set up computational analytics on device data when it is still in motion, extracting valuable information from it for automated decision making. DMS uses Complex Event Processing to analyze and react over streaming data, allowing for example, to trigger an actuator when a specific error or condition appears in the stream. The main goal of this work is to highlight the importance of a decision support system for decision analytics in the IoT paradigm. We developed a system which implements DMS concepts. However, for preliminarily tests, we made a functional evaluation of both systems in terms of performance. Our initial findings suggest that the Decision Support IoT Framework provides important approaches that facilitate the development of IoT applications, and provides a new way to see how the business rules and decision-making will be made towards the Internet of Things.**

I. Introduction

The term Internet of Things (IoT) was coined in 1998 [1] and defined as the computing paradigm that allows people and things to be connected Anytime, Anyplace, with Anything and with Anyone, ideally using Any path/network and Any service [2]. In this sense, there are current market statistics and predictions that demonstrate a rapid growth in computing device deployments related to IoT environments. By 2020, it is estimated that there will be 50 to 100 billion IoT devices connected to the Internet [3].

According to experts from industry and research, the IoT introduction into the manufacturing environment is ushering in a fourth industrial revolution called Industrial Internet of Things (IIoT or Industry 4.0). The adoption of several computing devices (RFID, sensors, and actuators) in industrial environments has improved manufacturing processes and their outcomes (e.g., cost savings, greater efficiency and speed, smarter products and services). These statistics and facts imply that we will be faced with a vast amount of IoT devices. In this way, the properly use of device data with the assistance of automated decision support systems can help companies to grasp the emerging opportunities from the IoT and improve their competitive advantage.

A huge number of applications requires continuous and timely processing of information as it flows from the perception layer. Examples include intrusion detection systems which analyze network traffic to identify possible attacks; environmental monitoring applications which process raw data coming from sensor networks to identify critical situations; or applications performing online analysis to identify trends and forecast future values. This requirement lead us to a question: How information systems working with IoT data will overcome the inherent complexity of device discovery and data volume in order to provide useful decision support?

To make it possible, streams of data from devices need to be defined through the selection of devices, as well as the data in motion (storage operation adds a great deal of unnecessary latency to the process) need to be constantly transformed, fused, and observed to enable automated actions. In this sense, the challenge and time-consuming task is to select devices that best suits with the IoT application requirements, and allow to IoT software engineers to define business rules for automated decision making over the vast amounts of data flowing from the perception layer (streams of data from millions of devices) to the application layer.

By analyzing others IoT middleware systems and frameworks we found that they do not provide methods to allow automated decision making based on device data stream. Trying to fill this gap, we propose in this work a software framework composed of a rule-based decision management system and previous works (COMPaaS [4], and COBASEN [5]). The framework allows IoT middleware user's engineers who are not aware of devices domain and middleware patterns to be able to define data streams through the discovery and selection of devices which are registered in the middleware, and use these data streams to define business rules for automated

978-1-5090-1314-2/16/$31.00

decision management.

The remainder of this paper is organized as follows: Section II highlights some important concepts, describes our previous works COMPaaS (Cooperative Middleware Platform as a Service) and COBASEN (Context-based Search Engine for Industrial IoT), architectures and reviews relevant literature. Section III presents our system approach. Section IV provides implementation details including tools, software platforms, and data sets used in this work, and our evaluation. Section V discusses some important issues addressed during this work. Section VI concludes the paper with final considerations and prospects of future work.

II. Background and Related Work

In this Section we briefly describe IoT middleware and IoT search engine systems focusing on our previous works – COMPaaS [4], COBASEN [5]. We also present definitions and methods for stream processing.

A. IoT Middleware Systems

IoT ecosystems are based on a layered architecture style and use this view to abstract the integration of objects and to provide services solutions to applications [6]. In IoT high-level system layers as the application layer are composed of IoT applications and middleware system, which is a software layer interposed between the infrastructure of devices and applications, and is responsible to provide services according to devices functionality [7]. IoT middleware systems have evolved from hiding network details to applications into more sophisticated systems to handle many important requirements, providing support for heterogeneity and interoperability of devices, data management, security [8], etc.

Many researchers have proposed the use of semantic middleware to interoperate the different classes of devices communicating through different communication formats [9]. The semantic model typically uses XML and ontologies to establish the metadata and meaning necessary to support interoperability [10], [11], [12], [13]. Like the semantic web, semantic middleware seeks to create a common framework that enables data sharing and exchange across distributed devices, applications and locations.

Several proposed system architectures for IoT middleware systems comply with Service-Oriented Architecture (SOA). This approach allows each device to offer its functionality as standard services. Moreover, SOA architecture refer to enables interoperability in various domains, like industry, environment, and society, and supports open and standardized communication through all layers of web services. The IIoT can use a SOA-based IoT middleware in order to meet applications needs, providing infrastructure abstractions to applications and offering multiple services [14].

In this sense, in a prior work we described COMPaaS [4]. COMPaaS is a SOA-based IoT middleware which provides to IoT software engineers a simple and well-defined infrastructure of services. Behind these services there is a set of system layers that deals with the users and applications

```
<ns3:ReportSpec
    xmlns:ns2="http://service.middleware.compaas/"
    xmlns:ns3="http://service.gather.middleware.compaas/">
    <resources>
        <resource>
            <uri>
                http://192.168.0.10:8080/Termometer14030/resource
            </uri>
            <uri>
                http://192.168.0.10:8080/Termometer14040/resource
            </uri>
        <resource>
    </resources>
    <constraintSpec>
        <repeatPeriod unit="MS">5000</repeatPeriod>
        <duration unit="MS">30000</duration>
    </constraintSpec>
    <outputSpecs>
        <outputSpec outputName="Spec14030">
            <setSpec set="CURRENT"/>
            <groupSpec groupBy="RESOURCE"/>
            <dataOutputSpec operation="AVERAGE"/>
        </outputSpec>
    </outputSpecs>
</ns3:ReportSepc>
```

Fig. 1. Example of data stream specification report to collect data from two thermometer devices.

requirements, for example, request and notification of data, management of computing devices, communication issues, and data management.

COMPaaS architecture is composed of three main cooperating systems: API, Middleware, and Logical Device. The API provides methods to be used by applications that desire to use middleware services. Middleware is the system responsible for abstracting the interactions between applications and devices and hides all the complexity involved in these activities. Logical Device is the system responsible for hide all complexity of computing devices and abstracts their functionality. These systems are based on "Subscribe/Notify" communication pattern.

In order to get data from devices using COMPaaS, we describe here the necessary interactions and exchange of data [4]:

- The application uses the API to get access to middleware web service interface (SOAP) to subscribe its specification. In the specification is defined which will be the devices used, and operators. An example of data stream specification report are shown in Fig. 1.
- Middleware interprets the application specification and creates the respective collection cycles.
- Middleware uses each Logical Device web service interface (REST) to subscribe and start the devices involved in the collection cycles.
- Each Logical Device provides the requested data to the Middleware.
- Middleware processes the data according to the specification parameters and, at the end of each cycle, sends the reports to the application. At the end of the entire cycle, Middleware stops and unsubscribes the devices.

B. IoT Search Engines

A search engine is a system that aims to create an index of objects and use this index to respond queries from users. In the IoT, the role of the search engines is to act as a meeting

point for IoT context producers to register the availability of their devices, and IoT context consumers to discover them. In this sense, the next works are related to device discovery.

Objects in the IoT will be mobile, dynamic, and will generate massive amounts of frequently changing information. Thus, there is a need for IoT search engines capable of identify smart objects, discovery their services and interact with those objects [15], as well as an IoT search engine that is capable of searching the rapidly changing information generated by IoT-enabled objects [16].

COBASEN addresses the research challenge regarding the discovery and selection of IoT devices available in IoT middleware systems. The main features of COBASEN can be summarized as follows: (1) Observe the middleware and gather the devices context provided by the IoT context providers; (2) Index the devices received from the Context Module using an Inverted Index strategy; (3) Enable the search, and selection of devices; (4) Guide the user in the definition of the data stream specification report (i.e., file which is defined how the data streams will be composed, grouped, fused, etc.); (5) Generate and send additional files to the middleware.

Linked Sensor Middleware (LSM) [17] [18] provides limited searching functionality such as selecting devices based on location and device types. Nevertheless, all the searching capability uses SPARQL query language. GSN [19] is another IoT middleware similar to LSM and address the challenges of device data integration and distributed query processing. In short, GSN lists all devices available in a combo-box, which is used by the user to select the desired device. Xively [20] (formely known as COSM) is another approach that connects and collects data from devices to provide real-time control and data storage. Xively offers only keyword search. Microsoft SensorMap [21] only allows users to select sensors by using a location map, by sensor type and by keywords.

There are already some pioneers works in real-time retrieval of sensor data with some prototype systems developed (e.g., Snoogle [22], Microsearch [23], and MAX [24]). All these systems require that the sensors to be searched have a textual description attached so that they can be searched through keyword matches.

Grosky et al. [25] proposed a method to express sensor locations in meta-data and to support current-location-based searches. However, the work mainly supports static locations while for moving sensors, only the latest locations can be retrieved.

Dyser is a search engine proposed by Ostermaier et al. [16] for real-time IoT, which uses statistical models to make predictions about the state of its registered devices. When a user submits a query, Dyser pulls latest data to identify the actual current state to decide whether it matches the user query. Prediction models help to find matching sensors with minimum number of sensors data retrievals. It support constraints based on the latest states of sensors, but historical states can not be retrieved. A hybrid search engine is proposed in [26], it is a search engine for effective multimodal query processing to obtain data generated by things in real time.

C. Complex Event Processing (CEP)

CEP has enhanced the capacity of data analysis in information systems, it is still a big challenging task since events are rapidly increasing in the Internet of Things [27]. Meanwhile, CEP is not a new terminology, as Luckham first introduced [28]. It has been exploited during the past years mostly in big data analytics [29] RFID [30], Internet of Things (IoT) [31], failure prediction systems [32], real-time grid monitoring [33], and healthcare [34]. There are some instances that they adopt CEP engines in their applications, while others design new event processing engines and system architectures [33], [35], [31], [32], [36] with evaluation to show the usefulness and scalability.

Appropriate CEP engines (also know as inference engines) already exist for complex event processing, such as Storm [37], Drools [38], Jess [39], and Esper [38], [40]. It play useful roles in formulating machine-readable rules for determining the trigger sequences of events for a particular activity or process.

Storm is a free and open source distributed real-time computation system, which makes it easy to reliably process unbounded streams of data in real-time processing by batch operations [37]. Drools [38], and Esper [41], [40] are CEP engines which include a module providing native support for events evaluation and temporal logic analysis in a single work node.

Drools uses natural-language-based rule set, editing, and development (*drl* and *dsl* files), whereas Jess requires specific syntax to represent rules (JessML or CLIPS). Drools and Jess both support facts that can be expressed in standard Java class and methods. They both follow the Java Bean approach to insert Java objects in working memory. With respect to Drools, facts are objects (Java beans) from the application that are being asserted into the working memory. The fact that Drools and Jess both support the JSR-94 standard is an asset that provide the potential for more easily interchanging these tools in the future, for research purposes.

III. System Approach

The goals of our system framework are to allow IoT software engineers to define device data streams through the discovery and selection of devices that best suit the application requirements and to allow software engineers to define business rules and use data from devices when it is still in motion, extracting valuable information from it through CEP, allowing automated decision making through event-based reports. The Decision Suport IoT Framework is complied with any SOA-based middleware solution that supports the Subscribe/Notify communication pattern, as well as data requesting mode using XML and Web Services.

A. Decision Management System (DMS)

For many people big data is, erroneously, synonymous with the Hadoop framework [42]. But Hadoop does not have the ability to deal with streaming data, as is the case with IoT data. While IoT data has similar characteristics as big data,

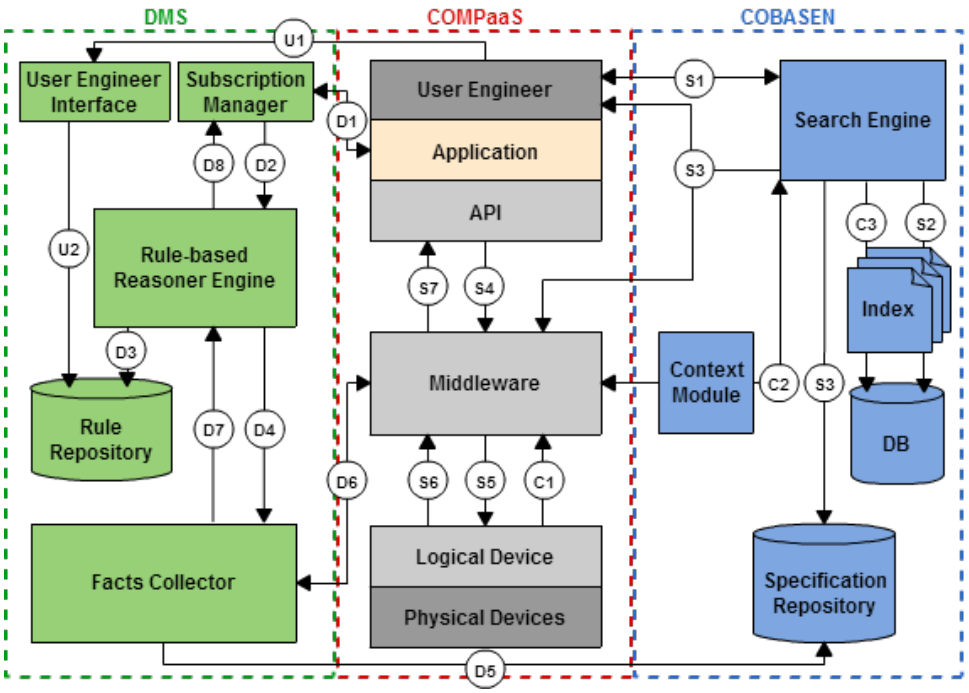


Fig. 2. Decision Managment System architecture.

IoT data is much more complex. One of the characteristics of IoT data is that it is "dynamic", in terms of "data in motion" as opposed to the traditional "data at rest". In this sense, DMS uses device data when it is still in motion, extracting valuable information from it through CEP to support decision making.

B. DMS Architecture

DMS architecture is an extension of COMPaaS (Section II-A) and COBASEN (Section II-B) architectures. DMS uses CEP to meet the needs of advanced users. It enables to trigger sophisticated reports to applications and enhance automation and efficiency in ubiquitous environment. To build an efficient and scalable system, a rule engine Drools [43] (mentioned in Section II-C) is used to quickly and reliably send notifications based on the business rules.

The heart of the system as shown in Fig. 2, is the *Rule-based Reasoner Engine* (RARE). RARE is responsible to load rules that are stored on the *Rule Repository*. The *Rule Name* and the *specifications IDs* are extracted from the rule base. The *Facts Collector* is used by the RARE to load the specifications on the COBASEN *Specification Repository*, and uses it to gather all the facts through COMPaaS. When all facts are collected, RARE uses the CEP engine to match the IF part of each rule against each asserted fact, executing the THEN part when a positive match occurs. When a positive match happens, *Subscription Manager* notifies all applications which are subscribers of that rule.

In this phase of the process (when a positive match occurs), in addition of feeding the application with the output data, DMS could take actions such as: acting through actuators registered in the middleware. However, for this work we are not managing the devices writing cycles, and this will be addressed in a future work. The system interactions are presented in Fig. 2. Each part of the DMS is described and explained bellow.

C. Rule Construction and Methods

As previously mentioned, RARE uses Drools Engine. Thus, the rule needs to follow its rule language. Moreover, in this section is described how the *User Engineer* can use the specifications previously created in COBASEN and some particular methods as the responsible to send the notification to the application.

The specification attributes that can be used in the rule is the specification *ID* and the specification *value*. The attribute *ID* is related to the ID given in the creation of the specification through *COBASEN*. The attribute *value* is the value that the *Facts Collector* module assigns to this specification *ID*. An example of the utilization of the specification attributes can be seen in Fig. 3.

The name of the rule is important and it will be used by the application to subscribe it (e.g., in Fig. 3 the name of the rule is *"Temperature Room 02 is higher than Room 01"*). When the *User Engineer* understands that the application needs to be notified when the rule is executed, the *Subscription Manager* needs to be used calling the method *notifyApp* in the THEN part of the rule. This method takes as parameter a string, and it can be used to describe the event or data related to the specifications. An example of its utilization can be seen in Fig. 3.

Let's try to explain it in a simple scenario where the *User Engineer* would like to his application be notified when the

```
rule "Temperature Room 02 is higher than Room 01"
        when
                $spec1  :   Specification ( )
                $spec2  :   Specification ( $spec1.id == 1, id == 2, value > $spec1.value )
                $submanager :   SubscriptionManager ( );
        then
                $submanager.notifyApp("Room 02 temperature is higher than the Room 01");
end
```

Fig. 3. Use of the data streams specifications in a rule.

temperature of the *Room 2* is higher than the *Room 1*. For that, consider that he already selected all temperature devices from the *Room 1* and *Room 2*, and generated the data streams specifications through COBASEN. In this case, let us consider that the COBASEN defined the specification ID for the *Room 1* as 1, and for the *Room 2* as 2. The resulted rule for this scenario is shown in Fig. 3.

D. User Engineer Interface

Users Engineer Interface is a User Interface (UI) which enables *Users Engineers* to upload rule files (.drl). As shown in Fig. 2, the rules are stored in the *Rule Repository*. This module also allows to the user engineer to manage the uploaded rules.

Prior to storing the rules is performed a validation to verify if the specifications IDs which were used in the rule are found in the COBASEN *Specification Repository*. In this way, the user can not deploy rules which uses data streams specifications which are not previously created.

It is not the intend of this work to provide any kind of rule editor, debugger or validator. The Drools Expert User Guide includes all background related to the Drools Rule Language and also presents some IDEs, APIs and Guided Editors that can be used to edit, debug and to validate the rules.

E. Subscription Manager

Subscription Manager is an interface that allows applications to subscribe and to unsubscribe the DMS rules. It also is responsible to notify the application.

To subscribe to DMS the application needs to sends a collection of names of the rules that it wants to subscribe, and the address of the Websocket that will be used to send back the notification. A rule can be subscribed by more than one application. In this way, when the method *notifyApp* is executed, all subscribers of the specific rule are notified.

To unsubscribe the application needs to sends a collection of names of the rules that it wants to unsubscribe and the Websocket address. Subscription Manager also is responsible to create in RARE the *Rule Lifecycles*. It is better explained in Section III-G.

F. Facts Collector

Facts Collector is responsible to gather all required facts to be inserted into the CEP engine working memory. Thus, when RARE sends a collection of data stream specifications IDs, *Facts Collector* creates a *Fact Lifecycle* for each ID. In this way, each *Rule Lifecycle* contains multiples *Fact Lifecycles* (according to the number of specifications used in the rule).

Moreover, when a *Fact Lifecycle* is started, it loads the related stored data stream specification in the COBASEN and starts the *Gathering Process*. The gathering process uses the specification and behave as an ordinal application that subscribes in the COMPaaS middleware to receive data (facts). The entire duration of the gathering process is assigned based on the highest specification duration.

The entire process is invalidated if any *Fact Lifecycle* was not successfully gathered its respective specification. In this case, RARE sends a report to application.

The following steps compose the *Gathering Process* using COMPaaS middleware: (1) create a connection (Web Socket) to receive back the data; (2) observe the connection to detect when receives new data; (3) send the specification XML defining which (devices) and how is desired the data; (4) makes the subscribe to starts receiving data.

When all *Fact Lifecycles* have completed the gathering process, then all facts are sent back to RARE.

G. Rule-based Reasoner Engine (RARE)

When all facts related to a rule are collected, RARE inserts them into the working memory. When all facts are inserted, it match the WHEN part of each rule against each asserted fact, executing the THEN part when a positive match occurs for all premises. The inference process continues until no more rules can fire.

An important case to reinforce, as explained in Section III-C, is that the *User Engineer* needs to declare the method *notifyApp* in the THEN part of the rule if, in this cases, he wants his application to be notified.

In the RARE is found the core of the system, the main *Lifecycle*. Each rule is related to a *Rule Lifecycle*. When an app subscribes a rule, the *Subscription Manager* creates a *Rule Lifecycle*, adds the application as observer, and starts it. In this way, there are a list of *Rule Lifecycles* and for each which is contained therein, there are a list of observers (used to know who needs to be notified when the *notifyApp* is fired). The *Rule Lifecycle* is responsible to coordinate all activities related to the inference process. That includes the cooperation with the *Facts Lifecycle* and others modules of the system.

H. DMS Sequence Flow

As previously mentioned, DMS architecture is an extension of COMPaaS and COBASEN architectures. The systems interaction are presented in Fig. 2. The steps (arrows) with letters “U” and “D” correspond to the DMS system. The steps with letter “C” are related to COBASEN (device discovery,

selection and data stream specification management). The steps with letter "S" correspond to COMPaaS middleware.

The U1 step, shown in Fig. 2, represents the user selecting the rule file. Only ".drl" files are allowed to be selected and uploaded. In U2 step, prior to storing the rules a rule validation is performed to verify if all the specifications IDs used in the rule file are found in the COBASEN *Specification Respository*.

The next steps are presented in Fig. 2: (D1) Application subscription; (D2) *Subscription Manager* creates the *Rule Lifecycle*, adds the application as observer, and starts it; (D3) RARE extracts the specifications IDs used in the rule; (D4) RARE sends a collection of specifications IDs to the *Facts Collector*; (D5) *Facts Collector* creates the *Fact Lifecycles* and each of it loads its related specification XML at the COBASEN; (D6) *Fact Lifecycles* gathers the facts; (D7) When all facts are collected, *Facts Collector* sends it back to RARE; (D8) RARE inserts these facts into the working memory and fire the rules. If the THEN part of some rule are executed and the *notifyApp* method are declared in it, the *Subscription Manager* notifies the subscribers.

IV. Implementation and Evaluation

We have developed a Java-based implementation of DMS, and its architecture is SOA. The Drools engine [38] version 6.3.0.Final was used as the rule-based reasoning component. We made experiments and analyzed the following features of the system: (A) Time taken to validate and extract the specifications IDs; (B) Time taken to start and load the system lifecycles in scenarios with different numbers of rules and specifications; and (C) How much DMS decreases the amount of messages transmitted over the network. We used a computer with AMD FX-8350 (8 core) 4.0GHz and 16GB RAM to evaluate DMS. The experiments were conducted in order to analyze the DMS performance, and each it were conducted 30 times and averages were considered in the results.

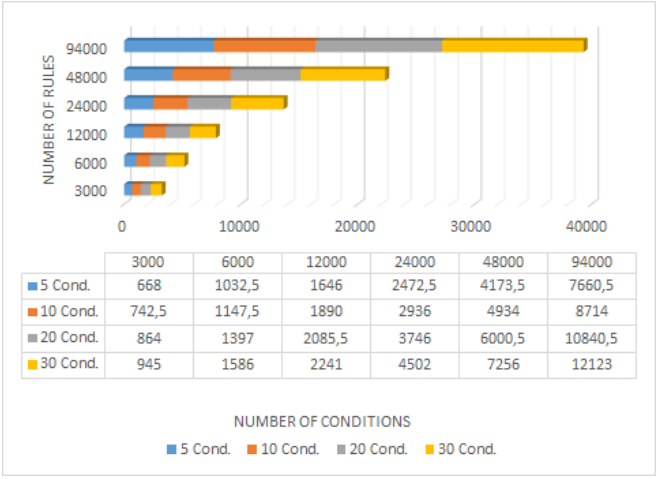
To perform the experiments we have defined a test plan with 24 different scales rule files. Each rule file has distinct scales in number of rules and number of conditions (IDs) per rule.

TABLE I
Rule validation function execution time (milliseconds).

Rule/IDs	5	10	20	30
3000	44	46	51	56
6000	77	79	94	106
12000	116	124	147	152
24000	129	133	158	190
48000	190	203	246	298
94000	351	348	434	485

A. Rule Validation

This experiment has been performed in order to test the rule validation process (step U2 shown in Fig. 2), which occurs after the upload of rule file (mentioned in Section III-D). The tests results are presented in Table I. The first column shows the number of rules contained in the rule file and the first row shows the number of conditions per each rule in the rule file.



	3000	6000	12000	24000	48000	94000
5 Cond.	668	1032,5	1646	2472,5	4173,5	7660,5
10 Cond.	742,5	1147,5	1890	2936	4934	8714
20 Cond.	864	1397	2085,5	3746	6000,5	10840,5
30 Cond.	945	1586	2241	4502	7256	12123

Fig. 4. DMS Overall Performance (milliseconds).

B. Overall System Performance

This experiment has been performed in order to verify and validate the steps D2, D3, D4, D5, and D7 of the system (steps presented Fig. 2). The time taken to gathering data though COMPaaS (step D6) is validated in [4], in this way, the gathering time is not taken into account. Moreover, in each test round we forced some applications to subscribe to all DMS rules. Thus, in a test round which the rule file contains 3000 rules and each rule contains 5 conditions per rule, the DMS was forced to react to all of it. In this case, the *Subscription Manager* was forced to create 3000 *Rule Lifecycles*, and for each of it the *Facts Collector* was forced to create 5 *Fact Lifecycles*, load these specifications in the COBASEN, subscribe these 5 specifications in the COMPaaS, and so on. The results of this experiment are presented in Fig. 4. The first column shows the number of rules contained in the rule file and the first row shows the number of IDs used in the conditions of the rule file.

The time taken to read the *rule base* only occurs in the initial charging of the system, and it are not taken into account in this experiment. However, it is presented in Table II.

TABLE II
Rule base (RB) reading time and rule base size

Rules	RB Reading Time (ms)	RB Size (bytes)
3000	16514	762,589
6000	35657	1,530,471
12000	77145	3,076,478

C. Network Exhaust

This experiment aims to verify how much DMS decreases the amount of messages transmitted over the network. Thus, we defined an IIoT scenario in which physical sensors were responsible to gather data readings from the field. The collected device data would allow the optimization of an industry process.

For that, an IoT software engineer has defined two data stream specifications containing six thermometer devices each. Thus, each middleware report was composed of the average of all six devices output and it had about 1,2KB. A new report was sent to the application every 1000 milliseconds.

There are two different scenarios in this experiment:

- Scenario 1: The user engineer uses COBASEN to search and select the desirable devices and create the data stream specification. The user engineer uses these specifications to gather the data directly on COMPaaS, and locally perform analytics.
- Scenario 2: The user engineer uses COBASEN as in the case 1, but in addition, he also defines a rule for this situation and subscribe to it in DMS. The rule is defined as follows: ***WHEN*** *the resulted value of the specification one is higher than 35 C°,* ***AND*** *the resulted value of the specification two is higher than 30 C°,* ***THEN notify*** *the application*.

To be able to evaluate the difference between these two scenarios we define three temperature variation ranges related to the industrial field: (1) Small variation: temperature ranges from 20 C° to 40 C°; (2) Medium variation: temperature ranges from 10 C° to 50 C°; (3) High variation: temperature ranges from 0 C° to 60 C°.

We have used an algorithm that simulates the temperature changes based on the pre-defined variations (low, medium, and high). For each experiment execution we have monitored the messages flow for 600 seconds.

Table III presents the number of messages needed to fulfill the scenario described above. The scenario 1 (2nd row) presents the number of messages towards the variations using COMPaaS directly communication. The scenario 2 (3th row) presents the number of messages towards the variations using DMS.

TABLE III
NETWORK EXHAUST EVALUATION

Scenario	Low Range	Medium Range	High Range
1	600	600	600
2	211	125	98

V. DISCUSSION

Today's device-driven world is forcing analytics to occur as fast as the data is generated. In this way, an important requirement for stream processing systems is to process messages "in-stream" as they fly by, without any requirement to store them to perform any operation or sequence of operations. Storage operations adds a great deal of unnecessary latency to the process (e.g., committing a database record requires a disk write of a log record).

DMS follows the *straight-through processing paradigm*, and incorporate event-driven processing capabilities. It avoid applications to continuously *poll* for conditions of interest, removing additional latency to the process, because (on average) half the polling interval is added to the processing delay [44].

By making use of validated architectures as COBASEN and COMPaaS, DMS provides improvements in the IoT application development process. The utilization of COBASEN, allows to inherits all its benefits related to discovery and selection of devices. On the other hand, COMPaaS is who are directly connected and interacting with devices, and also is responsible to collect data. More than that, DMS allows to apply immediate analytics insights from streams of data of devices into IoT applications.

As shown in Fig. 4 and Table I, DMS can operate with an inordinate number of rules. We verified that the system time grows more by the increase of rules than the increase of the number of conditions. In this way, in cases where the number of *Fact Lifecycles* are the same (e.g., 12000 rules each of it containing 5 conditions, and 6000 rules each of it containing 10 conditions, both requires 60000 *Fact Lifecycles*). In these cases, time has increased more for the scenario that has more rules, since higher number of rules require more processing to manage subscribers, among other functions.

Regarding results presented in Table III, DMS can reduces application processing overhead (i.e., application does not need to locally manage events) and makes it receives only relevant data. Furthermore, even in a scenario wherein the temperature is kept varying, and close to the rule condition, DMS reduces networking exhaust. The number of events and messages increase when the variation range occurs near the rule condition values. In this experiment, we have observed that DMS decreases 73,9% on average the amount of messages that are sent over the network from middleware to the application. DMS also prevents to store data that ultimately have no real value for future operations.

VI. CONCLUSIONS AND FUTURE WORK

With the fast proliferation of the IoT, things that seemed like science fiction 20 years ago are approaching the market. It means that the world around us will become filled with devices. We identified through literature that there are significant amount of middleware systems solutions for device data management, and integration. However, IoT becomes unfeasible without a support system to help us in the matter of how will we discover, identify, and interact with the IoT devices. The same occurs to stream analytics, to make the IoT devices useful, we need a system that support Analytics of Things.

The framework experimental results demonstrated that it is able to be used as an decision support system for streaming analytics towards scenarios with large numbers of devices, rules and facts. Further, it can reduces application processing overhead and networking exhaust. DMS allow streaming analytics and intelligence automated action on fast-moving data towards the pre-selection of "Things". The framework provides a new way to see how the business rules and decision-making will be made towards the Internet of Things.

In the future we are planning to expand DMS capabilities, going beyond send sophisticated reports to applications. In this

way, we are planning to start to work with devices writing cycles (e.g., actuators devices).

REFERENCES

[1] K. Ashton, "That internet of things thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer *et al.*, "Internet of things strategic research roadmap," *Internet of Things-Global Technological and Societal Trends*, vol. 1, pp. 9–52, 2011.

[3] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, E. C. D.-G. for the Information Society, and Media, *Vision and Challenges for Realising the Internet of Things*. Publications Office of the European Union, 2010.

[4] L. A. Amaral, R. a. T. Tiburski, E. de Matos, and F. Hessel, "Cooperative middleware platform as a service for internet of things applications," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 488–493.

[5] W. Lunardi, E. de Matos, R. Tiburski, L. Amaral, S. Marczak, and F. Hessel, "Context-based search engine for industrial iot: Discovery, search, selection, and usage of devices," in *IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8.

[6] Q. Jing, A. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the internet of things: perspectives and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2481–2501, 2014.

[7] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[8] R. Tiburski, L. Amaral, E. Matos, and F. Hessel, "The Importance of a Standard Security Architecture for SOA-based IoT Middleware," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 20–26, Dec 2015.

[9] A. Whitmore, A. Agarwal, and L. Da Xu, "The internet of thingsa survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[10] M. Hauswirth and K. Aberer, "Middleware support for the "internet of things"," *5th GI/ITG KuVS Fachgesprch "Drahtlose Sensornetze"*, 2006.

[11] A. Gómez-Goiri and D. López-de Ipiña, *A Triple Space-Based Semantic Distributed Middleware for Internet of Things*. Springer Berlin Heidelberg, 2010.

[12] Y. Huang and G. Li, "A semantic analysis for internet of things," in *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, vol. 1. IEEE, 2010, pp. 336–339.

[13] Z. Song, A. A. Cárdenas, and R. Masuoka, "Semantic middleware for the internet of things," in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[14] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of middleware for internet of things: A study," *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. 2, no. 3, pp. 94–105, 2011.

[15] J. A. Garcia-Macias, J. Alvarez-Lozano, P. Estrada-Martinez, and E. Aviles-Lopez, "Browsing the internet of things with sentient visors," *Computer*, no. 5, pp. 46–52, 2011.

[16] B. Ostermaier, K. Romer, F. Mattern, M. Fahrmair, and W. Kellerer, "A real-time search engine for the web of things," in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[17] Digital Enterprise Research Institute, "Linked sensor middleware (LSM)," https://www.deri.ie/, accessed: 2016-01-03. [Online]. Available: https://www.deri.ie/

[18] D. Le-Phuoc, H. N. M. Quoc, J. X. Parreira, and M. Hauswirth, "The linked sensor middleware–connecting the real world and the semantic web," *Semantic Web Challenge*, vol. 152, 2011.

[19] GSN Team, "Global Sensor Network," http://sourceforge.net/apps/trac/gsn/, accessed: 2016-01-03. [Online]. Available: http://sourceforge.net/apps/trac/gsn/

[20] LogMeIn, "Xively," http://xively.com/, accessed: 2016-01-03. [Online]. Available: http://xively.com/

[21] S. Nath, J. Liu, and F. Zhao, "Sensormap for wide-area sensor webs," *Computer*, vol. 40, no. 7, pp. 90–93, 2007.

[22] H. Wang, C. Tan, and Q. Li, "Snoogle: A search engine for pervasive environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 8, pp. 1188–1202, Aug 2010.

[23] C. C. Tan, B. Sheng, H. Wang, and Q. Li, "Microsearch: A search engine for embedded devices used in pervasive computing," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 4, p. 43, 2010.

[24] K.-K. Yap, V. Srinivasan, and M. Motani, "Max: Human-centric search of the physical world," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 166–179. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098937

[25] W. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *IEEE MultiMedia*, vol. 14, no. 4, pp. 8–13, Oct 2007.

[26] Z. Ding, X. Gao, L. Guo, and Q. Yang, "A hybrid search engine framework for the internet of things based on spatial-temporal, value-based, and keyword-based conditions," in *IEEE International Conference on Green Computing and Communications (GreenCom)*, Nov 2012, pp. 17–25.

[27] D. Wang, M. Zhou, S. Ali, P. Zhou, Y. Liu, and X. Wang, "A novel complex event processing engine for intelligent data analysis in integrated information systems," *International Journal of Distributed Sensor Networks*, vol. 2016, 2016.

[28] D. C. Luckham and B. Frasca, "Complex event processing in distributed systems," *Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford*, vol. 28, 1998.

[29] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "A knowledge-based platform for big data analytics based on publish/subscribe services and stream processing," *Knowledge-Based Systems*, vol. 79, pp. 3–17, 2015.

[30] L. Yao-Zong and H. Fa-Wang, "Rfid complex event processing for rtls," in *Multimedia Information Networking and Security (MINES), 2012 Fourth International Conference on*. IEEE, 2012, pp. 392–395.

[31] Y. Wang, K. Cao, and X. Zhang, "Complex event processing over distributed probabilistic event streams," *Computers & Mathematics with Applications*, vol. 66, no. 10, pp. 1808–1821, 2013.

[32] R. Baldoni, L. Montanari, and M. Rizzuto, "On-line failure prediction in safety-critical systems," *Future Generation Computer Systems*, vol. 45, pp. 123–132, 2015.

[33] B. Balis, B. Kowalewski, and M. Bubak, "Real-time grid monitoring based on complex event processing," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1103–1112, 2011.

[34] D. Wang, E. A. Rundensteiner, H. Wang, and R. T. Ellison III, "Active complex event processing: applications in real-time health care," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1545–1548, 2010.

[35] I. Zappia, F. Paganelli, and D. Parlanti, "A lightweight and extensible complex event processing system for sense and respond applications," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10 408–10 419, 2012.

[36] C. Zang and Y. Fan, "Complex event processing in enterprise information systems based on rfid," *Enterprise Information Systems*, vol. 1, no. 1, pp. 3–23, 2007.

[37] "Apache Storm," http://storm.apache.org/.

[38] W. Yao, C.-H. Chu, and Z. Li, "Leveraging complex event processing for smart hospitals using rfid," *Journal of Network and Computer Applications*, vol. 34, no. 3, pp. 799–810, 2011.

[39] "Jess," http://www.jessrules.com/.

[40] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "A model-driven approach for facilitating user-friendly design of complex event patterns," *Expert Systems with Applications*, vol. 41, no. 2, pp. 445–456, 2014.

[41] R. Bruns, J. Dunkel, H. Masbruch, and S. Stipkovic, "Intelligent m2m: Complex event processing for machine-to-machine communication," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1235–1246, 2015.

[42] Apache Software Fundation, "Apache Hadoop," https://hadoop.apache.org/, accessed: 2016-01-05. [Online]. Available: https://hadoop.apache.org/

[43] M. Bali, *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing, 2009.

[44] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.