Intelligent Event Broker: A Complex Event Processing System in Big Data Contexts

Completed Research

Carina Andrade
ALGORITMI Research Centre,
University of Minho,
Guimarães, Portugal
carina.andrade@dsi.uminho.pt

José Correia
ALGORITMI Research Centre,
University of Minho,
Guimarães, Portugal
josecorreia@dsi.uminho.pt

Carlos Costa
ALGORITMI Research Centre,
University of Minho,
Guimarães, Portugal
carlos.costa@dsi.uminho.pt

Maribel Yasmina Santos
ALGORITMI Research Centre,
University of Minho,
Guimarães, Portugal
maribel@dsi.uminho.pt

Abstract

In Big Data contexts, many batch and streaming oriented technologies have emerged to deal with the high valuable sources of events, such as Internet of Things (IoT) platforms, the Web, several types of databases, among others. The huge amount of heterogeneous data being constantly generated by a world of interconnected things and the need for (semi)-automated decision-making processes through Complex Event Processing (CEP) and Machine Learning (ML) have raised the need for innovative architectures capable of processing events in a streamlined, scalable, analytical, and integrated way. This paper presents the Intelligent Event Broker, a CEP system built upon flexible and scalable Big Data techniques and technologies, highlighting its system architecture, software packages, and classes. A demonstration case in Bosch's Industry 4.0 context is presented, detailing how the system can be used to manage and improve the quality of the manufacturing process, showing its usefulness for solving real-world event-oriented problems.

Keywords

Big Data, Complex Event Processing, Rules Engine, Machine Learning.

Introduction

Nowadays, data is generated with high frequency due to, for instance, the internet and the proliferation of devices that constantly connect people and make it possible to buy anything, anywhere, generating a vast amount of transactions. Furthermore, data is not only associated with business transactions, since, with the Industry 4.0 movement (Kagermann et al. 2013), products will be connected, machines with sensors will give feedback about their working status, and decisions will be made, in the shop floor, by the intelligent systems that anticipate the events and propose actions based on business rules or decision models. This phenomenon provides a vast amount of data generated at even-increasing velocities, which should be significantly valuable for organizations. Addressing this value is the main motivation of this work, making possible the proliferation of the most adequate decisions as soon as the data is available inside the organizations (e.g., Databases and IoT platforms), benefiting various business contexts.

Thereby, this proposal arises from Bosch's requirements together with a gap found in the literature regarding the existence of a common approach to integrate CEP in the Big Data era. The Design Science Research Methodology for Information Systems with an objective-centered approach (Peffers et al. 2007) is used in the overarching research process in which this work fits in, supporting the design of a CEP system integrated in Big Data contexts. Moreover, this type of system is relevant for several organizational and

social contexts, such as industry, smart cities, agriculture, among others that have several data sources available, being helpful for the monitoring of business processes and events, and for preventing possible problems through its real-time processing capabilities. Therefore, this system must meet the following high-level criteria: i) handle Big Data produced by several sources inside and outside the organization (e.g., production line, cars, and transactional systems); ii) process the data within the time frame needed for several decision-makers; iii) provide predictions and recommendations based on the data, rules, events, and indicators being processed; iv) autonomously execute adequate actions to avoid considerable problems (e.g., stop a production line machine that is producing several defective products in a continuous manner); and, v) should have capabilities of self-management and control, allowing the constant monitoring and visualization of what happened in the CEP system. Thus, the artifact proposed in this work reflects the first iteration of the Design and Development phase, using a proof-of-concept based on the Active Lot Release (ALR) application from Bosch Car Multimedia Portugal as a demonstration case. With this first iteration, the objectives i, ii, and iv where fulfilled, and the solution reveals an adequate effectiveness, considering the success of the presented proof-of-concept. The efficiency and scalability of the solution, although it is already a concern of the current architecture and proof-of-concept, will be rigorously evaluated in future iterations of the overarching research process.

This document is structured as follows: the second section presents the related work; the third section introduces the proposed system architecture and its software packages and classes; the fourth section presents a demonstration case based on data from Bosch's ALR application in the Braga plant shop floor in Portugal; the fifth section presents the conclusions for this work and some prospects of future work.

Related Work

CEP systems are considered as an evolution of the Active Database Systems and the Data Stream Management Systems (DSMSs), being all of them covered by the Information Flow Processing concept (Cugola and Margara 2012). These CEP systems aim to find patterns in different events that arrive from different sources in a way that all the stakeholders can be notified, as soon as possible, with the processing results (Cugola and Margara 2012).

The work of (Chakravarthy and Qingchun 2009) considers that the current capability to collect and process data, continuously generated from multiple sources, created the need for new types of applications. Focused on the perspective of quality of service related to DSMSs, it introduces several domains for these systems, being CEP identified as a challenge for them, once it is important to not only deal with the data in real-time as soon as it arrives at the system, but also to try to find patterns and trigger actions based on previously defined business rules. The authors mention that DSMSs with and without the CEP component are already available, as well as other CEP systems that do not have streaming capabilities.

Regarding the first published work that can be classified as a CEP system, Rapide (Luckham 1996; Luckham and Vera 1995) is often unanimously considered as being the first one that explores this concept (Cugola and Margara 2012; Leavitt 2009), which is a project started in the 90s that provides to users the capability to identify the temporal and causal relationship among events. Since then, new systems classified as CEP have emerged (Cugola and Margara 2012), some of them are open-source (mostly used by academics), while others are commercial tools for Business Intelligence (Tawsif et al. 2018).

The tutorial presented in (Giatrakos et al. 2017) focuses in the Big Data and Complex Event Recognition concepts, being this last one the identification of simple events that together are interesting once they meet certain patterns. The authors aim to provide a guide for the use of event streams that achieve the Big Data characteristics (volume, velocity, and variety), performing Complex Event Recognition in a distributed way.

The BiDCEP architecture is proposed in (Hadar 2016), based on the Lambda and Kappa architectures, integrating the Big Data streaming and CEP concepts. This architecture is divided into several components with different responsibilities: the ones responsible for the connection to the data sources; the ones responsible for adding more value and to filter the data considering the business needs; and, the CEP component that triggers actions to control consumer applications. The authors also present a motivational example that aims to explain a context in which the system can be applied. In this example, the authors refer that the use of IoT should be considered as an enabler for descriptive, predictive, and prescriptive analytics, although it is not noticeable where these aspects are considered in the proposed architecture.

Another architecture is presented in (Flouris et al. 2016) together with the FERARI prototype for real-time CEP with a vast amount of event data streams processed in a distributed way. The proposed architecture considers components like producers, consumers, and event processing agents for different event types. This work also refers a visualization component for dashboards and reports. The work of (Flouris et al. 2017) is focused on CEP issues related to Big Data, mainly focusing on hardware requirements. Therefore, the authors propose the use of cloud computing platforms, referring the main properties that should be considered in a Big Data CEP: parallelism, elasticity, multi-query, and distributed resources. In addition, the authors show that some works consider these four mentioned features, although concluding that the integration of CEP and Big Data technologies is still significantly unexplored.

In this context, other works (Babiceanu and Seker 2015; Krumeich et al. 2014) emphasize the importance of combining Big Data, CEP, and IoT, in this case, to support the manufacturing industry through Cyber-Physical Systems (CPSs) (Dundar et al. 2016). The QuantCloud architecture (Zhang et al. 2018) also aims to connect the CEP concept with Big Data, as well as the work of (Saenko et al. 2017) that presents an architecture divided into four components: data collection; data storage; data normalization and analysis; and, data visualization.

Some of the main points presented in the related architectures are also shared by the architecture proposed in this paper. However, to the best of our knowledge, the one here proposed provides further details about the data processing component of the CEP system for Big Data, thoroughly explaining how the processing of the events, rules, and triggers occurs, also considering aggregations and Key Performance Indicators (KPIs) calculated in real-time. Furthermore, once ML was identified as a major gap in these systems (Tawsif et al. 2018), the Machine Learning Models Lake (MLML) component can be significantly helpful for patterns discovery. On the other hand, the use of batch data available in the Big Data Warehouse (Costa and Santos 2018) is also relevant to increase the value of the results produced by the CEP system proposed in this work. In addition, there is no architecture concerned with the monitoring of a CEP system and its evolution, as it can quickly become untraceable in Big Data contexts, being this one of the core components of our work. Finally, the proposed architecture, software packages, and classes embody a physically implemented system presented in a detailed manner, so that other practitioners can follow the design and development guidelines, which is seen as a valuable contribution that is not frequently seen in other related works available in the emerging and scarce community related to CEP systems in Big Data contexts.

Intelligent Event Broker

The Big Data-oriented CEP system here proposed is a collection of several software components and data engineering decisions that are integrated and validated to function harmoniously. This section presents the design and development decisions made in this research work, including the system architecture (Figure 1) and the structure of software packages and classes (Figure 2) of the Intelligent Event Broker.

System Architecture

A Big Data-oriented CEP system should be able to collect and process data from an extensive variety of source systems, no matter their underlying communication interfaces. Depending on the implementation context, *Events* can be produced or stored in several systems, including *IoT gateways, Web servers, databases (e.g., SQL, NewSQL and NoSQL)*, and Hadoop-related components such as *Hive* or *HDFS*. In order to standardize the collection of *Events* in the system, we propose the deployment of an adequate event-oriented system backed up by *Kafka*, a distributed streaming platform (Kafka 2018), which can be used to publish *Events* in topics, through *Kafka Producers*, and to further develop *Kafka Consumers* that subscribe these topics. In the proposed system, *Kafka* supports the backbone of *Events* collection and dissemination. *Events* are collected from the corresponding *Source Systems* using *Kafka Producers* developed for this purpose, which can include applications developed in any programming language having available a *Kafka* client implementation (e.g., Java or C/C++), or a *Spark Application* that connects to the *Source Systems* using the available connectors. These two types of *Producers* collect the data, serialize it into the form of *Broker Beans* (simple classes representing business entities and information), and produce the *Events* through their publication into a *Kafka* topic stored in a cluster of *Kafka Brokers*.

The topics containing the previously published *Events* are subscribed by the *Event Processor* through the form of several *Kafka Consumers* embedded into *Spark Applications* that are continuously listening for the

arrival of the *Events*, no matter their frequency and quantity. *Spark Streaming Applications* that are constantly processing huge amounts of *Events* arriving at higher frequencies will require more cluster resources than more moderate workloads (e.g., a few *Events* per hour), but they can share a streaming-oriented computing cluster. The *Event Processor* is, therefore, one of the core components of the system, being tightly coupled with the *Rules Engine* that encapsulates all the business requirements in the form of *Strategical Rules* (e.g., market attractiveness indicators/suggestions), *Tactical Rules* (e.g., supply chain management indicators/warnings), and *Operational Rules* (e.g., stopping a production line due to repetitive failures) for specific implementations. In this context, *Drools* can be used as the *Rules Engine* (Drools 2018), wherein *Data Engineers* translate the business requirements into a series of *Rules* stored in *Drools* files (*Rules Repository*), which are then transparently translated at runtime by the *Event Processor*.

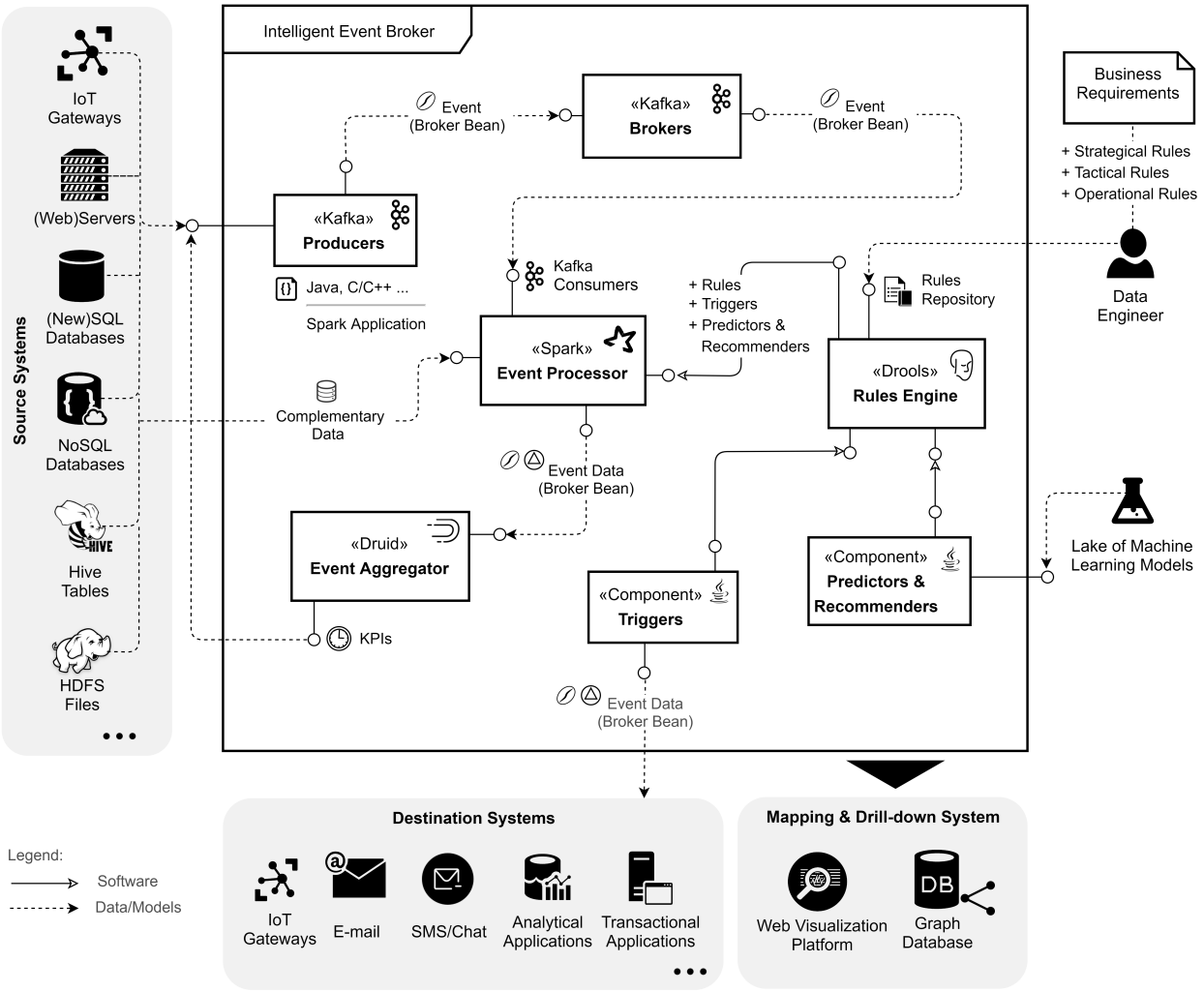

Figure 1. System architecture

At any moment, the *Event Processor* can take as input *Complementary Data* from any *Source System*, as *Spark Applications* can connect to these sources for querying historical or real-time data. This *Complementary Data* can be used to provide a richer additional context for *Business Rules* and *Events*, as the following business rule demonstrates: "when a defective part is detected in the production line, if in the last 10 minutes there were more than 3 defective parts, send a message to the operational manager". Considering this, the *Rules Engine* and the encapsulated business logic also embed the following software components: i) *Triggers*, which represent the connectors to several *Destination Systems*, i.e., after a condition in a certain rule is evaluated as being true, these *Triggers* perform certain actions according to the defined rule's consequence. For example, *Destination Systems* may include an *IoT gateway* that activates an actuator, a *Text* or *E-mail Message*, and *Transactional* and *Analytical Applications* that are

used to support daily operations or the decision-making process (via the database layer or via direct push mechanisms). This component can output: 1) the processed data related to the *Events* in the form of a new *Broker Bean*, if more data value is added after processing the *Event* according to the defined *Rules* and business logic; 2) the raw *Event* data in the form of a previously defined *Broker Bean*, if it is not relevant to include after-processing data (e.g., consequences of rules execution); and, ii) *Predictors and Recommenders*, which assure the capability of interpreting previously trained ML models and use them to predict occurrences and recommend actions considering the *Events* that are being processed by the system (e.g., predict the likelihood of a continuous failure in the production line, given the production *Events* for the past 5 minutes). These models are stored in the MLML, which may contain several file formats depending on the implementation details (e.g., Predictive Model Markup Language - PMML - or pickle files). In this work, we consider that the *Lake* can be implemented using any local or distributed file system, and the models can be accessed through Web services that, given certain features, predict the corresponding occurrence or recommend the corresponding action. Through the development of scalable Web services, the ML models can be adequately invoked in *Drools* files.

Besides the *Event Processor*, the Intelligent Event Broker also includes an *Event Aggregator*, supported by *Druid*, a columnar storage system that is able to aggregate *Event* data (Yang et al. 2014) with sub-second response times over huge amounts of data (Correia et al. 2018). This component takes as input either raw *Event* data (previously defined *Broker Beans*) or processed *Event* data (new *Broker Bean*), as previously explained in the *Triggers* component. Taking into consideration the input data, the *Event Aggregator* is responsible for ingesting the data, perform the aggregations as the ingestion takes place, and store them to calculate the *Key Performance Indicators (KPIs)* that are relevant for a particular business context. This calculation step takes place in other *Kafka Producers* that will periodically send *KPI* updates to specific *Kafka* topics, generating *Events* that will be consumed by the *Event Processor* as soon as they occur. Consequently, *KPI* information, in the form of an *Event* (through a *Broker Bean* representation) is also a relevant part of the *Rules* that are interpreted by the *Rules Engine*.

Due to the complexity associated with running the Intelligent Event Broker in production environments, we need to deploy adequate mechanisms that allow for the constant and long-term monitoring of the system's daily operations. This goal is achieved through the development and deployment of the *Mapping and Drill-down System*, which is composed of the following components: i) a *Graph Database* built upon the analysis and indexing of the *Rules Repository* (*Drools* files) and the Intelligent Event Broker codebase (e.g., Java files), as well as appropriate runtime logging mechanisms, in order to store all the relevant metadata regarding the system's structure (e.g., *Broker Beans, Rules, Producers, Consumers, Triggers, and Predictors and Recommenders*), functionality (e.g., the *Rules* that were triggered and the data that was processed), and performance (e.g., number of *Events* processed per minute); and, ii) a *Web Visualization Platform* fueled by the previously described *Graph Database*, which allows for an interactive and intuitive navigation through the Intelligent Event Broker metadata, in order to retrieve useful insights regarding the CEP scenarios related to a particular implementation context.

Software Packages and Classes

In terms of source code structure, the Intelligent Event Broker is composed of seven software packages, each one corresponding to a Maven module of the top-level project. Despite the adoption of specific technologies for the implementation of the Intelligent Event Broker, this work aims to provide general design guidelines for the development of Big Data CEP systems and, therefore, the software packages and classes depicted in Figure 2, as well as the components in Figure 1, should be seen as general constructs that can be easily adapted and extended to future implementations of similar systems.

Regarding the *Producers* package, the same is mainly based on a set of *Kafka Producers*, whose implementation of the main method varies depending on the specific data collection mechanism. In contrast, the *Consumers* package is composed of a class that is responsible for configuring a *Generic Spark Kafka Consumer*, providing a standard state and behavior for other child classes (*Specific Spark Kafka Consumer*) that will then provide particular implementations to adequately read a specific data stream (*Kafka* topic). The specific *Consumers* are the ones used in *Spark Applications*, one for each business goal (or group of goals). The pool of *Spark Applications* deployed at a specific moment form the *Event Processor*, as described in the previous section.

The *Rules Engine* package is used by the *Consumers* package, since the latter embeds the first during execution, encapsulating all the business logic defined in the *Rules Repository*. The *Rules Engine* package is mainly composed of two classes: the *Rules Engine* class assuring a connection to a *Drools* runtime environment that will scan the current *Rules Repository*; and, a *Rules Stateless Session*, which is an encapsulated *Drools Stateless* Session that can be used in any consumer by invoking the *"getStatelessSession"* method of the *Rules Engine* with the proper session name (set of *Business Rules* in the *Rules Repository* to apply in a specific *Consumer*, i.e., a set of *Drools* files). A *Rules Stateless Session* contains methods to execute all the *Rules* of the respective session, to add *Global Variables*, add *Triggers* (also treated internally as *Global Variables*), and to close the *Session* (invokes the "*close*" method in each *Trigger* used in this *Session*, e.g., database connections). These *Triggers* make up the *Triggers* package, wherein each *Specific Trigger* implements the *Trigger* interface, defining the behavior of each *Trigger*, namely the need to implement the "*fire*" method (execute the trigger at any given moment) and the "*close*" method. *Each Specific Trigger* also uses a *Specific Connection Factory* that, implementing the *Connection Factory* interface, is able to provide a connection to a *Destination System* (see Figure 1).

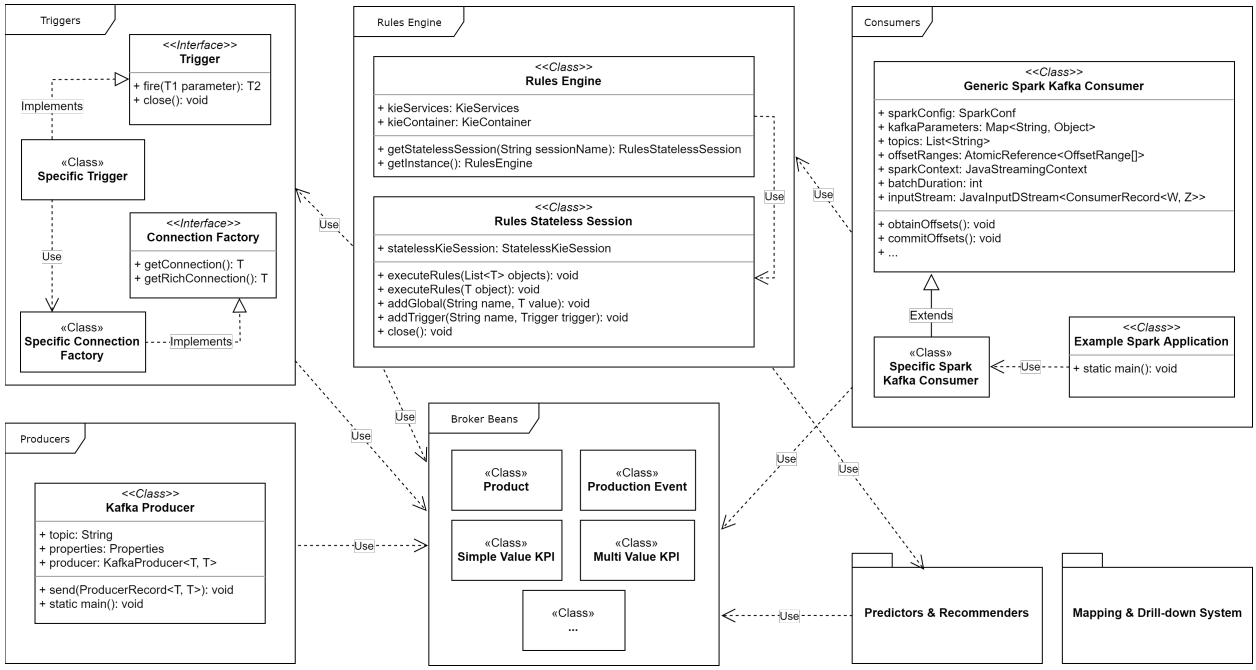

Figure 2. Diagram of software packages and classes

The *Producers, Consumers, Rules Engine*, and *Triggers* packages use the *Broker Beans* package to adequately represent business entities, assuring that *Events* and their corresponding data have the same meaning throughout the CEP system. Another package using *Broker Beans*, and being used by the *Rules Engine*, is the *Predictors* and *Recommenders* package containing the ML models that provide intelligence to the proposed Big Data CEP system, as these models will be applied to the data related to the *Events* (represented as *Broker Beans*). In order to conclude the description of the Intelligent Event Broker system, the *Mapping and Drill-down System* package implements an overarching set of features that allow for the adequate monitoring and analysis of the system's functioning. This package, together with the *Predictors and Recommenders* package, are not thoroughly detailed in Figure 2 due to the fact that their development is currently being initiated, thus constituting a perspective for future research work. These are seen as relevant design elements of the top-level project, reason why they are presented and described in this work.

Demonstration Case – Bosch Active Lot Release

The demonstration case presented in this section is based on data from the Bosch ALR system. This system is responsible for supporting quality control during the manufacturing and packaging processes in the factory. Summarizing, ALR applies a pre-defined set of rules to several distinct products contained in a lot, before shipping it to the final client. When these rules are verified for all the products in a lot, the same is

marked as *valid* and it can be shipped. If the lot does not comply with the rules, it needs to be repaired and resubmitted to the quality control process. The ALR system was created to solve a previously observed problem: the lots were created as *valid* (by default) before the use of this quality control process, which caused unnecessary costs, because after delivering the lots for shipping, if the quality department detected problems, the lots had to return to the factory. Besides this, the data about defective products (and lots) was not stored, making impossible to monitor or analyze past production problems. These issues are solved by the ALR system, as it stores this real-time data in a distributed data storage, allowing the decision-makers to analyze this information and act accordingly. The workflow from production to shipping, before and after the ALR implementation, is schematized in Figure 3.

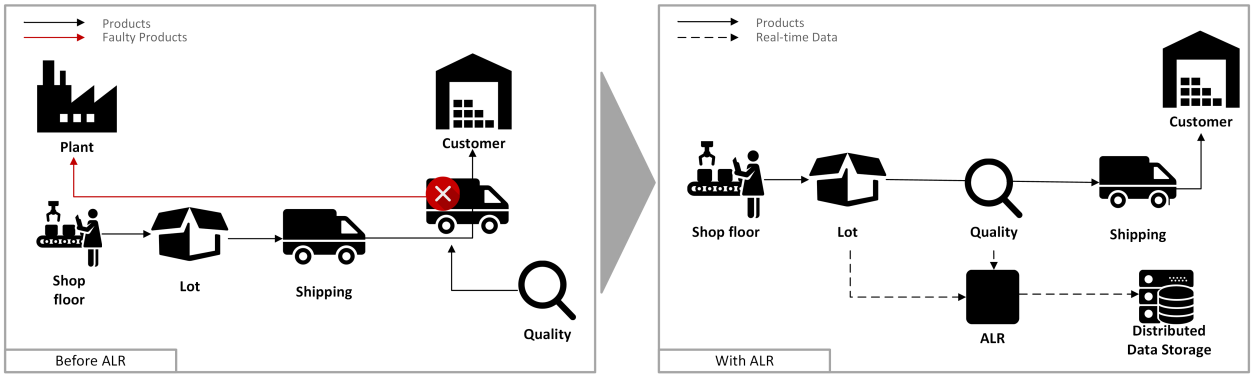

Figure 3. Quality control before and after the ALR implementation

Focusing on the description of the demonstration case, the system implemented in this context is entirely based on the architecture depicted in Figure 1. As previously mentioned, the data comes from Bosch, thus it is important to implement some privacy policies. To accomplish this, the developed *Kafka Producer* uses a subset of the original ALR data (represented as an *IoT Gateway* in Figure 1), shuffling the data and updating its temporal fields before providing this data to *Kafka Consumers* (one message per second). As soon as this data is available through a topic in the *Kafka Broker*, it will be used by several *Consumers*. In this demonstration case, there are 3 *Kafka Consumers* to be considered: i) *ALR Raw Data Consumer* - a) consumes real-time data (*Events*); b) applies all the necessary transformations to assure data quality; and, c) stores this data in Druid, in order to be immediately accessible; ii) *ALR Operational Consumer* - a) consumes real-time data (*Events*); b) applies all the necessary transformations to assure data quality; and, c) applies the operational *Rules* to each *Event*, activating the corresponding *Triggers* when the *Rule* is verified; and, iii) *ALR Analytical KPIs Consumer* - a) consumes data (*Events*) previously stored in Druid; and, b) applies the several *Tactical Rules* to the resulting *Events*, activating the corresponding *Triggers* when the *Rule* is verified.

This demonstration case implements two *Triggers*: i) a *Mail Trigger*, which sends an *E-mail Notification* to the related stakeholders; and, ii) a *Cassandra Trigger*, which stores the *Events* in this database for future analysis using an *Analytical Application* (Tableau (Tableau 2018) is used for this demonstration case). *Triggers* execution and the subsequent analysis, either by *E-mail Notification* or through the *Analytical Application*, are important for decision-makers to be aware of some unusual or unexpected behavior, and to rapidly analyze this data and act accordingly.

Regarding *Rules*, as previously mentioned, it is important to highlight that there are two different types of *Rules* used in this demonstration case, namely *Operational Rules* and *Tactical Rules*. The first type includes "*Lot State = 'INVALID'*", which activates the *Triggers* when a lot is invalid after the quality control verification. The second type includes: i) "*Num Invalid Lots by Day > x*", which activates the *Triggers* when the number of invalid lots in a day exceeds a predefined threshold; ii) "*Num Invalid Lots by Day & Line > Avg Line Last Week*", which activates the *Triggers* when the number of invalid lots of a specific production line in a day exceeds the average number of invalid lots of that production line in the last week.

Rules and *Triggers* are two important aspects in this system, therefore it is relevant to thoroughly explain how the *Consumers*, *Rules,* and *Triggers* are integrated and what are the tasks of each one. In order to illustrate the workflow from the *Rules* definition to the *Rules* execution, Figure 4 presents the definition of a *Rule*, its invocation in the *Event Processor,* and the *Triggers* execution.

The rule presented in the right side of Figure 4 corresponds to the *Rule* "*Num Invalid Lots by Day > x*" presented above, and it verifies the value of a *KPI* named "*Invalid Lots By Day*", which is a *Tactical Rule* executed in the *ALR Analytical KPIs Consumer*. As previously explained, this rule evaluates if the number of invalid lots by day is higher than a threshold (100 in this case) and, if so, it executes the *Mail* and *Cassandra Triggers*.

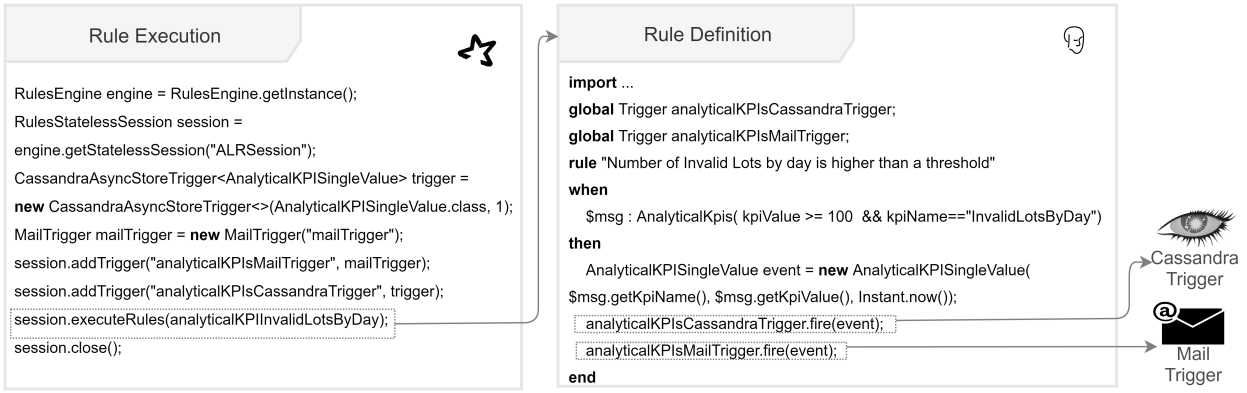

Figure 4. Example of the Rules definition and execution

This workflow is executed periodically (e.g., every day), submitting a query to *Druid* and sending the result via a *Kafka Producer* ("*KPIs*" in Figure 1). After this, the *Kafka Producer* broadcasts this data to the *Consumers* through a topic in the *Kafka Broker*. In this demonstration case, the only *Consumer* interested in this data is the *ALR Analytical KPIs Consumer*, which consumes the data, does the necessary transformation to interpret it, and executes the *Rules*. In the left side of Figure 4, we can observe how simple it is to embed the *Rules Engine* into the *Event Processor* (set of *Spark Consumers*), in order to execute *Rules*. It is only necessary to: *i)* get a *Rules Engine* instance and a *Session*; *ii)* instantiate and add the necessary *Triggers* to the previously created *Session*; *iii)* invoke the execution of the previously defined *Rules* according to the consumed *Events*; and, finally, *iv)* close the *Session*.

As already mentioned, for this demonstration case, two types of rules were implemented (tactical and operational). Taking this into account, two types of dashboards (Figure 5) were developed to show different kinds of analysis for the different levels of the decision-making process. Before presenting the dashboards, it is important to recall that due to confidentiality reasons, all the data here presented is fictitious.

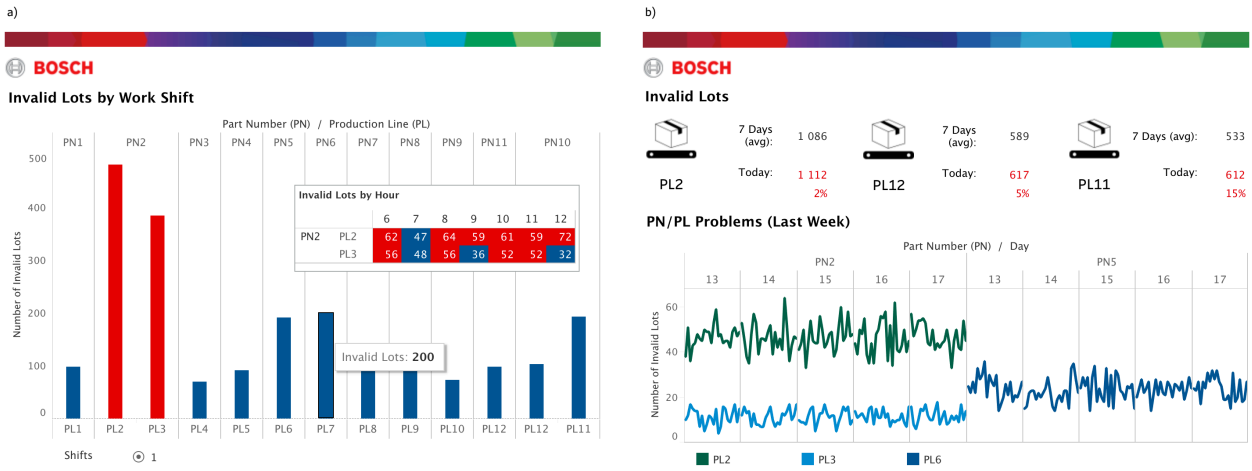

Figure 5. Example of Operational and Tactical Dashboards

Taking this into consideration, Figure 5a) shows a dashboard that can be displayed in the production line (PL), showing, for a specific work shift, the evolution of the invalid lots in the several PLs grouped by part number (PN). This type of dashboard provides to the PL manager the capability to take a specific action in the PL (or in the PN being produced) as soon as the quantity of invalid lots increases, having a constant

vision of the PL status (and, in this specific case, the problematic PN in production). Furthermore, the exact number of invalid lots in each hour already completed in the current work shift can be tracked for each PN. On the other hand, Figure 5b) presents the KPIs based on the difference between the number of invalid lots produced in a specific PL and day and the average of invalid lots produced in the same PL for the last seven days, being also illustrated the percentage of increase when compared with the average value. Moreover, to have a global view on the performance of the previous week, a chart with the three PLs with more production problems is also presented in this dashboard, also detailing their daily evolution regarding the number of invalid lots.

Conclusion

This paper presented the Intelligent Event Broker, a CEP system based on Big Data techniques and technologies (e.g., *Spark*, *Druid,* and *Kafka*), as well as on a MLML concept. A Bosch Car Multimedia Portugal demonstration case was also presented in this work, where the Intelligent Event Broker is used in the context of the manufacturing and packaging quality control, including the processing of *Tactical* and *Operational Rules* based on *KPIs* and raw *Event* data. The system architecture and software packages and classes shared in this work with the scientific and technical community are seen of major relevance for the advancement of CEP systems in the era of Big Data. In this work, the integration of the CEP and Big Data concepts is made through the use of a rule engine over Big Data technologies that provides an environment of scalability and distributed processing. Furthermore, in addition to the basic CEP components that were considered, we further propose components that fill other gaps identified in the literature, including the MLML for the predictions and recommendations, and the component that will monitor the evolution of the Intelligent Event Broker through the *Mapping and Drill-down System*.

The contributions of this paper help researchers and practitioners in the development of systems based on the Intelligent Event Broker logical components, and in the exploration of new ways of combining Big Data and CEP systems to build innovative data-based analytical systems, significantly relevant considering the number of devices and data sources available in current organizational contexts. For future work, as part of the followed research process, a benchmark is planned to access the response times, throughput, resource usage, and scalability of the system, as well as the extension of the work to address the objectives iii and v, namely by implementing and evaluating the ML capabilities and the *Mapping and Drill-down System*.

Acknowledgements

This work has been supported by FCT – *Fundação para a Ciência e Tecnologia* within the Project Scope: UID/CEC/00319/2019 and the Doctoral scholarship PD/BDE/135101/2017. This paper uses icons made by Freepik, from www.flaticon.com.

REFERENCES

Babiceanu, R. F., and Seker, R. 2015. "Manufacturing Cyber-Physical Systems Enabled by Complex Event Processing and Big Data Environments: A Framework for Development," in Service Orientation in Holonic and Multi-Agent Manufacturing, Studies in Computational Intelligence, Springer International Publishing Switzerland 2015, pp. 165–173. (https://doi.org/10.1007/978-3-319-15159-5_16).

Chakravarthy, S., and Qingchun, J. 2009. Stream Data Processing: A Quality of Service Perspective: Modeling, Scheduling, Load Shedding, and Complex Event Processing, Advances in Database Systems, Springer US. (//www.springer.com/la/book/9780387710020).

Correia, J., Santos, M. Y., Costa, C., and Andrade, C. 2018. Fast Online Analytical Processing for Big Data Warehousing, presented at the International Conference on Intelligent Systems, Madeira Island, Portugal, September.

Costa, C., and Santos, M. Y. 2018. "Evaluating Several Design Patterns and Trends in Big Data Warehousing Systems," in Advanced Information Systems Engineering, Lecture Notes in Computer Science, J. Krogstie and H. A. Reijers (eds.), Springer International Publishing, pp. 459–473.

Cugola, G., and Margara, A. 2012. "Processing Flows of Information: From Data Stream to Complex Event Processing," ACM Comput. Surv. (44:3), 15:1–15:62. (https://doi.org/10.1145/2187671.2187677).

Drools. 2018. "Drools - Business Rules Management System." (https://www.drools.org/, accessed October 11, 2018).

Dundar, B., Astekin, M., and Aktas, M. S. 2016. "A Big Data Processing Framework for Self-Healing Internet of Things Applications," in 12th International Conference on Semantics, Knowledge and Grids (SKG), Beijing, China, August, pp. 62–68. (https://doi.org/10.1109/SKG.2016.017).

Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., and Mock, M. 2017. "Issues in Complex Event Processing: Status and Prospects in the Big Data Era," Journal of Systems and Software (127), pp. 217–236. (https://doi.org/10.1016/j.jss.2016.06.011).

Flouris, I., Manikaki, V., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Mock, M., Bothe, S., Skarbovsky, I., Fournier, F., Stajcer, M., Krizan, T., Yom-Tov, J., and Curin, T. 2016. "FERARI: A Prototype for Complex Event Processing over Streaming Multi-Cloud Platforms," in Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, New York, NY, USA: ACM, pp. 2093–2096. (https://doi.org/10.1145/2882903.2899395).

Giatrakos, N., Artikis, A., Deligiannakis, A., and Garofalakis, M. 2017. "Complex Event Recognition in the Big Data Era," Proceedings of the VLDB Endowment (10:12), pp. 1996–1999. (https://doi.org/10.14778/3137765.3137829).

Hadar, E. 2016. "BIDCEP: A Vision of Big Data Complex Event Processing for near Real Time Data Streaming Position Paper - A Practitioner View," in CEUR Workshop Proceedings (Vol. 1600).

Kafka. 2018. "Apache Kafka Homepage." (https://kafka.apache.org/, accessed July 3, 2018).

Kagermann, H., Wahlster, W., and Helbig, J. 2013. "Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0. Final Report of the Industrie 4.0 Working Group," acatech - National Academy of Science and Engineering, April, p. 82.

Krumeich, J., Jacobi, S., Werth, D., and Loos, P. 2014. "Big Data Analytics for Predictive Manufacturing Control - A Case Study from Process Industry," in 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, June, pp. 530–537. (https://doi.org/10.1109/BigData.Congress.2014.83).

Leavitt, N. 2009. "Complex-Event Processing Poised for Growth," Computer (42:4), pp. 17–20. (https://doi.org/10.1109/MC.2009.109).

Luckham, D. C. 1996. "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events.," Stanford, CA, USA: Stanford University.

Luckham, D. C., and Vera, J. 1995. "An Event-Based Architecture Definition Language," IEEE Transactions on Software Engineering (21:9), pp. 717–734. (https://doi.org/10.1109/32.464548).

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. 2007. "A Design Science Research Methodology for Information Systems Research," Journal of Management Information Systems (24:3), pp. 45–77. (https://doi.org/10.2753/MIS0742-1222240302).

Saenko, I., Kotenko, I., and Kushnerevich, A. 2017. "Parallel Processing of Big Heterogeneous Data for Security Monitoring of IoT Networks," in 2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), St. Petersburg, Russia, March, pp. 329–336. (https://doi.org/10.1109/PDP.2017.45).

Tableau. 2018. "Tableau - Business Intelligence and Analytics Software." (https://www.tableau.com/, accessed October 16, 2018).

Tawsif, K., Hossen, J., Emerson Raja, J., Jesmeen, M. Z. H., and Arif, E. M. H. 2018. "A Review on Complex Event Processing Systems for Big Data," in 2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP), Kota Kinabalu, Malaysia, March 26. (https://doi.org/10.1109/INFRKM.2018.8464787).

Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., and Ganguli, D. 2014. "Druid: A Real-Time Analytical Data Store," in Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Utah, USA: ACM, pp. 157–168.

Zhang, P., Shi, X., and Khan, S. U. 2018. "QuantCloud: Enabling Big Data Complex Event Processing for Quantitative Finance through a Data-Driven Execution," IEEE Transactions on Big Data, pp. 1–13. (https://doi.org/10.1109/TBDATA.2018.2847629).